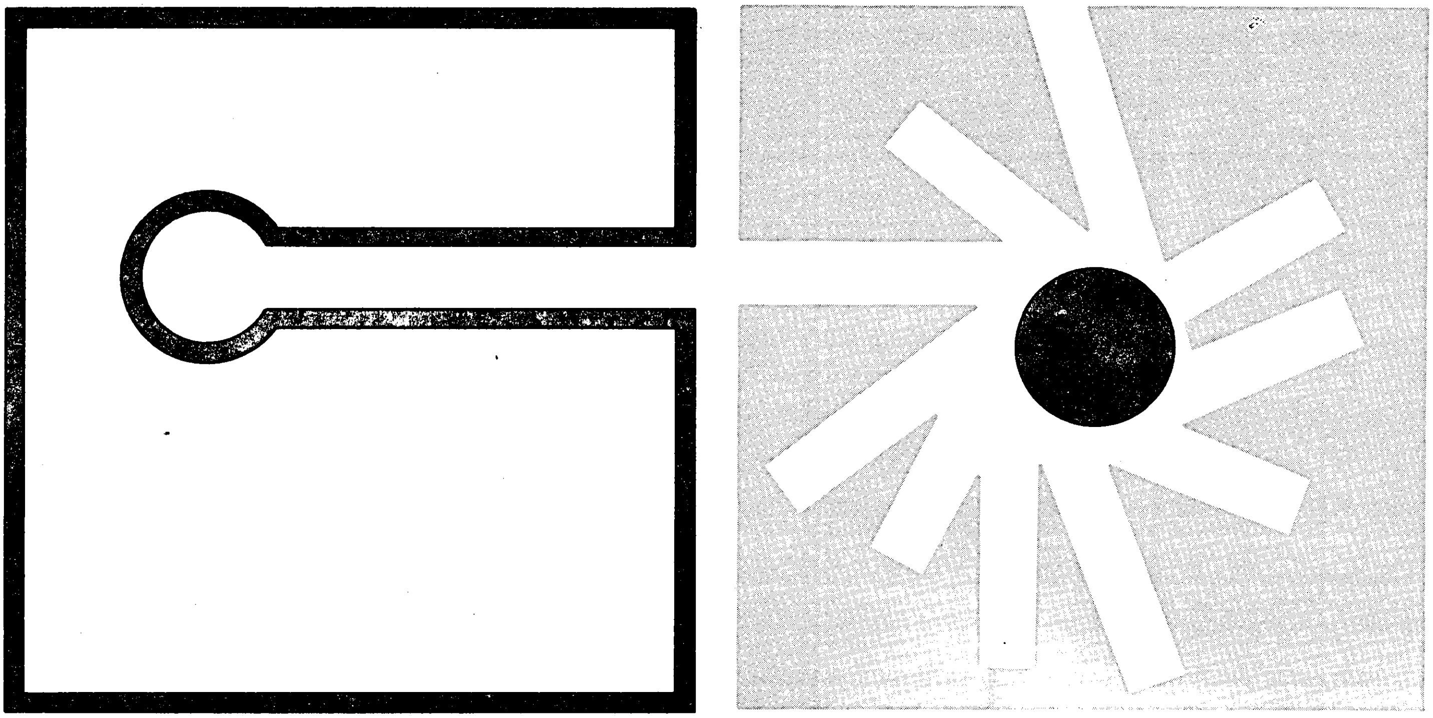
CONTROL DATA
CORPORATION

CYBERDATA™

OPERATING SYSTEM

Reference Manual



REVISION RECORD

REVISION	DESCRIPTION
A	Manual released
7/15/75	

Publication No.
22263100

Address comments concerning this manual to:

Control Data Corporation
Publications Department
Peripheral Subsystems Division
2200 Berkshire Lane
Plymouth, Minnesota 55441

or use Comment Sheet in the back of this manual.

Contents

Figures

Number Title Page

Tables

SECTION 1

INTRODUCTION

Introduction

OPERATING SYSTEM

The CDC® CYBERDATA™ Key-to-Disk System is controlled by a 1784 Computer System. The 1784 Computer is a small, high-speed digital processor which performs a variety of on-line control, real-time data acquisition, and batch job processing applications.

FEATURES

The 1700 CYBERDATA Operating System (COS) provides the following features:

- Reentrant monitor and request processors
- System modularity
- Interrupt handling
- Input/output drivers
- Maximum I/O error detection and recovery by drivers
- Engineering file for logging I/O errors
- Batch job processing
- Relocatable binary linking loader
- On-line system modification
- System maintenance and utility routines
- Macro assembler

MONITOR

The monitor is the real-time executive program for 1700 COS. It serves as an interface between all programs and the hardware. The monitor schedules the use of the central processor and the I/O equipment via various programs on a priority basis.

Real-time programs that must be executed in a time limit run at high priority levels (highest — 15). Non-real-time programs run at the lowest priority levels (lowest — 0).

A program-protect feature is used to segregate central memory into two areas: protected core and unprotected core. On-line programs, the monitor, the job processor, the library editing program, and the recovery program run in protected areas of core. Batch programs (job processing) run in unprotected core. This protect feature ensures that errors in the unprotected programs do not destroy the on-line system. Unprotected programs, however, may use protected routines, such as I/O Drivers, through requests to the monitor.

Manual Interrupt Processor

The manual interrupt processor responds to interrupts generated via the MANUAL INTERRUPT button. The message **MI** is issued on the system comment output device. At this time, one of the following commands may be entered on the system comment input device:

Command	Purpose
*BATCH	Initiates job processing
*	Is a *do-nothing* command
*R,LU	Restores a failed logical unit
DB	Initiates the on-line debug package (ODEBUG)
DX	Terminates on-line debug package I/O operations
EF	Prints the system engineering file information
EFLU	Prints the system engineering file for one logical unit
EFMM	Prints the core-resident mass memory information for the engineering file
VL	Initiates the CYBERDATA package
VX	Deactivates the CYBERDATA package
VLOS†	Initiates VLOS, which is used to set CYBERDATA activation to on/off

†These commands are explained in detail in appendix C.

VLBG† Is used to determine the amount of allocatable memory that CYBERDATA requests for table, buffer, and mass resident program use

VLST† Lists core allocation for CYBERDATA

VLTP† Allows modification of terminal types in a CYBERDATA system

SP Puts the control console into a supervisor mode, at which time any CYBERDATA supervisor command may be entered (SP is recognized only when CYBERDATA is active)

JOB PROCESSOR

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

The job processor initiates and supervises the following programs that are running in or utilizing unprotected core:

- Macro assembler
- COSY
- Off-line object programs
- Library editing
- I/O utilities
- System and program maintenance routines

The job processor is initiated by input through the console typewriter. Subsequent batch control may come from the teletypewriter console, or it may be assigned to standard input. The operator has the following options:

- Specify the input device for control statements
- Call the relocatable binary loader
- Instruct the loader to load a specified program
- Start execution
- Reassign standard I/O devices for job processing
- Reassign standard I/O devices for COSY
- Execute programs after loading, or load and execute by using load-and-go
- Rewind and/or unload magnetic tape units
- Temporarily suspend job processing for operator intervention
- Call the library editing program

†These commands are explained in detail in appendix C.

SECTION 2

CYBERDATA SYSTEM FLOW

CYBERDATA System Flow

START-UP

The CYBERDATA Key-to-Disk System is a table-driven system which runs under the CYBERDATA Operating System (COS). It is started from an inactive condition by an autoload operation. This causes the Autoload program, which resides on the mass memory library unit, to be loaded into core and executed. The Autoload program function is reading the core-resident system image from the library unit to core and then transferring control to the restart routine. The restart routine performs the following tasks:

- Sets up the allocatable core area table, which is located in SYSDAT
- Protects and unprotects all appropriate core locations
- Sets up the initial overlay program length for LIBEDT and the protect processor in their respective system directories
- Starts the system timer and schedules the diagnostic timer and time-of-day modules
- Issues a system identification message on the comment device
- Requests an entry of the current date and time
- Transfers control to the CYBERDATA initialization program VLSTRT

VLSTRT asks for the required configuration; i.e., number of stations, average record length, and then proceeds to read the necessary tables from disk and allocate core for buffers. VLSTRT allocates one buffer which is then able to accommodate disk routines, data buffers, format buffers, and search and interrogate buffers. This area must be at least 2K. Power failure recovery is performed at the same time as general housekeeping, such as reconstructing system tables, according to the system's last status. Idle terminals are then started up with the following message:

*****CYBERDATA SYSTEM*****

INTERRUPT PROCESSING

The terminal controller sends a periodic interrupt to the system. When the interrupt is received, VLINP performs the input for each terminal and stores each character in the terminal (software) input buffer (TIQ). After the input cycle is complete, output is performed on each terminal for which display is pending. Up to two characters or address codes per terminal is sent per interrupt cycle. For the self-scan (970-32 Key Entry Station) terminals, an image of the current display is kept in memory, and only new characters are sent. For the cathode-ray-tube (CRT) display, blocks of output (ALD) are strung together by the display driver, and all characters on the thread are sent. When the output process is over, the terminal processing routine (MPC) is scheduled.

PROCESSING THE TERMINALS

The interrupt handler schedules the terminal processing routine MPC to begin searching for data input at terminal 0. The MPC routine checks each input queue for a character which is to be processed. When a character is found, that terminal is earmarked for processing. Before processing is begun, the next terminal that has input is located and is scheduled for processing at the next opportunity. This system ensures that all terminals are serviced in turn, even though operators may be keying at varying rates.

Having decided which terminal is to be processed, a character is extracted on a first-in/first-out (FIFO) basis from its buffer. The character is converted according to format data type and keyboard type, and a decision is taken as to where to continue processing. The choices are:

- Entry mode
- Verify mode
- Function processing
- Interrogate mode character string collection
- Terminal Idle mode

If data entry is being performed, basic numeric and alphabetic validation checks are made by MPC.

When all the terminals have been processed for this cycle, a special program is entered for memory usage optimization. As mentioned, one large buffer (FRP) is allocated in memory for buffers and disk routines. Each block in this buffer has a header which contains the length of the block, how many devices are using it, and whether or not it may be moved. This routine, which is part of the core allocation module (CAM), checks for unused blocks and consolidates the buffers. Some routines which are used frequently have time control and are eliminated from memory only if the buffer is full and a request for space is received. When CAM has finished its task, and the next interrupt has not yet been received, the CPU time remaining until the next interrupt is not used by the system (unless an outstanding I/O request for one of the terminals is completed).

IDLE MODE

Characters that are received before a terminal is signed on for data entry, verification, or supervision are transposed directly to the display as lowercase characters. Function keys show as the symbol @except for the interrogate (INT) key, which initializes data entry and verification and enables a terminal to be used as a supervisor station.

INTERROGATE MODE PROCESSING

When the interrogate function (INT) key is pressed with the terminal in the idle state, the display is cleared, and the characters are accepted in a string which may be terminated by the release (REL) key for a normal exit or by the cancel (CNCL) key. When the cancel key is pressed, the terminal reverts to its former state. When the release key is pressed, control is given to the interrogate function which analyzes the characters that have been entered. From the idle state, only the following three commands are valid:

- ENT — Sign-on for data entry
- VER — Sign-on for data verification
- SUP — Supervisory mode request for this terminal

Any other command causes an error message.

Control is then passed to the required routine to validate the remainder of the character string. For the commands ENT and VER, if the remaining parameters are valid, data entry or verification may begin.

ENTRY MODE FOR DATA ENTRY

The operator performs the sign-on procedure, which consists of pressing the INT function key and entering a job name, batch number, operator number, and an optional autosequencing number. The routine ENTRY checks that the job name exists; i.e., it has been defined by an EJB command, that the batch number of the specified job does not presently exist in the system, and that the operator number and autosequencing number are within the specified limits. Then a buffer is requested for the data, and a track on disk is requested for intermediate storage. Initial values are set up in the terminal's active terminal table (ATT), and the display is set up for data entry.

The operator must now enter a valid format or document number so entry may begin. When the first character of the record is entered, the beginning of record processing (BORENT), which consists of setting up disk addresses in the terminal's record buffer (TRB) and its ATT, is performed. Each function determines for itself whether it has to perform BORENT (SKIP, REL, DUP, etc.) or not (READ, INT, FMT SEL, etc.). Autosequencing is performed, and any other automatic fields are processed in order to enable all pointers to the first field in the record to be manually entered. Control then passes to the entry mode module (EMM), which increments the record and field pointers, stores the character in the TRB (SBYTE), and displays it. EMM checks for the end of field (field length is defined by the format). It the end of field has not been reached, exit is made, and no more processing is performed for this terminal until the next character is entered.

When the end of field is reached, validation tests are performed (VALDTE). If the data fails one of these tests, an error is displayed, and flags are set to allow (after the RESET key is pressed) only replace field (REP FLD) or error override (ERR OVR) to be pressed. Replace field sets the pointers back to the beginning of the field and updates the display. Data entry may then continue as normal. Error override flags the field which is in error and continues processing. If automatic flagging was specified in the format, the error override routine is called immediately, and the operator is not made aware of any error condition.

After validation, EMM tests for an end-of-record condition. If end of record has been reached, control is passed to EORENT for end-of-record processing. This consists of resetting record pointers, writing the current record to disk, updating the number of storage words left on the current track, and handling document control (automatic format call), if it is required.

If end of record has not been reached, the ATT pointers are updated to point to the next field which is to be keyed, and processing is terminated. When all the records in the batch have been entered, pressing the INT key followed by keying EOB calls the end-of-batch (EOB) processor. This EOB routine performs the batch-balancing test by checking that all seven counters assigned to the batch are zeros. If an out-of-balance condition is displayed, the operator may override the error; and the batch is marked as unbalanced in the active batch table (ABT), or the end-of-batch processing may be canceled by using the cancel (CNCL) key, and the batch may be examined in Read mode. The EOB routine updates the entry for this batch in the ABT, storing the status of the batch (unbalanced, flagged) and what verification is required, if any. A 16-word statistics record is then output to a special area on the disk for subsequent processing, the ATT is written on the disk for use by the VERIFY routine, and the terminal is returned to the Idle mode.

VERIFY MODE FOR DATA VERIFICATION

The sign-on sequence for the VERIFY routine is similar to the sequence used for the ENTRY routine except that no autosequence number may be entered. The VERIFY routine checks that the job name exists and that the batch is waiting for verification. A data buffer is requested, and the ATT is set up to include various parameters [such as counter values, disk tracks used, and the next available storage space on disk in case an insert (INS key pressed) is done] obtained from the ATT, which was written on the disk at the end-of-batch condition during Entry mode. The ABT entry is updated, and control is passed to the verify mode module (VMM) where the format is checked to locate the field to be verified. Verification may be performed on the fields that are specified by the formats used, on error-flagged fields, on fields that contribute to unbalanced counters, or on any combination of these three fields. The verification options are defined by EJB. They may be changed by the CBS command (see the CDC® CYBERDATA™ Key-to-Disk System Reference, publication 22262200). Keying in the verify function sets switches in the ATT to indicate which verification options are in force. VMM checks them in the following order:

1. Key verification according to format
2. Sight verification according to format
3. Error-flagged field
4. Field contributing to unbalanced counters

KEY VERIFICATION

When the first record is ready for verification, the format is scanned until a field which meets the specified verification options is found. The display is updated to this field, and exit is made to wait for characters. Each character entered is checked against the corresponding character in the TRB. If the field is left-zero or blank filled, the first nonzero or nonblank character entered will be matched against the first nonzero or nonblank character in the data field. If the character matches, the pointers are updated, and the character is displayed. An end-of-field check is made. If end of field has not been reached, processing is terminated until the next character is entered. If the character does not match the original character entered, a mismatch error occurs.

There are now two courses of action open to the operator:

1. Reenter the original character.
2. Use the correction (COR) key. The correction function enters the new character in place of the original Entry-mode character. Pressing COR sets a switch which causes validation at the end of field and also sets the forced-reverification switch if this option is specified by the format.

At end of field, revalidation is performed whenever a change is made; validation errors are treated as in Entry mode. If forced reverification is required, pointers are set back to the beginning of the field, the display is updated, and the characters in the field are reentered as in Verify mode. At end of record, processing automatically continues with the next record. At end of batch, a flag is set to prevent any characters from being entered.

SIGHT VERIFICATION

Sight verification fields are displayed at the terminal. The operator may use the back space character (BSC) or space (space bar) functions to move through the field. Pressing CONT causes VMM to search for the next verifiable field.

When verification is complete, EOB updates the ABT entry for this batch, including status such as balanced, flagged records, etc. The ATT and a second statistics record are output to disk.

READ MODE

Read mode may be entered from Entry or Verify modes. REDKEY writes the current partial record (unless at end of record) and saves in the ATT all the pointers to the present position. Parameters saved include:

- Disk address of the last record written
- Record count
- Format number

This information is used by the RTB routine to return to the basic mode position from Read mode.

In Read mode, positional function keys may be used to examine the data. Those which move backwards from the current position (BSF, BSR, BOR, DOC) are very similar. The pointers are moved back one field, record, or document; and the new field is displayed (with asterisks preceding it if it is error-flagged). When reading backwards through a batch, a test in the I/O routine checks a bit in the TRB to determine whether the first record has been reached. At this point, no further backspacing is allowed.

The forward functions (space bar, SKIP key, REL key, and DOC) position forward to the next character, field, record, or document which displays the relevant field. Each routine tests separately for end of data, which is defined as follows:

1. In Entry mode, it is the last field entered.

2. In Verify mode, it is the last field of the batch. It is possible to advance over the current field to be verified.

Changing a field in Read mode is accomplished by using the REP FLD key. Pointers are set to the beginning of the field, and the display of the field is cleared. If this field was added to the counters, the old value is subtracted. If the field was flagged, the record is checked for more flags, and the number of records in error is decremented if no other error flags are found. A pseudo entry bit is set in the ATT, and the characters entered are processed by EMM. At the end of the field, validation is performed with errors treated as they were in Entry mode. A switch is set which allows only another REP FLD or COR key to be entered to complete the field. Pressing the COR key initiates a process to check for forced reverification in Verify mode. If forced reverification is required, the pointers are set back to the beginning of the field, and a pseudo verify bit is set in the ATT, which causes the characters that are entered next to be processed by VMM. If reverification is not required, the field is redisplayed from the beginning, and the mode reverts from pseudo entry back to read.

SUPERVISOR CONTROL

The supervisory routines are entered via the COS manual interrupt processor by using the mnemonic SP. The message -SV- appears at the system comment device, and a read request is issued. When input is terminated by a carriage return, control is passed to DSKSUP where the characters are analyzed. If a legal command was entered, the appropriate routine is called from disk to analyze the remainder of the parameters. All standard errors are processed by REJSUP, which takes the contents of the A register on entry as the index to an error message table. The supervisor input and output units may be assigned to any suitable device. The current logical unit numbers are kept in the supervisor communication area (part of the ATT for terminal 0). After a command has been executed, a new read request is issued.

Some supervisor commands are available to key entry stations through the use of the interrogate function SUP. A special table in DSKSUP determines whether or not the requested command is available to the station which is requesting it.

Whenever a display is ready for a CRT, and the screen is filled, a call is made to the subroutine CONCAN to wait for the CONT or CNCL keys. To abort a printout on the teletype, the manual interrupt SX is used. The manual interrupt processor sets the cancel flag, which is tested after each line printed.

SECTION 3

REQUESTS

Requests

INTRODUCTION

Requests are used in programs to instruct the monitor to perform operations such as reading, writing, loading, and program scheduling (table 3-1). Requests may be written in the assembly language as macro instructions, which the macro assembler converts into calling sequences. Each calling sequence contains an indirect return jump to the monitor entry and a list of parameters.

Table 3-1. Monitor Requests

Request Code	Request Mnemonic	Request Function
1	READ	Normal read
2	WRITE	Normal write
3	STATUS	I/O request status
4	FREAD	Formatted read
6	FWRITE	Formatted write
14	MOTION	Tape motion
16	INDIR	Indirect

ENTRY FOR REQUESTS

Programs make requests of the monitor (NMONI) by calling the monitor entry. The general format of the calling sequence is as follows:

```
RTJ-($F4)   Location in fixed communication region
            containing address of request entry
            processor

   •  )
   •  }     Parameters
   •  )
```

THREADING

Requests to the monitor that use I/O drivers must wait in a queue for processing. These requests are processed on a priority basis (first in/first out—FIFO by priority).

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

INTERRUPT LEVELS AND PRIORITIES

When programs make monitor requests, there are two priority systems which are used; that is, request priority and completion priority. The request priority determines how a request is to be queued with respect to other requests which are already queued — the higher the request priority, the closer it is to the front of the queue. The completion priority defines the level at which execution takes place after a request has been completed; i.e., after input/output is completed, 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 level 0 and level 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 at levels 0 and 1 and by the job processor at 2.

REQUEST DESCRIPTIONS

Requests are explained in the following paragraphs.

READ, FREAD, WRITE, FWRITE Instructions

READ and 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 and FWRITE requests read or write records in a specific format as associated with each device.

The macro format for READ, FREAD, WRITE, and FWRITE requests (1, 2, 4, and 6) is shown in the following example. The parameter descriptions are the same except for the n parameter. Refer to the description of n for an explanation of the difference.

Example:

```
READ   \
FREAD   |
        > lu,c,s,n,m,rp,cp,a,x,d
WRITE   |
FWRITE /
```

INSTRUCTION PARAMETERS

The following list of parameters is defined briefly. A detailed description of each parameter follows immediately after the calling sequence which is macro-generated.

lu —logical unit

c —completion address

s —starting address

n —number of words to transfer

m —mode

rp—request priority

cp—completion priority

a —absolute/indirect indicator for logical unit

x —relative/indirect indicator (affects parameters c, s, and n)

d —part 1 request indicator (absolute parameter addresses)

MACRO CALLING SEQUENCE

The request codes are 1 (READ), 2 (WRITE), 4 (FREAD), and 6 (FWRITE). The calling sequence generated by the macro is shown in figure 3-1.

<table>
<tr><td></td><td colspan="6">15 11 9 8 7 3 0</td></tr>
<tr><td></td><td colspan="6">RTJ-($F4)</td></tr>
<tr><td>0</td><td>0</td><td>d</td><td>rc</td><td>x</td><td>rp</td><td>cp</td></tr>
<tr><td>1</td><td colspan="6">c</td></tr>
<tr><td>2</td><td colspan="6">thread</td></tr>
<tr><td>3</td><td>v</td><td>m</td><td>a</td><td colspan="3">lu</td></tr>
<tr><td>4</td><td colspan="6">n</td></tr>
<tr><td>5</td><td colspan="6">s</td></tr>
</table>

Figure 3-1. Macro-Generated Calling Sequence

FIELD DESCRIPTIONS

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

RC—request code

thread—thread location; used to point to the next entry on the threaded list

v—error code; passed to the completion address in bits 15 through 13 of register Q and set in the request by the system at completion

PARAMETER DESCRIPTIONS

Detailed parameter descriptions for the requests are:

lu—This is the logical unit. It is an ordinal in the physical device table which may be modified by parameter a.

c—This is the completion address which is the address of the core location to which control transfers when an I/O operation is completed. If the completion address is 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 on completion of the requested I/O operation. Use of the (c) option by unprotected programs results in job termination.

Completion routines are operated by threading the I/O requests on the scheduler thread. A 3-bit code in the v field of the fourth word of the request indicates completion status. See table 3-2 for bit positions of the fourth word indicating completion status.

When control is returned to the completion these bits are set in similar positions in the Q register. If less than n words were transferred on a read, the location which follows the last word filled is placed in the last word of the user's buffer.

s—This signifies the starting address which is the address of the first block location to be transferred (see parameter x).

n—The number of words to transfer is given by n.

(n)—The number of words to be transferred is determined by parameter x.

Table 3-2. Bit Positions

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

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

NOTE

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

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

Macro

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

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

Coding

0 —This stands for binary.
1 —This refers to ASCII.

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

cp—The completion priority (15 through 0) is the level at which the sequence of code specified by parameter c is executed. It is automatically one for unprotected requests.

a—This signifies the absolute/indirect indicator for the logical unit.

Macro

blank—The first parameter (lu) specifies the logical unit.

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

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

Coding

0—lu is a logical unit number.

1—lu is a signed increment (+1FF); however, it is not allowed if d = 1.

2—lu is a core address which contains the logical unit number.

x—The relative/indirect indicator parameter affects parameters c, s, and n as shown. Because of the wraparound feature, computed addresses may be located before or after the parameter list.

The x parameter affects the c parameter as follows:

- If (c) is indirect, x is meaningless; c represents an index to the system directory.
- If x is 0 or blank and c is direct, c is the completion address.
- If x is not 0 and is not blank and c is direct, c is a positive increment added to the address of the first word of the parameter list to form the completion address.

The x parameter affects the s parameter† as follows:

- If x is 0 or blank and s is direct, s is the starting address. If a request is made for mass memory, the mass memory address follows the request.
- If x is 0 or blank and (s) is indirect, s is a core location which contains the starting address. If a request is made for mass memory, the mass memory address follows the core location which contains the starting address.
- If x is not 0 and is 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 a request is made for mass memory, the mass memory address follows the request.
- If x is not 0 and is 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 which contains another positive increment. If the request is for mass memory, the loction containing the second increment is immediately followed by two words which contain the mass memory address.

  The mass memory address format is shown in figure 3-2.

15	14 … 0
Most-significant bits of MS address (msb)	
0	Least-significant bits of MS address (lsb)

Figure 3-2. Mass Memory Address Format

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.

The x parameter affects the n parameter† as follows:

- If n is direct, x is meaningless; n is the length of the block to be transferred.
- If x is 0 or blank and (n) is indirect, n is core location containing the block size.
- If x is not 0 and is not blank and (n) is indirect, n is a positive increment added to the address of the first word of the parameter list to obtain the address of the location containing the block size.

d—The part 1 request indicator parameter indicates that the request requires the use of 16-bit address arithmetic.

0 or blank—The preceding description of the parameter applies.

1—x is ignored; c, n, and s parameters are interpreted as the 16-bit absolute address. lu is processed the same as for d = 0, but parameter a may not be set to R.

INDIR Request

The INDIR request (16) allows indirect execution of any other request as determined by the parameter list referenced by p.

FORMAT

The macro format is:

INDIR p,i

PARAMETER DESCRIPTIONS

Parameter descriptions for the INDIR request are:

p—The address of the first word of the parameter list of any other request; p must not be enclosed in parentheses.

i—This is the indicator for the request used.

0 or blank—INDIR has no request code. The calling sequence generated by the macro is shown in figure 3-3.

15	14 … 0 (11, 7, 3, 0)
RTJ-($F4)	
1	p

Figure 3-3. INDIR Macro Calling Sequence (0 or Blank)

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

This form is useful only when the address of the request to be executed is at an address below $8,000.

1—INDIR has a request code of 16. The calling sequence generated by the macro is shown in figure 3-4.

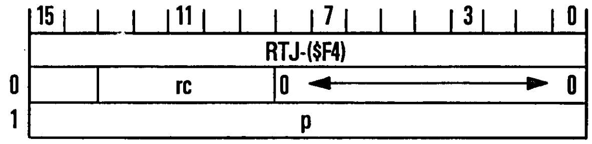


Figure 3-4. INDIR Macro Calling Sequence (1)

MOTION Control

The MOTION control request, (14), is used to control motion and end-of-file processing.

FORMAT

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

PARAMETER DESCRIPTIONS

Parameter descriptions for the MOTION control request are:

lu—Logical unit

c—Completion address

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

dy—Magnetic tape density selection is as follows:

0—no change
1—800 bpi
2—556 bpi
4—1600 bpi

External rejects will result when an illegal density selection is attempted.

rp—Request priority

cp—Completion priority

a—Absolute/indirect indicator for the logical unit.

x—Related only to the completion address.

d—The part 1 request indicator indicates that the request requires 16-bit address arithmetic.

0—all parameters are processed as described.

1—this indicates the part 1 request; c is a 16-bit absolute address and must not equal R for the a parameter.

m—Mode

A—ASCII

B—binary

MOTION Request Calling Sequence

The MOTION control request code is 14, and the calling sequence generated by the macro is shown in figure 3-5.

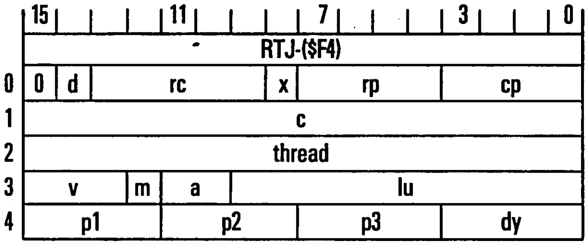


Figure 3-5. MOTION Request Calling Sequence

MOTION Field Description

The field descriptions for the calling sequences are:

rc—Request code

thread—The thread location used to point to the next entry or the threaded list

v—Error code setting

MOTION Parameters for Magnetic Tape

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

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

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

r—repeat function indicator must equal R. The parameter must also equal R.

p—motion code

n—number of times to be executed; not to exceed 4095

0—null parameter

MOTION Coding Sequence

The coding sequence generated is the same as figure 3-5 except for the last word, which is generated as shown in figure 3-6.

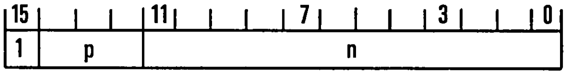


Figure 3-6. MOTION Coding Sequence

The field description for the calling sequence is:

1—indicates the request can be repeated

MOTION Request Macros

The following macros can also be used for MOTION requests. Each macro performs only one motion-type request. Refer to table 3-3 to determine the action taken by the driver.

BSR*†	lu,a,n,c,P	(backspace record)
EOF*	lu,a,n,c,P	(write file mark)
REW*	lu,a,n,c,P	(rewind)
UNL*	lu,a,n,c,P	(rewind unload)
ADF*	lu,a,n,c,P	(advance file forward)
BSF*	lu,a,n,c,P	(backspace file)
ADR*	lu,a,n,c,P	(advance record forward)

MOTION Request Parameters for Additional Macros

These parameters are as follows:

lu—logical unit number of device

a—absolute/indirect/relative indicator for logical unit

blank—lu is the actual logical unit number.

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

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

n—number of iterations; if it is blank, it is assumed to be 1 (not to exceed 4095).

c—completion address; if the macro call terminator is an *, completion is relative (all that is required is the label name). If the macro call terminator is a blank, the completion is absolute. If c is left blank, there is no completion.

P—priority level defines both the request and completion priority; if it is left blank, priority is zero.

NOTE

All parameters are optional and may be left blank except lu.

Examples of MOTION Control Requests

The following parameters are common to the examples:

- NEXT—completion address
- 6—logical unit of magnetic tape
- 10—logical unit of card punch

†An asterisk (*) specifies relative completion address; if blank, absolute completion (Macro computes the relative address constant).

Table 3-3. Driver Action for MOTION Request Parameters p_1, p_2, p_3

Refer to the key below the table for a description of the column headings.

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

†Key

MT–Magnetic tape
CR–Card reader
LP–Line printer
TTY–Teletypewriter

MSD–Mass storage driver
PTD–Pseudo tape driver
CD–COSY driver

‡Assumes magnetic tape physical device

- MT—program location containing a 6
- $FA—low-core location containing standard binary output device

The following example is a backspace macro. The macro includes a relative location containing the logical unit number, backspace 3 records, a relative completion address, and a request and completion priority of 3.

```
BSR*        MT,R,3,NEXT,3
```

The following sample end-of-file macro has the actual logical unit number, write one end-of-file, zero completion and a priority of 0.

```
EOF         10
```

The following end-of-file macro is the same as above except that the logical unit number is in a low-core location.

```
EOF         $FA,I,,,0
```

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 LOG1A table are illegal.

Illegal Control Information

Requests must not contain illegal control information (any information that can cause destruction of part of the system). A read into protected core or a write into system areas of mass storage, for example, is illegal.

Busy Requests

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

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. The input/output devices and core location are as follows:

Device	Core Location
Standard input device†	$F9
Standard binary output device†	$FA
Standard print output device†	$FB
Output comment device	$FC
Input comment device	$FD
Mass storage scratch	$B3
Mass storage library	$C2

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

SECTION 4

DRIVERS

Drivers

INPUT/OUTPUT REQUEST FORMAT

READ/WRITE calls request processors to transfer data between a specified input/output device and core memory. Word count which is specified in the request determines the end of the transfer.

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

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

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

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 READ, WRITE, and MOTION requests passed to the monitor by a user program.

Whenever a program requires input or output of the data it is processing, it makes a monitor request to affect the desired transfer. The monitor queues the request for processing by an I/O driver.

When a request is queued, the request processor determines whether or not the driver is already busy. 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 MOTION request processor.

Completion Routines

The completion address specified in a parameter list is scheduled when the I/O operation has been completed. On 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. The Q register is negative (bit 15 = "1") if an error occurred and should be tested by the completion routine. The original state of registers at the time of the monitor request is not preserved. The priority level is that which is specified as the completion priority in the monitor request parameter list.

Completion routines which are in unprotected core are always executed at priority level 1.

TELETYPEWRITER DRIVE

The discussion of the teletypewriter drive defines the operation of the following device driver:

D1711—1711/1712/1713/713 Keyboard Driver

Teletypewriter Keyboard Requests

Four types of requests (READ, WRITE, FREAD, and FWRITE) are honored from the keyboard. Each request specifies the starting address location in core that is being read into or written from, the number of words, and the completion address. All data is in ASCII format.

READ REQUEST

The number of words in the READ request is filled, starting at the specified core location. Two characters fill one word; the first character is put into the upper half of the word. Bit 7 of each character is an even parity bit and is set to "0" after it is checked and before it is packed. If the parity bit is incorrect, a hardware error is indicated.

READ Request Coding Sequence

If no words are specified, only one character is read into the upper half of the specified core location. The lower character is filled with binary ones. The calling sequence format is shown in figure 4-1.

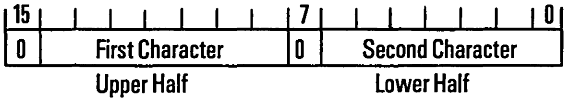


Figure 4-1. READ Request Calling Sequence

FREAD REQUEST

Words in core are filled, starting at the specific core locations and continuing until the number of words specified is filled or a carriage return is encountered. Two characters are packed in each word. Bit 7 of each character is interpreted as an even parity bit before being cleared to "0". Line-feed characters are ignored. If cancel character is encountered, characters are passed and no information is stored until a carriage return is detected.

The request is then repeated from the beginning. If a carriage return is not encountered before the specified number of characters are read, characters are passed until a carriage return is detected. A carriage return before the number of words specified is read constitutes a short read.

WRITE REQUEST

The number of specified words is printed, starting at the specified core location. Each word causes two characters to be printed with the upper half being printed first. If no words are specified, only one character is printed from the upper half of the specified core location. If an ASCII end-of-text character ($03) is encountered, the request is terminated at that point regardless of the number of words specified.

FWRITE REQUEST

The FWRITE operation is the same as that of WRITE request except that before any words are printed, a carriage return and line-feed function are executed by the teletypewriter driver.

MOTION REQUEST

A write end-of-file request to the keyboard is honored by executing a top-of-form function. All other MOTION request parameters cause no action with the normal completion of the request.

MANUAL INTERRUPT

A manual interrupt is caused by pressing the MANUAL INTERRUPT button on the teletypewriter. If a manual interrupt is detected by the teletypewriter driver, the manual interrupt processor (MINT) is entered.

Error Conditions

The driver recognizes the following errors:

- Internal reject on input or output instructions
- External reject on input or output instructions
- Failure to interrupt
- Alarm
- Parity error on input (1711/1712/1713)
- Checksum (1713)
- Lost data (1712/1711/1713)

These errors are considered irrecoverable by the driver. The driver sets the error fields in the physical device table and the error parameter in the request. On entry to the completion program, the Q register is negative, which indicates an irrecoverable error to the user.

CARD READER DRIVER

The following discussion defines the operation of the D17293—1729-3 Card Reader Driver.

Card Reader Requests

The card reader requests are:

- READ binary
- READ ASCII
- FREAD binary
- FREAD ASCII

READ BINARY

The calling sequence specifies the number of words in core to be filled, starting at a requested beginning address. Card columns are packed, leaving no unused bits. After reading in binary mode, the buffer appears as shown in figure 4-2.

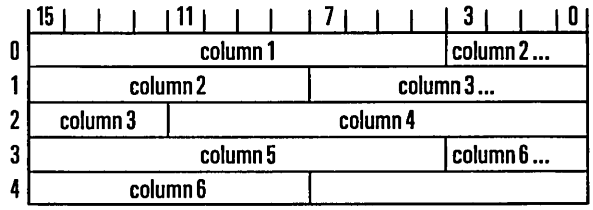


Figure 4-2. READ Binary Buffer Formatting

If no words are requested, only the first column of the card is read. The unused bits are set to "1"s; the remainder of the card is unavailable. If a read ends in the middle of a card, motion continues, but the unread portion of the card is unavailable.

READ ASCII

Words in core are filled, starting at a given address until the number of requested words is filled. The READ proceeds in the same manner as for READ binary except that each column is converted from Hollerith code to a 7-bit ASCII equivalent before being stored. These ASCII characters are stored in two per word, leaving bits 7 and 15 as a "0". If no words are being read, one column is read, and the character is placed in the upper half of the word with "1"s placed in the lower half. Refer to the FREAD ASCII discussion on page 4-4 for conversion code information.

FREAD BINARY

Although a formatted read operation may be specified with the request as binary or ASCII, the format of the card actually determines the mode. If column 1 of the card contains a 7/9 punch, it is read as a formatted binary record; otherwise, it is read as a formatted ASCII record, regardless of the mode specified in the request.

First Card for FREAD Binary

The first binary record format card is shown in figure 4-3.

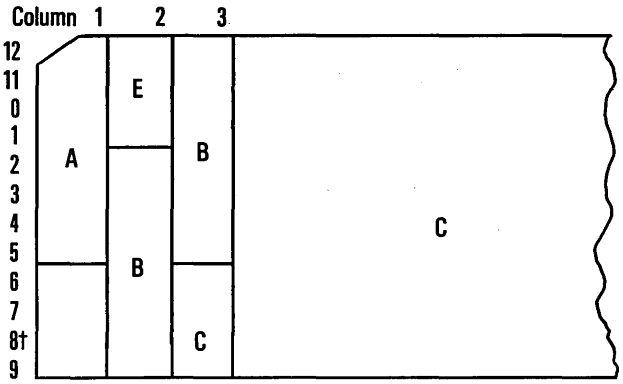


Figure 4-3. First Binary Format Card for FREAD Binary

In figure 4-3, each area is defined as follows:

A—sequence number (column 1, rows 12 through 5)

B—complemented record length (column 2, rows 2 through 9; column 3, rows 12 through 5 on first card)

C—data (first card starts in column 3, row 6; other cards start in column 2, row 2)

D—16-bit checksum (follows last data word of a record)

E—reserved (column 2, rows 12 through 1; blank unless checksum override is indicated in column 1; overriding the checksum is not recommended)

Subsequent Cards for FREAD Binary

Subsequent binary record format cards are as shown in figure 4-4.

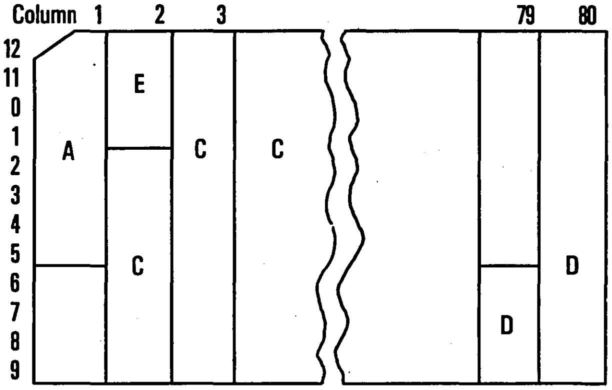


Figure 4-4. Subsequent Binary Format Cards

The lettered areas indicated in figure 4-4 are defined as follows:

A—sequence number (column 1, rows 12 through 5)

B—complemented record length (column 2, rows 2 through 9; column 3, rows 12 through 5 on first card)

C—data (first card starts in column 3, row 6; other cards start in column 2, row 2)

D—16-bit checksum (follows last data word of a record)

E—reserved (column 2, rows 12 through 1; blank unless checksum override is indicated in column 1; overriding checksum is not recommended)

†If row 8 in column 1 is punched, the driver ignores the checksum.

SEQUENCE NUMBERS FOR FREAD BINARY

All cards in a record must have the sequence numbers in order. If a record requires multiple-card storage, and a sequence error is detected on any card in the record; the error is fatal, and the alternate device handler is entered.

Sequence numbers between records are handled as follows:

- If the number is out of sequence, two actions are possible: 1) the operator resequences the cards, rereading the out-of-sequence card—respond with RP to sequence error and 2) the operator wishes to read the cards out of sequence—reread the out-of-sequence card twice and respond with RP to the sequence error.

- If the number of words requested is less than the length of the record, cards are passed with no data being transferred until the entire format record is passed.

- If the number of words requested is greater than the length of the record, data transfer ceases at the end of the record, and no further cards are read for that request.

Row 8 of column 1 on a formatted binary card is the checksum override bit. When the driver detects this condition, the checksum of the card is ignored. The following example illustrates how this function could be used.

Example:

Assume that the user is loading a binary deck, and the card reader jams and damages a card. The user is successful in duplicating the card using the LIBEDT *T processor. Now, however, the sequence number of the card is incorrect. This can be corrected by punching the sequence number on a keypunch machine. This, in turn, causes the checksum to be incorrect. The user punches the checksum override bit, which allows the card to be read properly.

FREAD ASCII

Columns are read in ASCII mode until either one entire card is read or the number of words requested is filled, whichever occurs first. If the number of words requested is depleted prior to reading one card, the remainder of the card is unavailable, and the read operation is in READ ASCII mode.

If a binary card is read when a FREAD ASCII request is specified, either the card is read in binary (if it is the first card of a record) or a 7/9-punch error occurs (if it is not the first card).

Hollerith-to-ASCII Code Conversion

The Hollerith code conversion to ASCII code can be done by using ASCII63 (026 Keypunch type) or ASCII68 (029 Keypunch type). A table program is appended to the driver for this conversion:

- CR026—ASCII63 conversion

- CR029—ASCII68 conversion

EOF Processing and MOTION Requests

The 1729 Card Reader Driver provides the capability for handling an end of file (EOF). If an end-of-file record is detected by the driver during input, the drive offsets the record, sets bit 11 in word 12 of the physical device table, and completes the request with an error code. Control is given to the caller's completion routine with the error code set in bits 15 through 13 of the Q register. An end of file is a card with column 1 punched with the configuration set into bits 11 through 0 of word 16 of the physical device table. Normally, it is a 6/7/8/9 punch; therefore, PHYSTB word 16 contains $000F_{16}$. The MOTION request skip file forward (parameter code 5) is honored by the driver. All other MOTION requests cause no action.

Error Conditions

All errors detected by the driver are caused by equipment malfunction or improper card decks. The following conditions are detected:

- Internal or external reject to any INP or OUT command

- Alarm interrupt

- Illegal Hollerith punch detected during a read ASCII

- Data interrupt after column 80 on read

- EOP interrupt before column 80 on read

- Incorrect checksum at the conclusion of the read

- Pre-read error

- Incorrect sequence number

- Bit 15 of the complemented length is not "1"

- 7/9-punch error

Refer to appendix E for I/O error codes and descriptions.

CARTRIDGE DISK DRIVE

Cartridge disks utilize a single fixed disk and a single removable disk in a cartridge case. Both disks have two recording surfaces. The disks are referred to as disk 0 and disk 1. They are individually addressed by the hardware controller and are differentiated by a single bit designator in the file address word. Autoload is always loaded from disk 0, which is ordinarily the removable disk whereas disk 1 is usually the fixed disk. Through the use of a toggle switch on the maintenance panel, disk addressing may be reversed. When this toggle switch is used, disk 0 becomes the fixed disk, and disk 1 is the removable disk. Note that Autoload is still executed from disk 0.

Software users reference the entire cartridge disk drive as a single logical unit with disk 0 containing the lowest sector address and disk 1 containing the highest sector address. The disk which is referenced depends on the toggle switch position for disk addressing.

Data Transfer Request Formats

Execution of a data transfer request transfers n words from mass storage (READ/FREAD) or to mass storage (WRITE/FWRITE), starting at any first word address indicated by s and mass storage address indicated by msa. If n is zero, one word is transferred. No data formatting is involved since corresponding core and mass storage locations contain identical 16-bit images. The disk driver data transfer request differs from conventional requests in that the two-word mass storage address must be included. To accommodate this, two formats are provided. All parameters are described in section 3.

The first request format is consistent with normal (x parameter not set) requests by providing a seven-word format as defined in section 3. If the x parameter is set, this indicates an indirect reference to the first word address increment contained in the s parameter. This positive increment is added to the address of the parameter list to form the address of a location which contains another positive increment. The second increment is added to the address of the parameter list to obtain the starting address. This second increment is followed immediately by two words which contain the mass storage address (msa) and must comply with the first request format.

If parameter x is zero, both s and s" are absolute addresses; otherwise, they are 15-bit positive increments which are added to the address of the first request parameter (word 0) to form absolute addresses. Control is returned to the location which follows word 5 after the request is made. The first request format appears as shown in figure 4-5.

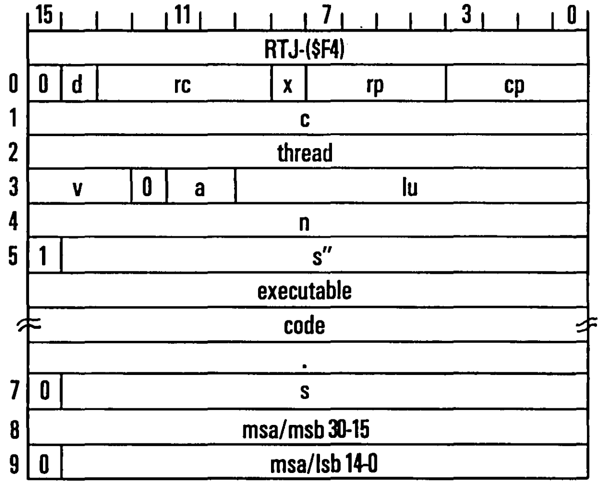


Figure 4-5. First Data Transfer Request Format

The second request format (figure 4-6) adds two additional words which contain msa in line with the conventional data transfer format, identified by a direct reference to s (bit 15 = "0").

The conventions defined previously (section 3) for s in relation to x and d also apply here. Control is returned to the location following word 7 after the request is made.

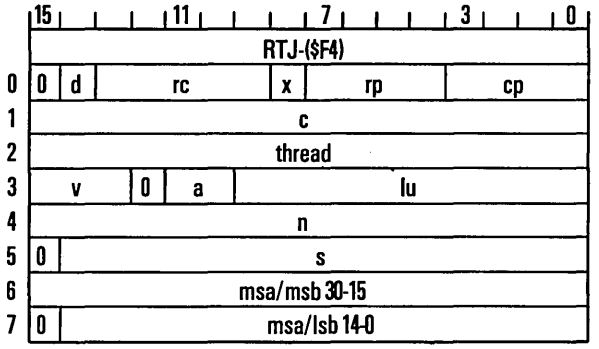


Figure 4-6. Second Data Transfer Request Format

Disk Requests

The disk driver processes requests made by user programs for data transfer to mass storage (WRITE/FWRITE) and from mass storage (READ/FREAD). The driver also provides a program overlay capability and handles the transfer of mass storage resident system directory programs into core as the result of a SCHDLE request (mode has no meaning).

The number of words specified in the calling sequence is transferred to or from core, beginning at the specified starting address and sector number. Sectors are read or written sequentially until the requested number of words has been transferred. If no words are requested, the driver transfers one word to or from core.

READ/WRITE REQUESTS

READ and WRITE requests provide the ability to simulate the word address by allowing the msa to be any word address in the size range of the disk. The word address is converted by the driver to sector and word in the sector by dividing by 96.

READ Request

The READ request fills core, starting at a specified address, with the specified number of words. If no words are requested, one word is transferred. Transfer is initiated from the disk word address which is specified by the most-significant bits (msb) and the least-significant bits (lsb) of the request (lsb is a 15-bit value). A carry into bit 15 of lsb should be treated as an overflow condition, and msb should be incremented by one. The following is an example of a READ request where C is the completion address:

```
READ      8,C,BUFFER,15,B,4,4,,1

 .

 .

 .

ADC       $1,$6D59

JMP-      ($EA)

BSS       BUFFER(15)
```

As a result of this request, 15 words are read from logical unit 8 (disk), starting from disk word address $1,6D59_{16}$ (figure 4-7). The words are read into core, starting from the first-word address buffer. Disk word address $1,6D59_{16}$ is the same as sector 632_{10}, word 88_{10} (divide $0000ED59_{16}$ by 96_{10} for sector and word).

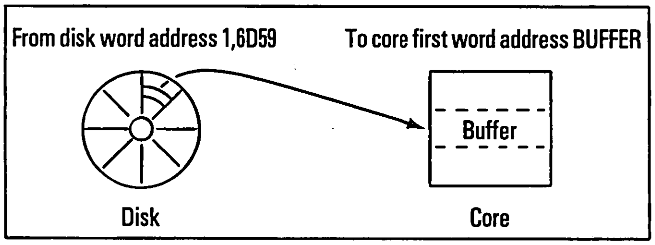

From disk word address 1,6D59
To core first word address BUFFER
Buffer
Disk
Core

Figure 4-7. Data Transfer Example

WRITE Request

The WRITE request transfers the requested number of words from core to disk. The disk starting address (msb, lsb) is interpreted by the driver as a word address. When writing on disk in this mode, the remainder of partially updated sectors is preserved.

A partial sector WRITE request causes:

- The entire sector to be read into a buffer in the driver

- The user's data to be moved into the appropriate portion of that buffer

- The entire buffer to be written onto the disk

Figure 4-8 is an example of a word-oriented WRITE request across several sectors.

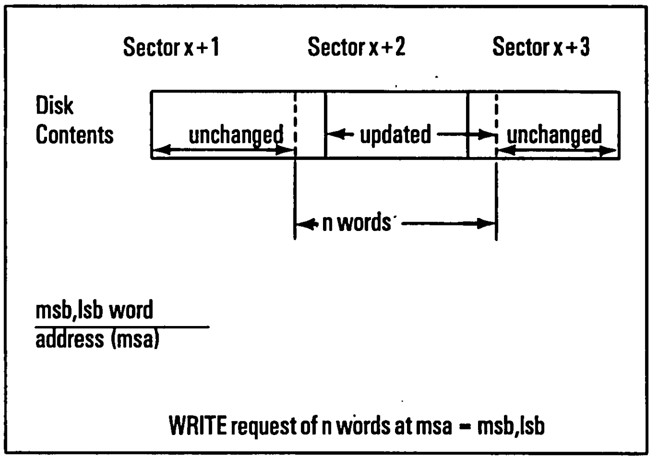


Figure 4-8. Example WRITE Request

FREAD/FWRITE REQUESTS

FREAD and FWRITE requests utilize the sector orientation of the disk. The formats of FREAD and FWRITE are the same as READ and WRITE. The msa represents a sector number; n represents the number of words to be transferred. If n is not a multiple of 96 for an FWRITE request, the unused words of the last sector are set to zero.

FREAD Request

FREAD fills core, starting at a requested address, with the specified number of words. If no words are requested, one word is transferred.

FWRITE Request

This request transfers the specified number of words from core to disk. The starting disk address is interpreted by the driver as a sector address; the msb must be zero. If no words are requested, one word is transferred. The remainder of a partially updated sector is not preserved. Using the same symbolic conventions as in the previous example, a normal FWRITE request appears as:

```
FWRITE      8,COMP,BUFFER,113,B,4,4,1
   .
   .
   .
ADC         0,103
   .
   .
   .
BSS         BUFFER(113)
```

In this case, 113 words are written from the core first word address BUFFER onto the disk, starting at sector 103.

The 856-4 Cartridge Disk has more than $7FFF sectors and requires both words of the request to specify the mass memory sector address. The sector address is defined in the same manner as a word address (the 16 msb in word 1 and 15 lsb in word 2 with bit 15 of word 2 set to 0).

word 1		16 msb
word 2	0	15 lsb

MOTION REQUEST

Motion requests to the disk result in no action, and return is through a normal completion.

Error Conditions

The following errors are detected by the drivers:

- Internal and external rejects
- Parity error
- Seek error
- Address error
- Lost-data error
- Protect fault
- Checkword error
- Defective-sector error
- Compare error
- Time-out error

On the following error conditions, error recovery is not attempted:

- Parity error
- Protect fault
- Time-out error

ERROR DETECTION FOR DISK TRANSFERS

Several methods for error detection are used during disk transfers. After data reads and writes, a hardware compare function can be issued to compare the data read or written with the data contained on the file. On detection of an error, a reposition and retry is attempted up to 10 times.

When an irrecoverable error occurs, the driver sets the error field of the disk physical equipment table and the error parameter in the request. The Q register is negative on entry to the completion program and indicates an irrecoverable error to the user. No information about the nature of the error is passed to the user.

LINE PRINTER DRIVER

This section defines the operation of the following device driver:

D42312—1742-30/1742-120 Line Printer Driver

WRITE/FWRITE REQUESTS

WRITE and FWRITE requests are honored by the line printer driver. Binary or ASCII mode has no significance and is ignored.

The driver prints up to 136 characters per line. The requester output buffer may be any length provided it contains embedded control characters. If more than 136 characters are supplied for one line, the additional characters are ignored. Printer control characters are†:

Character	Action Before Printing
0	Space two lines
1	Page eject
+	No space
All others	Space one line

The FWRITE mode advances one line before printing, and the first character of the record is printed. The unformatted WRITE does not cause a preceding advance. It prints the buffer only when a control character which causes a print or paper motion is encountered. The 1742-120 Printer requires a train image table to be appended to the driver (T5954).

MOTION REQUEST

A MOTION request to write end of file is honored as a page-eject function. All other MOTION requests cause no action with normal completion of the request.

Character Editing

All characters are edited as follows before they are sent to the print buffer:

Character	Action
$20-$5F	Send to buffer
$60-$7E (lowercase)	Change to $20-$5E
$03-EOT	Print buffer; advance one line; terminate request
$04-EOT	Same as $03
:T $09-HTAB	Simulated TAB; send blanks to buffer
:L $0A-Line feed	Ignore
:V $0B-VTAB	Print; select format tape level 2 and continue
:F $0C-Form feed	Select format tape level 1; top of form
:R $0D-Carriage return	Print buffer; advance one line
$1B-Escape	Used for direct function control of line printer

The next character is interpreted as follows:

$00-$2F	Ignored
$30	Print buffer; no advance; next line starts at the beginning
$31	Print buffer; single space; next line starts at the beginning
$32	Print buffer; double space; next line starts at the beginning
$33-$3E	Print buffer; select format tape level (01 to 12); continue printing from next printing position
$3F	Select eight lines per inch
$40	Clear controller; continue
$41-$7F	Ignored

Tab stops for tab simulation are assumed to exist every n characters of the print line. Each time a tab character is encountered, sufficient space characters are sent to the print buffer to advance the character counter to the next tab-stop position. n is a maximum of 20 in the released version.

†Applies to FORTRAN line printer only.

Error Conditions

The following errors are detected by the driver:

- Internal or external reject
- Hang-up
- Alarm

When the driver detects an irrecoverable failure, it sets the error field in bits 15 through 13 of word 9 of the physical device table for the device, and sets the error word in the Q register. Refer to appendix E for I/O error codes and descriptions.

MAGNETIC TAPE DRIVER

This discussion defines the operation of:

D17322—1732-2/615-73/615-93/10300 Buffered Magnetic Tape Driver

The 1732-2 Magnetic Tape Driver provides communication with the 615-73 and 615-93 Tape Transports. The driver can operate up to four transports. The controller has a buffered data channel capability in its standard configuration.

Magnetic Tape Driver Requests

The following paragraphs discuss magnetic tape driver requests.

READ/WRITE REQUESTS

READ and WRITE requests on the 615-73/615-93 Tape Transports differ from similar requests on other tape drives. Record lengths are defined by the requester. The number of words specified in a READ/WRITE request defines a logical record.

The type of magnetic tape drive determines the physical records. For 615-93 WRITE requests, a logical record is written as a physical record. For READ requests, the user-defined logical record length may be unrelated to the length of the physical record. The driver reads through record gaps until the logical record is complete or until it encounters a file mark.

For 615-73 requests, binary information must be repacked in allocatable core to use the assembly/disassembly hardware feature. The maximum length of a physical record is set to 192 words to limit core use. If a logical record is greater than 192 words, the record is segmented and written as a series of physical records. An analogous procedure is used for reading a logical record. Any unused portion of the last physical record is lost on subsequent READ operations unless the user retrieves that particular segment.

FREAD/FWRITE REQUESTS

FREAD and FWRITE requests are oriented toward physical records. A maximum length is a logical record. Any record greater than the specified logical record is truncated.

The necessary ASCII and BCD conversions take place for ASCII transfers on 615-73 drives. All information on 615-93 drives is transferred as binary in odd parity. Tapes generated on nine-track drives are not compatible with those generated on seven-track drives.

MOTION REQUEST

Refer to the section 3 discussion for the tape MOTION request.

Error Conditions

The following error conditions are recognized by the driver:

- No write ring on a write or write end-of-file mark
- Tape unit not ready
- Unit number not dialed
- Parity error
- Failure to interrupt
- Buffer channel not operative
- Lost data switch mode
- Missing processing module

These error conditions are considered irrecoverable. The user may continue, repeat the request, or down the driver.

PSEUDO TAPE DRIVER

The pseudo tape driver drives pseudo devices which, to the user, have the external characteristics of a magnetic tape. The pseudo devices are sequential files which may be accessed as normal magnetic tape by using the following monitor requests:

- READ
- WRITE
- FREAD
- FWRITE
- MOTION

The pseudo devices are also accessible with the use of the following job control statements:

- *REW
- *UNL
- *EOF

Pseudo Tape Driver Requests

The following paragraphs briefly define the pseudo tape driver requests.

READ/FREAD/WRITE/FWRITE REQUESTS

READ/FREAD and WRITE/FWRITE requests have a maximum record length of 192 words. Any attempt to write a longer record results in an error. Any attempt to read a longer record results in a short-read condition. Format and mode have no meaning. All information is transferred in binary mode.

MOTION REQUEST

The request format is described in section 3. Density has no meaning. All other motion codes are processed. When the driver is advancing or backspacing records and a file mark is encountered, the current MOTION command is terminated with the end-of-file status (bit 11, word 12) set in the physical device table.

SECTION 5

JOB PROCESSING

Job Processing

INTRODUCTION

The 1700 COS batch-processing subsystem initiates, monitors, and terminates all jobs that are executed in unprotected core. This batch-processing subsystem is scheduled for execution by the operator who must manually interrupt COS and type:

    *BATCH

On recognition of the *BATCH statement, the batch-processing subsystem is scheduled to begin processing user jobs. Processing continues until an *Z control statement is encountered, at which time, all job processing is terminated, and the batch-processing subsystem is released from core.

Jobs which are recognizable by the batch-processing subsystem consist of all processing features, executable through the use of the available job control statements. Each job to be initiated for execution must have a *JOB as its first control statement and must be terminated by a device-detectable, end-of-file statement[†]. In a job, all legal batch-processing control statements (except *V, *R, and *Z) are permissible only in the bounds defined by a *JOB and the end-of-file statements. This job structure permits a continuous flow of jobs through the subsystem without individual job initiation by operator intervention.

In the event of an abnormal job termination, all open files are closed, and the subsystem proceeds to the next job. This procedure is executed for all job processing prior to initiation of the next job.

If the control statement input device is the standard comment device, the character J is output, indicating the subsystem is waiting for a new control statement.

JOB CONTROL STATEMENTS

Control statements to the subsystem are format records in ASCII mode. A maximum of 72 characters is allowed for each control statement. The first character of an input statement must be an asterisk; the last must be a blank or a carriage return if input is on the teletypewriter. Intervening characters identify the type of statement and the action.

Legal Control Statements

The set of legal control statements for the subsystem can be categorized as follows:

- Control statements acceptable in a job

  *JOB
  *V[‡]
  *U
  IL
  *X
  *LGO
  *REW
  *UNL
  *CTO
  *PAUS
  *EOF

- Control statements acceptable to both a job and the manual interrupt routine

  *Z
  *R
  *K } May be entered at any time after manual interrupt

  *CSY May be entered only after a *JOB control statement has been entered

- Control statements for loader response during a job

  *

  *E

  *T

Control Statements in a Job

Control statements which are in a job are defined in the following paragraphs.

*JOB STATEMENT

An *JOB statement instructs the subsystem to begin accepting a new sequence of control statements. It must be the first control statement in a job, and only one is allowed for each job. The date and information on the JOB card is printed on the list device.

[†] For TTY input, a pseudo end of file is recognized by the job processor (*G statement).

[‡] *V,lu is also allowed outside the bounds of a job to permit the user to initiate input from a device other than standard input.

Format

The control statement format is:

```
*JOB
    or
*JOB,n,u,i
```

Parameters

The following parameters are defined for the *JOB control statement:

n—job name; first six characters are used by the job processor

u—user identification; first six characters are saved by the job processor (required if n is used)

i—comments (optional)

*V STATEMENT

An *V statement directs the subsystem to read all subsequent control statements from the specified logical unit.

Format

The control statement format is:

```
*V,lu,m
```

Parameters

The *V statement parameters are as follows:

lu—logical unit number; if not specified, standard input is assumed

m—mode in which control statements are read:

A or blank—formatted ASCII

B—formatted binary

*U STATEMENT

An *U statement directs the subsystem to read all subsequent control statements from the comment device. A printout at the comment device indicates that the job processor is ready to receive statements. An *U statement may occur in any order with respect to other statements in a job.

Format

The control statement format is:

```
*U
```

*L STATEMENT

An *L statement instructs the subsystem to call on the loader to load relocatable binary information. Once it is initiated, the loader continues loading from each specified logical unit until it reads an EOL block or a system control statement. The EOL block is an *T in the first two character positions. The subsystem can load from multiple logical units.

Format

The control statement format is:

*L,lu_1,lu_2,lu_n

Parameter

The parameter for the *L control statement format is:

lu_n—logical unit number for the loading device

If the unit is not specified for loader input, the standard input device is used. The loader keeps track of the upper and lower limits of available core and adjusts limits according to the amount of core allocated during input.

*X STATEMENT

An *X statement instructs the subsystem to begin program execution.

Format

The control statement format is:

```
*X,N
```

Parameters

The *X control statement parameters are:

N = Blank —loader is directed to produce a memory map after loading

= Specified—no memory map is produced

Statement Execution

When this statement is executed, the loader detects any unpatched externals and searches the program directory for a matching name. For each one found, the library routine is loaded into unprotected core as part of the job. If an unpatched external does not match any name in the program directory, the loader comments with an E on the standard comment device. When all unpatched externals are printed on the standard print device, the loader interrogates the comment device for an I, an *E, or an *T statement. If one of these occurs, they are interpreted as follows:

- * —causes execution regardless of unpatched externals
- *E—directory of part 0 core-resident entry points is searched for missing names; if externals still undefined, the loader interrogates the comment device for an *, an *E, or an *T statement
- *T—causes job termination

*LGO STATEMENT

*LGO is the load-and-go command. The subsystem calls the loader to load relocatable binary programs. The loader loads from each specified logical unit until it reads an EOL block or a system control statement. Loading may occur from multiple logical units.

Format

The control statement format is:

*LGO,N,lu_1,lu_2,...,lu_{10}

Parameters

The parameters are defined as follows:

N—no memory map is produced

lu—logical unit number for the loading device; if no parameters are specified (*LGO,N), standard scratch device is used

Second *LGO Format

The format for an additional possible *LGO statement is:

*LGO,lu_1,lu_2,...,lu_{10}

Parameter

In this second *LGO control statement format, the parameter is:

lu_1,lu_2,...,lu_{10}—logical unit number for the loading device; memory map is produced; if no parameters are specified (*LGO), standard scratch device is used

*LGO Statement Function

The loader keeps track of the upper and lower limits of available core and adjusts the limits according to the amount of core allocated during input. When the go portion of the *LGO statement is executed, the loader detects any unpatched externals and searches the program directory for a matching name. For each one found, the library routine is loaded into unprotected core as part of the job.

If an unpatched external does not match any name in the directory, the loader comments with an E on the standard comment device. When all unpatched externals are printed on the standard print device, the loader interrogates the comment device for an *, an *E, or an *T statement.

After all externals are patched, the user's program is executed.

Load-and-go operations use the *L, *X, or *LGO statements as per the following method. The load-and-go option provides for execution immediately following compilation or assembly. When load-and-go output is specified to the assembler, binary output is placed on the scratch unit, starting at sector 1 of the scratch area. The assembler produces an EOL statement for the end of binary output and stores the end of the load-and-go block count in $E4. The binary output of the next assembly begins at the sector which contains the EOL and continues until the assembly is completed. An *LGO statement is entered to load binary information from the scratch unit.

Example

The library is on logical unit 8 and programs are to be assembled, loaded, and executed. The following statements are required:

*ASSEM—load and execute the assembler; one of the parameters to the assembler requests load-and-go

*LGO —loads load-and-go information and executes the program which was just loaded

*REW STATEMENT

The *REW statement instructs the subsystem to rewind the specified logical units to their load point. Several logical units can be specified in one request. The maximum number of logical units which can be specified in an *REW statement is 5. An error message occurs if the number exceeds 5.

Format

The *REW statement format appears as follows:

*REW,lu_1,lu_2,lu_3,lu_4,lu_5

Parameter

The parameter is defined as:

lu_1,lu_2,...—logical unit number for the loading device; memory map is produced; if no parameters are specified (*LGO), standard scratch device is used

*UNL STATEMENT

An *UNL statement instructs the subsystem to rewind and unload the specified logical unit. The maximum number of logical units specified in an *UNL statement is 5.

Format

The format for the *UNL statement appears as:

*UNL,lu_1,lu_2,lu_3,lu_4,lu_5

*EOF STATEMENT

An *EOF statement instructs the subsystem to write one end-of-file mark to the current standard binary output device.

* CONTROL STATEMENT

An * control statement resets the load-and-go pointer to 1 and clears the loader-in-core flag. When the load-and-go mass storage area is to be used by more than one program in a job, this statement should be used to ensure proper execution.

An * statement should be entered to clear the loader-in-core flag whenever load operations have been performed but not executed and the user wishes to initiate a new load sequence from the start of unprotected core.

*CTO STATEMENT

An *CTO statement causes the comments that appear on the card to be printed on the standard comment device for operator information. Continuation cards are not allowed.

Format

*CTO format is:

*CTO,comments

*PAUS STATEMENT

This is the pause indicator.

Format

The format is:

*PAUS

On encountering an *PAUS, the following message is typed on the standard comment device:

READY?

The next control statement is read only after a carriage return to reply to this message. This statement can be used in conjunction with the *CTO statement to control operations. A time-out causes the message to by typed and a new request for input to be made.

*ENTRY POINT NAME STATEMENT

An *entry point name statement instructs the subsystem to call on the loader to load a program from the program library. The entry point name must appear in the program library directory.

The program, which is stored in the program library in relocatable binary format, is loaded into available core. The loader records the limits of available core before and after the load.

This operation is a program load. A program, which is loaded into core by a program load, is entered immediately for execution. An *X statement is not needed.

Format

The format is:

*entry point name

END-OF-FILE

This is either a user-supplied or a device-detectable code which terminates a job. It must be the last control statement in a job. A 6/7/8/9 sequence in column 1 is used with the 1729-3 Card Reader. For the teletypewriter input, an *G control statement serves the function of an end-of-file.

Statements Acceptable to Job and Manual Interrupt Routine

These statements are defined in the following paragraphs.

*K STATEMENT

An *K statement is used to reassign standard system logical unit numbers. By using an *K statement, the operator can select devices for system units other than those currently used.

One location in the communications region contains a physical device table ordinal for each of the standard system devices. The logical unit number in an *K statement replaces the number in the communication region. If a unit number designates a protected device, an error exit is taken.

Format

The control statement format is:

*K,Ilu,Llu,Plu

Parameters

The parameters in an *K statement are not ordered, but must be separated by a comma and followed by a carriage return or a space. The parameters are:

lu—logical unit number in all cases

I—system input unit

L—systm print unit

P—system binary output unit

*CSY STATEMENT

An *CSY statement reassigns standard COSY logical unit numbers. This subsystem control statement is used with the COSY driver. The I, P, and L parameters may also be used to assign logical units for the COSY program control statements. If no logical units are specified on the COSY control cards, the assigned units are used.

Format

The control statement format is:

*CSY,Ixx,Lyy,Pzz

Parameters

The parameters in an *CSY statement are not ordered, but must be separated by commas, and the last parameter must be followed by a carriage return or by a space. The parameters for the *CSY statement are:

xx—logical unit of COSY input library

yy—logical unit of COSY list output

zz—logical unit of Hollerith or COSY output

*CSY Command Sequence

The following sequence of commands is required to convert COSY source to Hollerith source:

- *CSY,Ixx,Lyy,Pzz

  Where:

  xx—logical unit of COSY input
  yy—logical unit of COSY list output
  zz—logical unit of Hollerith output

- *K,Iaa,Lbb,Pcc

  Where:

  aa—logical unit of COSY control statements

* STATEMENT

An * statement restores job execution at the place of interruption. When execution of a program in unprotected core is interrupted by a manual interrupt; typing an * causes the job execution to continue.

*Z STATEMENT

An *Z statement, which marks the end of batch processing, is accepted by the subsystem regardless of the order of its appearance.

Format

The format is:

```
*Z
```

Functions

After reading an *Z statement, the subsystem performs the following functions:

- Releases core space occupied by the subsystem
- Sets protect bits for all locations previously in unprotected core
- Releases this core area to the core allocator, which makes it available to protected system programs
- Resets the load-and-go pointer in location $E4 to 1

Functions After *Z Job Termination

During the execution of a program in unprotected core, the operator may terminate a job with a manual interrupt followed by typing an *Z. When this is done, the following functions are performed:

- Deletes interrupt stack entries that refer to unprotected core
- Sets the completion of all input/output into unprotected core to the address of the dispatcher
- Waits for the completion of all input/output from unprotected core if unbuffered protect processor operations are in execution
- Waits for the completion of all timer requests from unprotected core
- Job is terminated
- New job is initiated

*R STATEMENT

An *R statement informs the operating system that a device that previously failed is operable and is ready for input/output.

Format

The control statement format is:

```
*R,lu
```

Parameter

The following parameter is defined for the *R statement:

lu—logical number of failed device

Procedure To Assign Alternate Device

If a device fails, and an alternate device has been assigned, the alternate device is used to process results.

Example

When a device which is associated with logical unit 2 fails (and logical unit 2 has an alternate), the operator is notified by a comment, and input/output processing continues on the alternate device.

When the operator has taken corrective action regarding the device that failed, he notifies the operating system as follows:

Press:

MANUAL INTERRUPT

Type:

```
*R,2
```

This procedure restores logical unit 2 to the primary device.

Loader Response During Job Execution

The following list of control statements is used for loader response during job execution:

- * statement
- *E statement
- *T statement

These three statements are discussed briefly in this section under an *X statement and also under an *LGO statement.

SECTION 6

DEBUGGING AIDS

Debugging Aids

ON-LINE DEBUG PACKAGE

The on-line debug package (ODEBUG) allows the programmer to access both protected and unprotected core in order to change core and mass storage locations and to execute debugging functions while the system is running in an on-line state. The program is resident on mass storage; however, it is executed in allocatable core. When ODP is initiated, allocatable core is divided into three parts:

- Area permanently assigned to the executive program
- Area containing function processors; may extend into the third area, when necessary
- Area to which subroutines are transferred as needed

These areas are released when ODEBUG terminates.

OPERATOR PROCEDURES

Debugging is completed via the following initialization and termination procedures.

INITIATE ODEBUG

ODEBUG is initiated by pressing MANUAL INTERRUPT on the teletypewriter and keying in the characters DB. ODEBUG alerts the operator that it is in core and is ready for operator use by typing the following message:

DEBUG IN

The operator may then type in a request, which must be terminated by a carriage return. All requests are limited to one line on the input device. After the request is completed and all associated messages have been typed, ODEBUG types:

NEXT

The teletypewriter then waits for the next request from the operator.

TERMINATE ODEBUG

To terminate ODEBUG, the operator types:

OFF

ODEBUG replies by typing:

DEBUG OUT

The MANUAL INTERRUPT terminates any I/O action which is initiated by periodic requests to the monitor (this excludes magnetic tape motion requests). To terminate input/output, the operator must complete the following actions:

Press:

MANUAL INTERRUPT

Type:

DX

The I/O action stops, and DEBUG OUT is printed on the teletypewriter.

Diagnostic messages that are produced by ODEBUG are prefixed with DB (refer to appendix E).

Debug Mainframe Requests

The following requests are used to debug the mainframe.

STORE DATA IN CORE REQUEST

Data is stored in core by using this command.

Format

The statement format is:

LHX,cl,b/d,d,...,d

Parameters

The parameters used to store data in core are:

cl—core location

b—base (0 if not specified)

d—data (four-digit hexadecimal number)

LHX Examples

The following examples indicate how to utilize the LHX request.

To store data into core location plus base, use:

```
LHX,1600,0/14EA
```

This stores 14EA into location 1600.

A one-word relative instruction would be as follows:

```
LHX,cl,b/2-digit OP code*address
```

This loads core with the two-digit OP code which precedes the asterisk and the 8-bit relative increment (which is obtained by subtracting the address of the core location in which the data is to be stored from the address which follows this asterisk). The relative increment must be less than ±127; otherwise, an incorrect relative increment will be stored.

NOTE

> The address which follows the asterisk is always a 16-bit absolute address.

To store C805 into location 1601; therefore, use:

```
LHX,1601,0/C8*1606
```

For a 16-bit relative address, use:

```
LHX,cl,b/*address
```

This loads core with the 16-bit relative increment. The increment is obtained by subtracting the address of the core location in which the data is to be stored from the address which follows this asterisk.

NOTE

> The address which follows this asterisk is always a 16-bit absolute address.

To store FEFE into location 1602; therefore, use:

```
LHX,1602,0/*1501
```

In the example, the address is $0101 locations relative to the current location.

NOTE

> LHX,1600,0/14EA,C8*1606, *1501 has the same result as the three preceding examples.

LOAD DECIMAL INTO CORE REQUEST

A decimal is loaded into core via this request.

Format

The statement format is:

LDC,starting address/d_1,d_2,...,d_n

DUMP CORE REQUEST

To dump a five-digit decimal value of core from start plus base to end plus base, the following format is used:

```
DDC,sc,ec,b
```

Parameters

The parameters are:

sc—start core

ec—end core

b—base (0 if not specified)

To dump a four-digit hexadecimal value of core from start plus base to end plus base, the following format is used:

```
DPC,sc,ec,b
```

Parameters

The parameters in this example are:

sc—start core

ec—end core

b—base (0 if not specified)

WRITE CORE TO DISK REQUEST

To write core to disk, use this command.

Format

The statement format is:

```
WCD,sector,word in sector,core location, number of
  words
```

Parameters

The number-of-words parameter is decimal, and all other parameters are hexadecimal.

READ DISK TO CORE REQUEST

This statement is used to read disk to core.

Format

The statement format is:

```
RDC,sector,word in sector,core location, number of
  words
```

Parameters

The number-of-words parameter is decimal, and all other parameters are hexadecimal.

SEARCH CORE LOCATIONS REQUEST

By using this command, core locations are searched from start core to end core by the increment for a match between AND (mask, number) and AND (mask, core).

Format

The statement format is:

```
SCN,sc,ec,no,m,i
```

Parameters

The following parameters are applicable for the search core request format:

sc—start core

ec—end core

no—number

m—mask; "1" bits in mask indicate the position in core is examined; "0" bits in mask indicate the bit position in core is not examined

i—increment

Example of Search Core Location

The following search could be used:

```
SCN,0,7FFF,A1F7,FF00,2
```

In this example, the command calls for a search 0,2,4,6,8,... for AND (A1F7,FF00) or AND (A1xx,FF00) where xx may have any value. The locations and contents of the locations which contain the searched configuration are printed after the following heading:

```
CELL CONTENTS
```

REASSIGN LIST DEVICE USED BY DEBUG REQUEST

This command allows the user to reassign any list device which is required for output.

Format

The statement format is:

```
CLU,lu,
```

Parameter

The following parameter is used in the format for the CLU request:

lu—logical unit number of list device to be used by DEBUG

SEARCH CORE FOR PARITY ERROR REQUEST

Core is searched for parity errors via this request.

Format

The format for this command is:

```
SPE,last location in core
```

With this command, location of the parity error is printed. The following comment terminates the request:

```
SEARCH FINISHED
```

ADD HEXADECIMAL NUMBERS REQUEST

Hexadecimal numbers can be added through the use of this request.

Format

The format for adding up to eight hexadecimal numbers is:

```
ADH,number1,number2,...,number8
```

SUBTRACT HEXADECIMAL NUMBERS REQUEST

To subtract hexadecimal numbers, use this request.

Format

The format for subtracting number 2 from number 1 (hexadecimal) is:

```
SBH,number1,number2
```

SCHEDULE COMPLETION LOCATION REQUEST

Schedule the specified core location at the specified priority level by passing the contents in the Q register. Completion location may also designate directory calls.

Format

The statement format is:

SCH,cl,Q,cpl

Parameters

The parameters for this request are:

cl—core location (four-digit hexadecimal number)

Q—contents of Q register

cpl—completion priority level

SET CORE REQUEST

This command enables the operator to set core from the start to the end with pattern.

Format

The statement format is:

SET,sc,ec,pattern

Parameters

The parameters for the set core request format are defined as follows:

sc—start core

ec—end core

pattern—word to store in core

SET PROGRAM PROTECT BIT REQUEST

The program protect bit is set via this request.

Format

The statement format is:

SPP,sc,ec

Parameters

This request uses the following parameters as defined:

sc—start core

ec—end core

CLEAR PROGRAM PROTECT BIT REQUEST

The program protect bit is cleared via this request.

Format

The statement format is:

CPP,sc,ec

Parameters

The parameters for this statement request format are:

sc—start core

ec—end core

MOVE BLOCK IN CORE REQUEST

To move a block in core, this request is used.

Format

The statement format is:

MBC,sc,ec,nl

Parameters

These parameters are as follows:

sc—start core

ec—end core

nl—new location

Debug Core Allocation Requests

The debug core allocation requests are GEN and REL.

GENERATE SCRATCH AREA

Scratch area is generated by the GEN request.

Format

The format to generate scratch area in allocatable core is:

GEN,length,rp

Parameters

length—given in hexadecimal

rp—request priority (the minimum priority is three) of the location of allocated core is printed out as follows:

CORE ALLOCATED FROM $h_1h_1h_1h_1$ to $h_2h_2h_2h_2$

Where:

$h_1h_1h_1h_1$—start of allocated core, excluding the first two words used by the allocator

$h_2h_2h_2h_2$—end of allocated core

RELEASE ALLOCATED CORE

The allocated core is released by the REL request.

Format

The format to release allocated core is:

REL,scr

Parameter

scr—start of core to be released ($h_1h_1h_1h_1$ of a GEN request)

Magnetic Tape Requests

The following requests manipulate the magnetic tape on a specific unit; lu specifies the logical unit number. The parameters of these requests are decimal.

NOTE

Limitation on the number of records or files is 4,095 for all magnetic tape requests. Other standard device drivers accept the ODEBUG magnetic tape requests for MOTION; however, only one motion function is performed (that is single file or record skips, etc.).

ADVANCE FILES REQUEST

The advance files request advances the tape a number of specified files.

Format

The statement format is:

ADF,lu,number of files

BACKSPACE FILES REQUEST

To backspace the tape a number of specified files, use the backspace files request.

Format

The statement format is:

BSF,lu,number of files

ADVANCE RECORDS REQUEST

To advance the tape a number of specified records, the advance records request is used.

Format

The statement format is:

ADR,lu,number of records

BACKSPACE RECORDS REQUEST

The backspace records request is used to backspace the tape a number of specified records.

Format

The statement format is:

BSR,lu,number of records

Parameters

The backspace record request uses the following parameters:

lu—logical unit

number of records—one of blank

WRITE END-OF-FILE REQUEST

This request is used to write end-of-file.

Format

The statement format is:

WEF,lu,number of records

Parameters

The following parameters apply to the write end-of-file request format:

lu—logical unit

number of records—one if blank

REWIND TAPE REQUEST

This request is used to rewind the tape.

Format

The statement format is:

REW,lu

Debug Mass Storage Device Requests

The mass storage device can be debugged via the following commands.

CHANGE CORE-RESIDENT IMAGE ON SYSTEM LIBRARY UNIT REQUEST

This statement changes the core-resident image portion of the system on the system library unit. A maximum of five words can be changed by one LHC request.

Format

The statement format is:

LHC,cl,b/d,d,...,d

Parameters

The parameters for this request are:

cl—start core location

b—base to be added to cl (0 if not specified)

d—data (four-digit hexadecimal number)

LHC Example

The following LHC example can be used to change the core-resident image on the system library unit:

LHC,00E9,1400/18FC

In the example, the sector which contains data for core location 14E9 is changed to 18FC, and the core location itself is likewise modified. Relative instructions may be included, as in the debug of the mainframe as previously discussed.

REQUEST TO MODIFY MASS STORAGE RESIDENT PORTION OF OPERATING SYSTEM

This statement allows modification of the mass storage resident portion of the operating system. Changes are verified the same as for the LHC statement.

Format

The statement format is:

LHO,ord,loc,b/d,d,...,

Parameters

For this modification request, the parameters are:

ord—ordinal number of ordinal to be changed

loc—P location of the listing

b—base added to loc to obtain core location

d—data (four-digit hexadecimal number; maximum of five words for each request)

Example

The following statement is used to load hexadecimal data onto mass storage:

LHM,sector,word in sector/d_1,d_2,...,d_n

Where:

$n \leq 5$

Relative instructions may be included, as in the discussion of debugging the mainframe.

Since the LHC, LHO, and LHM functions accomplish writes to the system library, the maximum check for data integrity is achieved by printing out the image of the statement typed. If the statement is correct:

Press:

RETURN

The update then proceeds.

If the statement is incorrect:

Type:

* (Function is not performed)

Press:

RETURN

PANIC DUMP TO LINE PRINTER

In the case of a system crash, a dump of core memory should be taken. All the operations are performed on the computer front panel. The dump procedure is as follows:

1. Press STOP.
2. Press P and record the value that is displayed.
   Press A and record the value that is displayed.
   Press Q and record the value that is displayed.
3. Press MASTER CLEAR.
   Press P and enter 0140 in the register.
4. Press A and enter the first word address to be dumped (usually zero).
   Press Q and enter the last word address to be dumped (usually highest word in memory):
   - 7FFF for 32K system
   - 6FFF for 28K system
   - 5FFF for 24K system
5. Ensure that the printer is on-line.
6. Press GO.

The program results after execution of this procedure is as follows:

- Paper is set to the top of form
- Absolute and relative heading of 16 columns at the top of each page
- Absolute and relative addresses and 16 words are printed per printer line
- Lines, for which 16 words are the same as the last line printed, are ignored by printing a line of asterisks
- 60 lines are printed per page
- Program hangs when the requested number of words are printed

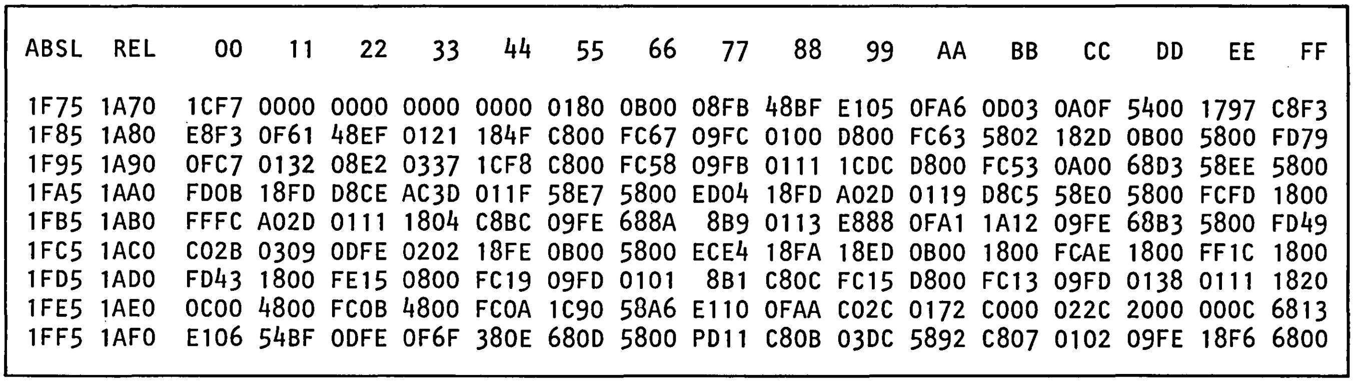
The panic dump program can be executed as many times as required to dump selected contents of core by repeating the operating procedure.

Printout

Figure 6-1 is a sample printout of this dump.

NOTE

In order to conserve core memory, the train image is not loaded, assuming an existing train image.

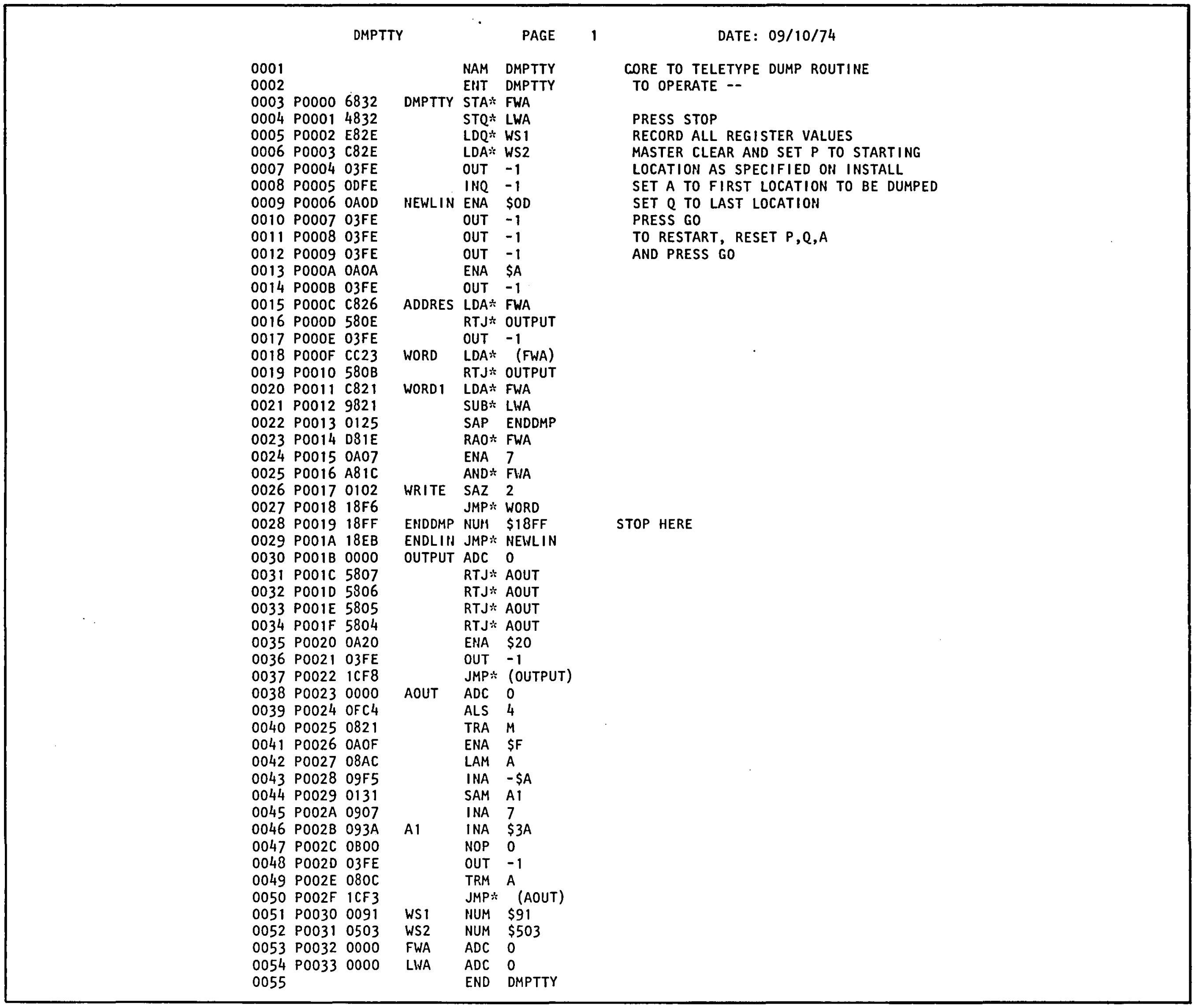


```
ABSL  REL    00   11   22   33   44   55   66   77   88   99   AA   BB   CC   DD   EE   FF

1F75 1A70  1CF7 0000 0000 0000 0000 0180 0B00 08FB 48BF E105 0FA6 0D03 0A0F 5400 1797 C8F3
1F85 1A80  E8F3 0F61 48EF 0121 184F C800 FC67 09FC 0100 D800 FC63 5802 182D 0B00 5800 FD79
1F95 1A90  0FC7 0132 08E2 0337 1CF8 C800 FC58 09FB 0111 1CDC D800 FC53 0A00 68D3 58EE 5800
1FA5 1AA0  FD0B 18FD D8CE AC3D 011F 58E7 5800 ED04 18FD A02D 0119 D8C5 58E0 5800 FCFD 1800
1FB5 1AB0  FFFC A02D 0111 1804 C8BC 09FE 688A  8B9 0113 E888 0FA1 1A12 09FE 68B3 5800 FD49
1FC5 1AC0  C02B 0309 0DFE 0202 18FE 0B00 5800 ECE4 18FA 18ED 0B00 1800 FCAE 1800 FF1C 1800
1FD5 1AD0  FD43 1800 FE15 0800 FC19 09FD 0101  8B1 C80C FC15 D800 FC13 09FD 0138 0111 1820
1FE5 1AE0  0C00 4800 FC0B 4800 FC0A 1C90 58A6 E110 0FAA C02C 0172 C000 022C 2000 000C 6813
1FF5 1AF0  E106 54BF 0DFE 0F6F 380E 680D 5800 PD11 C80B 03DC 5892 C807 0102 09FE 18F6 6800
```

Figure 6-1. Printout of Panic Dump

Panic Dump to Teletype

A panic memory dump routine is included in the COS package; however, the only output media is a line printer. The dump routine in figure 6-2 is given to show a memory dump to the teletypewriter.



```
            DMPTTY              PAGE    1            DATE: 09/10/74

                         NAM  DMPTTY        CORE TO TELETYPE DUMP ROUTINE
                         ENT  DMPTTY         TO OPERATE --
P0000 6832        DMPTTY STA* FWA
P0001 4832               STQ* LWA            PRESS STOP
P0002 E82E               LDQ* WS1            RECORD ALL REGISTER VALUES
P0003 C82E               LDA* WS2            MASTER CLEAR AND SET P TO STARTING
P0004 03FE               OUT  -1             LOCATION AS SPECIFIED ON INSTALL
P0005 0DFE               INQ  -1             SET A TO FIRST LOCATION TO BE DUMPED
P0006 0A0D        NEWLIN ENA  $0D            SET Q TO LAST LOCATION
P0007 03FE               OUT  -1             PRESS GO
P0008 03FE               OUT  -1             TO RESTART, RESET P,Q,A
P0009 03FE               OUT  -1             AND PRESS GO
P000A 0A0A               ENA  $A
P000B 03FE               OUT  -1
P000C C826        ADDRES LDA* FWA
P000D 580E               RTJ* OUTPUT
P000E 03FE               OUT  -1
P000F CC23        WORD   LDA*  (FWA)
P0010 580B               RTJ* OUTPUT
P0011 C821        WORD1  LDA* FWA
P0012 9821               SUB* LWA
P0013 0125               SAP  ENDDMP
P0014 D81E               RAO* FWA
P0015 0A07               ENA  7
P0016 A81C               AND* FWA
P0017 0102        WRITE  SAZ  2
P0018 18F6               JMP* WORD
P0019 18FF        ENDDMP NUM  $18FF         STOP HERE
P001A 18EB        ENDLIN JMP* NEWLIN
P001B 0000        OUTPUT ADC  0
P001C 5807               RTJ* AOUT
P001D 5806               RTJ* AOUT
P001E 5805               RTJ* AOUT
P001F 5804               RTJ* AOUT
P0020 0A20               ENA  $20
P0021 03FE               OUT  -1
P0022 1CF8               JMP* (OUTPUT)
P0023 0000        AOUT   ADC  0
P0024 0FC4               ALS  4
P0025 0821               TRA  M
P0026 0A0F               ENA  $F
P0027 08AC               LAM  A
P0028 09F5               INA  -$A
P0029 0131               SAM  A1
P002A 0907               INA  7
P002B 093A        A1     INA  $3A
P002C 0B00               NOP  0
P002D 03FE               OUT  -1
P002E 080C               TRM  A
P002F 1CF3               JMP*  (AOUT)
P0030 0091        WS1    NUM  $91
P0031 0503        WS2    NUM  $503
P0032 0000        FWA    ADC  0
P0033 0000        LWA    ADC  0
                         END  DMPTTY
```

Figure 6-2. Panic Dump to Teletypewriter

SECTION 7

LIBRARY EDITING

Library Editing

INTRODUCTION

The library editing program allows the user to:

- Add a program or file to the program library
- Remove a program or file from the program library
- Replace one program or file with another in the program library
- Combine several relocatable binary programs in absolute binary record and output this record on the binary output device
- Transfer information between peripheral devices

The program library is comprised of programs that are stored in either relocatable or absolute form. Each relocatable binary program in the library is referenced by one or more entries in the program library directory. These entries consist of all entry points which are declared in the programs which referenced it. No two programs in the program library may have duplicate entry point names, although a file may have the same name as an entry point in a program. Each file is referenced by a file name in the program library.

LIBEDT PROGRAM

The control statement *LIBEDT, instructs the job processor to load the library editing program into protected core and to begin operation. The library editing program types LIB on the comment device to indicate it has been entered. After completing its functions, the program exits to the job processor.

Output for the library editing program is on three devices.

- Comment device

  This device prints error messages and indicates the entrance to the library editing program.

- Standard binary output device

  This device produces absolute records on an external device from relocatable binary input.

- Standard print output device

  This device lists the system or program library directory.

CONTROL STATEMENTS

Control statements to the library editing program are format records. The first character of a control statement must be an asterisk; the last must be a carriage return. Intervening characters identify the type of statement and the action.

Types of Control Statements

The standard list of control statements for the library editing program includes the following:

- *M—replace program in system library
- *L—add/replace program in program library
- *P—produce absolute record
- *U—return to comment device for next control statement
- *V—get next control statement
- *Z—terminate processing
- *DL—list program library directory
- *DM—list system library directory
- *N—modify program library files
- *T—transfer information
- *K—change I/O devices
- *R—remove program
- *F—end-of-transfer indicator
- *FOK—transfer indicator

REPLACE PROGRAM IN SYSTEM LIBRARY

An *M (replace program in system library) statement replaces a program in the system library which executes in allocatable core with another program.

Format

The statement format is:

```
*M,or,s,d,M,N
```

Parameters

The parameters utilized for this statement are as follows:

or—This is the ordinal number in the system library directory and is a required parameter. If the ordinal number does not appear in the system library directory, or if the parameter is not specified, the statement is illegal.

s—This is a mass storage address. This parameter is illegal if M is blank.

After the relocatable binary programs are loaded and linked, a check is made on the thread of the directory entry which is being replaced. If the thread of the directory entry is busy, LIBEDT waits for it to be freed, indicating this directory entry is not currently being operated. The thread is then set to a busy state to prevent scheduling of the file while LIBEDT is manipulating the directory and writing the new file onto mass storage (beginning at address s). If s is not specified in the input statement, a search of mass storage is made to find the first block of sectors which is large enough to contain this file. In most instances, this block is at the end of the library. When the new program is shorter than the one it is supposed to replace, it is stored on the sectors of the file which it is replacing. At the completion of the update, the thread is cleared.

d—This is the data base indicator. This parameter is specified to allow linkage of allocatable core programs to data which has been previously linked to the program at initialization.

NOTE

When using this parameter, the following restriction exists: data cannot be preset into the established labeled common block by the loader, but program references will be absolutized properly.

M—M is the mass storage indicator. When M appears in the statement, the program which is to be replaced is mass-storage resident, and the ordinal represents its position in the system library directory relative to other mass-storage resident programs. When the length of the replacement program is less than or equal to the length of the program being replaced (rounded-up to the nearest sector), the new one overlays the old one on mass storage.

If the replacement program is longer than the program to be replaced, it is added to the library on the first block of available sectors which is large enough to contain the file. The total length of the file which is being loaded cannot exceed the length of allocatable core, including unprotected core.

If M is omitted in the statement, the program to be replaced is core-resident, and the ordinal represents its position in the system library directory relative to other core-resident programs. The length of the new program, if core-resident, must be equal to or less than that of the program which is being replaced. Since the directory entry for a core-resident ordinal does not contain length, no error indication can be given if it is longer than the program being replaced.

CAUTION

On-line use of LIBEDT for replacement of system library programs exposes the operation to the following potential faults: 1) The interrupt system is disabled during core-resident program replacement, which for large programs, could inhibit system response for excessive periods of time. The program length of a core-resident program which is being replaced cannot be checked by LIBEDT. 2) A larger program is loaded without indication of error, potentially destroying part of the system.

N—This indicates that linking to the program library is not required. When a new program is added to the system library, the library editing program issues a loader request to load one or more relocatable binary programs from the standard input device until a loader EOL statement (*T), a nonloader statement, or a device failure is detected. If at this time any unpatched externals exists, automatic linkage is performed to the core resident entry point (CREP) tables. Any unpatched externals that remain after this linkage are listed, and the user has the option of continuing by typing an *. Termination of the load is performed by manual interrupt followed by *Z. If the field is blank, automatic linkage to the program library is performed after the CREP linkage. Any remaining unpatched externals are listed, and the user may continue or terminate as previously described.

If loading is terminated with an EOL statement, the library editing program looks to the comment device (*U statement) or standard input device (*V statement) for the next control statement. If loading was terminated by a nonloader statement, the nonloader statement is processed as a control statement to the library editing program

ADD OR REPLACE PROGRAM IN PROGRAM LIBRARY

An *L (add/replace program in program library) statement adds a new program to the library or replaces a program in the library.

Format

The statement format is as follows:

*L,epn

Parameter

The parameter used in the *L statement is:

epn—If the entry-point name does not appear in the program library directory, this statement adds a new program to the library. When an addition is made to the library, the library editing program reads format records of binary input from the standard input device and writes them onto mass storage.

Entry-point names for programs added to the library are recorded in the directory along with the beginning mass storage addresses.

If the entry-point name does appear in the directory, the program which contains this entry point is replaced. The new entry-point name is placed in the directory. When a program is replaced in the library, its entry-point name is removed from the directory.

If a mass storage unit is to be used as a system input device, the input operation begins at the first scratch sector. This feature allows the user to assemble and obtain load-and-go output. By assigning the load-and-go unit as the system input device with a monitor control statement (*K,I unit number), the load-and-go unit becomes the input device for processing an *L, entry-point name control statement.

PRODUCE ABSOLUTE RECORD

An *P (produce absolute record) statement directs the library editing program to produce an absolute record from one or more relocatable binary programs. The relocatable binary programs are loaded in core by the loader under control of the library editing program.

Format

The statement format is as follows:

*P,n,P/R,sa

Parameters

The parameters for the request to produce an absolute record are as follows:

n—The parameter n indicates the record format

n ≠ F or omitted

A single format record is written on the standard binary output device

n = F

Output is in format records of 96 words each. If binary output is assigned to a mass storage device, this unit must be the library unit, since subsequent operations expect the absolute file to start on the scratch area following the library (that is, the sector defined by the contents of $C0 and $C1).

P/R—This parameter indicates the order of linkage. The use of the P/R option makes it possible build subprogram parts for foreground-allocatable or partition-core programs. Such programs can then be stored on the disk as files by using the *N LIBEDT processor and can be overlaid in foreground user buffer areas by using the user programs. The implementation of this technique negates the need of applying system directory entries for such files.

P parameter

If the P parameter field is blank, the order of linkage is the preset table, the program library, and/or the unprotected, unlabeled common area. If at this time, unpatched externals exist, the *P processor links to the CREP tables. If any unpatched externals still exist following this linkage, they are listed, and the user can enter an * to continue or an *T to terminate.

If the parameter field is set to P, linkage is performed in the following order: protected unlabeled common, CREP tables, and the program library. If unpatched externals exist following the linkage, a list is printed, and the user can enter an * or an *T.

R parameter

If the parameter field is set to R, linkage is performed in the following order: protected unlabeled common, CREP 1 table, CREP table, and the program library. If unpatched externals exist following the linkage, a list is printed, and the user can enter an * or an *T.

If the parameter field is set to a numeric value between 1 and 16, the relocatable binary programs are absolutized at the beginning of the partition which was specified by the field. In all other cases of the P/R parameter, absolutizing begins at ($F7) plus one.

sa—This indicates the starting address and can be one of the following:

$hhhh

This is the hexadecimal number of the core address.

entry point name

This indicates the core address for an entry point of the relocatable binary program read in by the loader. The entry point name DATBAS is used to reference the data block set aside during a loader operation.

entry point name + hhhh

hhhh is added to or subtracted from the core address to find the starting address.

If the starting address is omitted in an *P statement, the starting address is the beginning of the load at the address specified by ($F7) plus one. The binary output terminates with the last word of the load.

If the starting address is specified, the binary output extends from the starting address to the last word of the load. The starting address, therefore, must not be specified beyond the last word address of the relocatable binary load. If the *P statement is unacceptable to LIBEDT because it exceeds the last word address of the relocatable binary load, the error message E11 appears on the print device. The operator must type in an acceptable starting address without repeating an *P,n. A carriage return without a starting address has the same effect as a starting address equal to the contents of location $F7 plus one. The error message is issued by the loader which is processing the starting address portion of an *P statement.

RETURN TO COMMENT DEVICE FOR NEXT CONTROL STATEMENT

An *U (return to comment device for next control statement) statement directs the library editing program to go to the comment device for subsequent control statements.

Format

The format is as follows:

```
*U
```

GET NEXT CONTROL STATEMENT

An *V (get next control statement) statement directs LIBEDT to read control statements from the specified logical unit until an *U statement is read.

Format

The format is as follows:

```
*V,lu,m
```

Parameters

The parameters for the *V statement are as follows:

lu—logical unit; if the logical unit is not specified, LIBEDT (system) input unit is assumed

m—mode of control statement

A—formatted ASCII mode

B—formatted binary mode

If mode is not specified, the control statements are read in formatted ASCII mode.

If the control statement *LIBEDT is read under an *V option in the job processor, LIBEDT continues processing with the same *V option.

TERMINATE PROCESSING

An *Z (terminate processing) statement terminates the library editing program processing and returns control to the job processor.

Format

The format is as follows:

*Z

LIST SYSTEM LIBRARY DIRECTORY

An *DM (list system library directory) statement directs the library editing program to list the system library directory on the print output device. The statement includes an *Y or an *YM ordianl at the beginning of each line.

Format

The format is as follows:

*DM

LIST PROGRAM LIBRARY DIRECTORY

An *DL (list program library directory) statement directs the library editing program to list the program library directory on the standard print output device.

Format

The format is as follows:

*DL

MODIFY PROGRAM LIBRARY FILES

An *N (modify program library files) statement is used to add, replace, or edit a permanent binary file in the program library.

Format

When a file is added or replaced, only name and mode have to be specified in the following format:

*N,n,w_1,w_2,m

Parameters

Parameters for the *N statement are:

N—name of file; name is a 1-to-6-character identification by which the file is addressed

n—name of file; name is a 1-to-6-character identification by which the file is addressed

w_1—first word of file to be changed; w_1 and w_2, where $w_1 < w_2$, are used if only part of a file is to be changed

w_2—last word of file to be changed (refer to w_1); if w_2 is omitted, only the word specified by w_1 is changed

m—the mode of input:

A—ASCII format records

B—binary format records

Input to *N Processor

Input to an *N processor consists of format records of 96 words or less. Input is terminated with any valid LIBEDT or an *Z statement.

The load-and-go unit can be used as an input device for processing an *N statement in the same way as for an *L statement.

SET CORE REQUEST PRIORITY

An *S (set core request priority) statement sets the core request priority of an entry in the system directory. This determines the area in core where the file runs.

Format

The format is as follows:

*S,or,v,M

Parameters

or—ordinal number which refers to an entry in the system directory

N—level at which the request priority is to be set: 0 ≤ v ≤ 15

M—specifies mass-storage resident or core-resident

M—ordinal number is mass-storage resident

≠ M—ordinal number is core-resident

TRANSFER INFORMATION

An *T (transfer information) statement permits the transfer of information between any two peripheral devices such as a card-to-tape or a tape-to-printer device†.

An *T statement can be used in conjunction with an *F pseudo LIBEDT statement to perform information transfer in a batch job. On recognizing an *F during a transfer operation, LIBEDT outputs the IN message and proceeds to read the next control card.

Format

The format for an *T statement is as follows:

*T,i,mi,o,mo,n,f

Parameters

The statement format parameters are identified in the following manner:

i—input logical unit; if omitted, LIBEDT's standard input binary unit is selected

mi—mode of input

A—ASCII

B—binary

o—output logical unit; if omitted, LIBEDT's standard output unit is selected

mo—mode of output

A—ASCII

B—binary

n—sets an upper limit on the number of records to be transferred; if n is omitted, records are transferred until the input device is empty or fails or encounters an *F control statement. The upper limit on the number of records to be transferred is decimal. At the end of transfer, the number of records and files encountered is printed in decimal format on the standard print device.

f—sets an upper limit on the number of files to be transferred; if f is omitted, files are transferred until the input device is empty, fails, or encounters an *F control statement. The upper limit of the number of files to be transferred is decimal. At the end of transfer, the number of records and files encountered is printed in decimal format on the standard print device.

CHANGE INPUT/OUTPUT DEVICES

An *K (change input/output devices) statement, which may occur in any order in respect to other statements, allows the operator to change LIBEDT devices. These

†LIBEDT for Phase-Encoded Tapes
LIBEDT does not provide density selection. In order to use LIBEDT to copy tapes, density for each phase-encoded tape unit must be preset by using the following IOUP commands

- TSD,u,8 (select 800 bpi)
- TSD,u,16 (select 1,600 bpi)

With the appropriate densities selected, LIBEDT will copy:

- 800 to 800 bpi tapes
- 800 to 1,600 bpi tapes
- 1,600 to 1,600 bpi tapes
- 1,600 to 800 bpi tapes

CAUTION

The hardware returns to 1,600 bpi (and lights the HI DEN light) whenever a master clear or clear controller is issued. But with word 16 of each units' physical device table set to the required density, proper density returns to each unit as soon as a connect is made to it.

changes are internal to LIBEDT and do not affect the system device assignments.

Format

The format for an *K statement is as follows:

*K,Ilu,Plu,Llu

Parameters

The *K statement uses the following parameters:

lu—logical unit number; if a unit number designates a protected device, an error exit is taken

I—LIBEDT's input unit

P—LIBEDT's binary output unit

L—LIBEDT's print unit

Example of *K Statement

The parameters of an *K statement may be in any order, but must be separated by commas. In the following example, this statement sets LIBEDT's input unit to logical unit 2, its print unit to logical unit 5, and its binary output unit to logical unit 3.

*K,I2,L5,P3

An *K statement is terminated by a carriage return.

REMOVE PROGRAM

An *R (remove program) control statement removes a program with entry point n from the program library.

Format

The format is as follows:

*R,n,F

Parameters

The parameters are defined as follows:

n—entry point of program or name of file to be removed; if F is included, the file name n is removed

F—specifies that n is a file name

END-OF-TRANSFER INDICATOR

The *F (end-of-transfer indicator) statement is a pseudo instruction to the *T processor of LIBEDT. When an *F followed by two spaces is encountered, the current *T operation is terminated and the next LIBEDT control statement is read from the standard input device.

TRANSFER INDICATOR

The *FOK (transfer indicator) statement, like an *F statement, is a pseudo instruction to the *T processor of LIBEDT. On encountering this statement during input, an *F is transferred to the output device.

Table 7-1 shows sample transfer request statements.

Table 7-1. Sample Transfer Request Statements

Type	Statement	Description
Call	*JOB	Call job processor
	*LIBEDT	Call LIBEDT
Typical uses	*T,6,A,9,A,,2	Transfer two files of information from logical unit 6 to logical unit 9
	*L,PROG	Put relocatable program on library
	*N,PROG1,,,B	Put absolutized file on library
	*M,10,,,M	Put allocatable core program on system library
	*A12,2,1,,,,,	Put partition core program on system library

SECTION 8

SYSTEM MAINTENANCE AND UTILITY ROUTINES

System Maintenance and Utility Routines

INTRODUCTION

To increase the ease of installing, updating, and debugging programs, the COS has system and maintenance routines. These routines are discussed in this chapter.

CALLING STATEMENTS

The following list gives the calling name for the various maintenance routines. Each statement is described in this sequence in the remainder of this section.

- LCOSY—list COSY
- DTLP—disk-to-tape loading program
- IOUP—input/output utility package
- COSY—source program compression
- SKED—skeleton editor
- LIBILD—library builder
- PAULA—update install tape
- BOOT—tape bootstrap loader
- PIC—COSY update manipulating utility
- CYFT—COSY formatting program
- SUP—system utility processor
- ASSEM—macro assembler

LCOSY PROGRAM

The list COSY (LCOSY) program provides a means of listing the names of programs on a COSY tape and punching DCK/control cards for each program.

LCOSY Execution

To execute this program, the procedure is as follows:

```
*JOB

J

*K,Ilu,Plu,Llu

*LCOSY
```

PARAMETERS

The parameters are:

I—assigns a logical unit of COSY tape to be read

P—assigns logical unit where DCK/ control statements are to be punched; the END/ statement is not punched

L—assigns logical unit where names of programs are listed

LCOSY EXECUTION LISTING

The listing appears in the following format:

```
PBY      CSY/

PBYA     CSY/

PBZ      CSY/

PBZA     CSY/
```

On executing LCOSY, a typeout occurs such as the following:

```
DCK/ I,H,C
```

LCOSY waits for a two-digit logical unit number for each parameter separated by a comma; for example:

```
06,07,18
```

If a parameter is not desired, a slash replaces the logical unit number; for example:

```
06,/,18
```

If a slash is used under the I parameter, no DCK/ control cards are punched; only a listing of the deck names from the input source is produced.

DTLP PROGRAM

The disk-to-tape loading program (DTLP) has two purposes:

- Save the contents of the system disk by dumping the absolute image to magnetic tape
- Load a new disk with the absolute image previously saved

DTLP Execution

To execute this program, the procedure is as follows:

```
*JOB

J

*DTLP
```

When DTLP is called from the program library for execution, it prints the following message on the standard comment device:

DTLP FIRST WORD ADDRESS WILL BE XXXX

Where:

XXXX—core location

DTLP then prints the following message on the comment device:

TURN OFF PROTEC SWITCH,
TYPE CARRIAGE RETURN

The operator should turn the autoload console key to M (horizontal position), and press the carriage RETURN key. DTLP then enters into the following message dialog with the operator.

1. 4 DIG. EQ. CODE FOR..
   MAG TAPE

   Response to this first message is the four-digit hexadecimal equipment code, 0381.

2. 4 DIG. EQ. CODE FOR..
   MASS MEMORY

   Respond to this message by typing the four-digit hexadecimal equipment code for the mass memory controller, 0181.

3. ILLEGAL PARAMETERS SPECIFIED

   - If either equipment code given contains a nonhexadecimal digit, this message is printed.

   - Control reverts to message 1.

4. SCRATCH SECTOR IN $C1 IS --XXXX

   - When the equipment codes have been properly specified, this message is typed with XXXX being the current beginning of scratch mass memory.

   - If a system is being saved on tape, all mass memory sectors up to XXXX should be saved.

5. TYPE LOAD FOR TAPE-TO-DISK,
   SAVE FOR DISK-TO-TAPE
   OR CARRIAGE RETURN

   - This message is self-explanatory.

   - If LOAD is the response, control goes to message 6.

   - If SAVE is the response, control goes to message 7.

   - If a carriage return is the response, control goes to message 11.

6. INPUT TAPE ON UNIT 0. READY?

   Respond with a carriage return when ready.

7. OUTPUT TAPE ON UNIT 0.
   HOW MANY SECTORS?

   - Mount a blank tape (with write ring) on tape unit 0.

   - Respond with the four-digit hexadecimal number of sectors to be saved. If the number is not valid, message 7 is repeated.

8. DISK ERROR (XXXX)

   Where:

   XXXX is the last disk status

   If the mass memory device does not respond properly to I/O commands, this message is typed, and control passes to message 5.

9. TAPE ERROR (XXXX)

   Where:

   XXXX is the last tape status

   If the magnetic tape does not respond, this message is printed, and control passes to message 5.

10. XXXX SECTORS LOADED

    This message is printed when the LOAD operation has completed, and control passes to message 11.

11. TYPE V FOR VERIFY, A FOR AUTOLOAD,
    OR A CARRIAGE RETURN TO RESTART

    - When the SAVE operation is completed, this message is printed.

    - The tape rewinds and unloads.

- If a carriage return is the response, control passes to message 5.
- If the response is a V, the tape is verified against mass memory when message 12 is responded to with a carriage return.

12. VERIFY TAPE ON UNIT 0. READY?

    Reload the tape, and press carriage RETURN.

13. XXXX SECTORS VERIFIED

    - When the verify is complete, this message is output, and control goes to message 11.
    - If the response to message 11 is an A, the program simulates an autoload by reading the first track of the mass memory device to location 0 and jumping to 0. This is the only exit from DSKTAP.
    - When errors are encountered on verify, control goes to message 14.

14. SECTOR XXXX WORD -- WW --
    DOES NOT COMPARE
    TYPE C TO CONTINUE OR
    A CARRIAGE RETURN TO ABORT.

    Where:

    XXXX is the sector and WW is the word in the s sector.

    - Only the first verify error in a block of 16 sectors is logged. Verify errors cause this message.
    - When a carriage return is the response, control passes to message 11.

Tape-to-Disk Load Using System Save Tape

Loading the CYBERDATA operating system to disk from the image on the system save tape is done as follows:

1. Load the seven- or nine-track bootstrap.

   Enter the applicable seven- or nine-track bootstrap manually (see procedure in appendix A), or load *BOOT from the CYBERDATA program library.

2. Mount the system save tape on unit 0.

3. Press MASTER CLEAR and GO on the console.

   The first file (disk initializer) is read from tape.

4. Press MASTER CLEAR and GO.

   The disk initialization messages appear on the teletype.

   NOTE
   Be sure the correct disk is mounted.

5. Execute the disk initializer.

   This requires 1 to 3 minutes, depending on the options selected.

6. When disk initialization is complete, press MASter clear and GO.

   Read next file (DTLP) from tape.

7. Press MASTER CLEAR and GO.

   DTLP messages will appear on teletypewriter as follows:

   ```
   4 DIG. EQ. CODE
   FOR..MAG TAPE
   ```

8. Enter 0381 $\boxed{\text{CR}}$

   Teletype message:

   ```
   4 DIG. EQ. CODE
   FOR..MASS MEMORY
   ```

9. Enter 0181 $\boxed{\text{CR}}$

   Teletype message:

   ```
   SCRATCH SECTOR IN$C1 IS...xxxx.
   TYPE LOAD FOR TAPE-TO-DISK,
   SAVE FOR DISK-TO-TAPE
   OR A CARRIAGE RETURN
   ```

10. Enter LOAD $\boxed{\text{CR}}$

    Teletype message:

    ```
    INPUT TAPE ON UNIT 0. READY?
    ```

11. Enter $\boxed{\text{CR}}$

    This copies the tape to disk, rewinds and unloads the tape.

    Teletype message:

    ```
    xxxx SECTORS LOADED
    TYPE V FOR VERIFY, A FOR AUTOLOAD,
    OR A CARRIAGE RETURN TO RESTART
    ```

12. Enter A CR

    Teletype message:

    ```
    CYBERDATA LEVEL X.XX LOADED
    PP
    ```

13. Restore the autoload console key to N (vertical position)

    Teletype message:

    ```
    ENTER DATE/TIME
    MMDDYYHHMM
    ```

14. Enter Date, Time CR

    Teletype message:

    ```
    month,date,year,time
    ```

To verify the load, proceed as follows:

1. Press MANUAL INTERRUPT

   Teletype message:

   ```
   MI
   ```

2. Enter DB CR

   Teletype message:

   ```
   DEBUG IN
   ```

3. Mount the system save tape again.

4. Enter ADF,6,2 CR

   This advances over the first two files on tape.

   Teletype message:

   ```
   DEBUG OUT
   ```

5. Enter OFF CR

6. Press MANUAL INTERRUPT

7. Enter *BATCH CR

8. Enter *JOB CR

   Teletype message:

   ```
   J
   ```

9. Enter *DTLP CR

   Teletype message:

   ```
   DTLP FIRST WORD ADDRESS
   WILL BE XXXX TURN OFF PROTEC
   SWITCH, TYPE CARRIAGE RETURN
   ```

10. Turn autoload console key to M (horizontal position). This removes system protect.

11. Press CR

    Teletype message:

    ```
    4 DIG. EQ. CODE FOR...MAG TAPE
    ```

12. Enter 0381 CR

    Teletype message:

    ```
    4 DIG. EQ. CODE FOR..MASS MEMORY
    ```

13. Enter 0181 CR

    Teletype message:

    ```
    SCRATCH SECTOR IN$C1 IS..xxxx
    TYPE LOAD FOR TAPE-TO-DISK,
    SAVE FOR DISK-TO-TAPE
    OR A CARRIAGE RETURN
    ```

14. Enter CR

    Type V for verify, A for autoload or a carriage return to restart.

15. Enter V CR

    Teletype message:

    ```
    VERIFY TAPE ON UNIT 0. READY?
    ```

16. Press CR

    After successful verify, teletype message:

    ```
    xxxxSECTOR VERIFIED
    TYPE V FOR VERIFY, A FOR AUTOLOAD,
    OR A CARRIAGE RETURN TO RESTART.
    ```

17. Label the new disk.

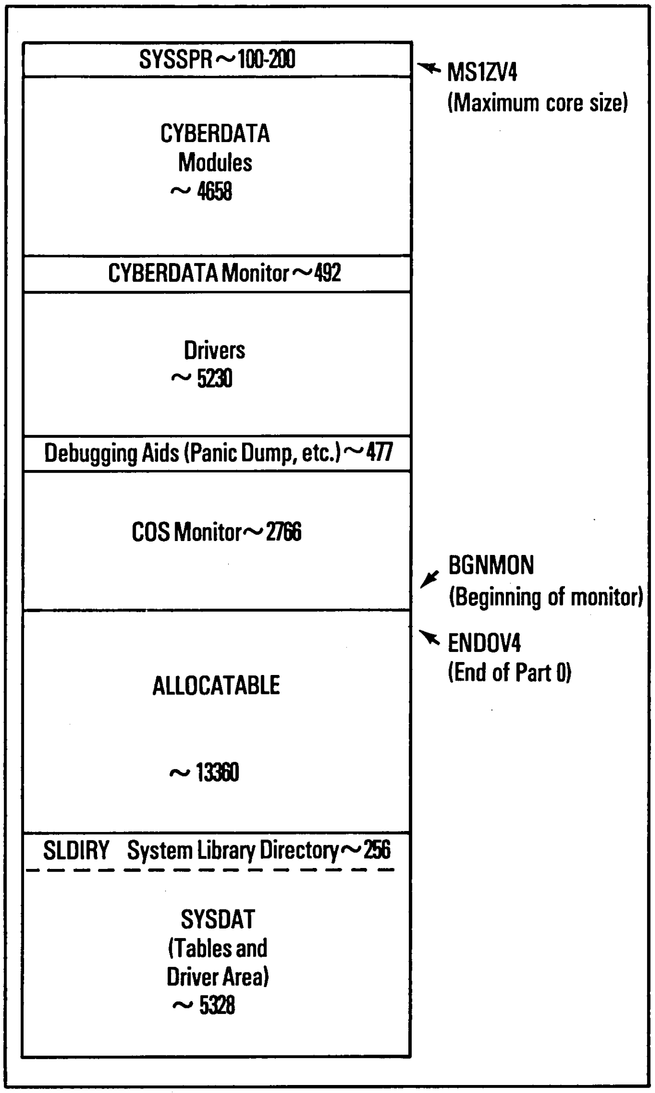


Figure 8-1. CYBERDATA Operating System Core Memory

INPUT/OUTPUT UTILITY

Via requests entered at the standard input or comment device, the background I/O utility program (IOUP) for COS enables the user to perform peripheral operations simultaneously in the background during normal foreground processing.

*IOUP Statement

The IOUP program runs in the background and is called by the job processor when the operator types an *IOUP statement. IOUP reads control statements from the standard input device. The operator may switch control to the standard comment device by entering an *K,I4. it continues to process the IOUP requests until it is terminated by either an OUT or an *Z statement, followed by a RETURN.

All operator requests are limited to 40_{10} characters on the input device.

An *IOUP statement initiates the program, which alerts the operator that it is in core and ready for use by typing:

```
UTILITY IN
NEXT IOU
```

After the request is completed, and all messages associated with the completed request have been typed, the IOUP program types:

```
NEXT IOU
```

The IOUP program then waits for the next request from the operator. To terminate the IOUP program, the operator must complete the following items:

- Type: OUT
- Press: RETURN

The IOUP program types:

```
UTILITY OUT
```

To abort a request which is being executed, the operator must complete the following steps:

- Press: MANUAL INTERRUPT
- Type: *Z
- Press: RETURN

The job processor terminates the processing of the request.

Theory of Operation

The three tasks performed by an IOUP request are:

- Transfer of data
- Comparison of data
- Motion control request

Execution of these tasks is employed by the IOUP program through the use of the following devices referred to by their logical units:

- 1729-3 Card Reader
- 615-93/73 Magnetic Tape
- 1742 Line Printer

References to the I/O devices used by the IOUP program are made by logical units or the standard I, P, or L notation assigned by the system or the job processor.

TRANSFER AND COMPARISON OF DATA

Transfer and comparison of data are performed via this method. Data is read from one logical unit and written to or compared with the data on the second logical unit. Multiple copies of the input data can be made on the output device.

MOTION CONTROL REQUEST

The tape motion request may be forward or backward skipping of the specified number of files or records, setting the density, writing an end-of-file, or rewinding or unloading a magnetic tape.

Data Transfer

The IOUP program uses two methods for data transfer. The method to be used is governed by the following:

- Intermediate storage is used if the input and output units are the same and/or multiple copies of the input data are requested (method 1).
- Intermediate storage is not used (record-by-record transfer occurs) if the input and output units are not the same and only one copy of the input data is requested (method 2).

Method 1

Read the entire input data from the input, and write to an intermediate storage (scratch unit). For transfer requests, the input data from the intermediate storage is read one record at a time and then output to the specified device. If multiple copies of the output are required, the intermediate storage is read over again and output until the repeat count is satisfied.

Using this method, the amount of data to be transferred is restricted by the size of the available scratch unit. if the input data to be transferred is specified to be larger than the available scratch or while actually writing, the scratch is exhausted, and the request is aborted.

Method 2

The intermediate storage is not used in the execution of an IOUP request. Data transfer from one record of the input device is read and written directly to the specified output device. This record-by-record transfer continues until either the specified amount of data is transferred or a physical end-of-the-input device is detected. Only one copy of the data can be made.

Data Verification

The IOUP program uses one of the following methods for data verification.

- Two of the same logical units with data to be compared (method 1)
- Two different logical units with data to be compared (method 2)

Method 1

When the two logical units are the same, the entire data from the first logical unit is read and written on intermediate storage. This process continues until all the specified number of records are read and written or until the end of data on either logical unit is detected. After the completion of reading from the first logical unit and writing on intermediate storage, one record at a time is read from the second logical unit and compared with the intermediate storage for record length and data matching. This comparison continues until the end of the data on the second unit or to the end of intermediate storage. When a mismatch occurs, a diagnostic is output.

Method 2

When the data to be compared is on two different logical units, a record from the first logical unit is read and compared with the corresponding record of the data on the second logical unit. The two records are checked for the same number of words and word-by-word likeness. The comparison continues until all the specified number of records are compared or until the end of data on either logical unit is detected. When a mismatch occurs, a diagnostic is output.

Data Record Size

The maximum data record sizes for the I/O devices that are used by an IOUP request are as follows:

- Card reader — 192 words (assembly parameter)
- Magnetic tape — 192 words (assembly parameter)

Peripheral Operations Performed by IOUP Program

The peripheral operations performed by the IOUP program are categorized as follows:

- Data transfer requests

  Card to magnetic tape — CM,u_1,u_2,m,x

  Card to printer — CL,u_1,u_2,m,x

  Magnetic tape to printer — ML,u_1,u_2,R/F,n,m,x

  Magnetic tape to magnetic tape — MM,u_1,u_2,R/F,n,m

- Data verification requests

  Card and card — VCC,u_1,u_2,x

  Card and magnetic tape — VCM,u_1,u_2,n,x

  Magnetic tape and magnetic tape — VMM,u_1,u_2,n

- Motion control requests

  Advance unit number of files — TAF,u,n

  Advance unit number of records — TAR,u,n

  Backspace unit number of files — TBF,u,n

  Backspace unit number of records — TBR,u,n

  Rewind unit — TRW,u

  Write end-of-file mark on unit — TEF,u

  Set density of unit — TSD,u,d

  Unload unit — TUL,u

The u, u_1, u_2, and u_3 of all the preceding IOUP requests indicate the devices used by the job processor to execute an IOUP request. If the type of the device indicated by the I, P, L, u, u_1, u_2, or u_3 do not match the type of the IOUP request, the request is rejected.

Data Transfer Requests

The following paragraphs discuss the data transfer requests.

CARD TO MAGNETIC TAPE REQUEST

This request is used to request a card-to-magnetic-tape data transfer.

Format

The request format is:

CM,u_1,u_2,m,x

Parameters

For this format, the parameters are:

u_1—card reader unit

u_2—magnetic tape unit

m—number of times the input deck is to be copied

x—optional

0 or blank—format of input data is 1700-formatted binary/ASCII

1 to 99999—80-column card image in binary

Request Function

The CM request performs the following functions:

- Card deck input at the card reader unit u_1 is written m (1 to 10) times on the magnetic tape on u_2.
- An end-of-file mark is written at the end of each copy on the magnetic tape unit.
- The end-of-input information occurs when the card reader hopper is empty.
- When an end-of-tape condition on magnetic tape is detected, a diagnostic is printed.

- The writing of data on magnetic tape u_2 begins at the current physical position of the tape. The tape density is determined by the setting on unit u_2.

- On completion of the request, the number of records (one record = one card) transferred is typed.

CARD TO PRINTER REQUEST

This request is used to transfer data from cards to the printer.

Format

The request format is:

CL,u_1,u_2,m,x

Parameters

The parameters are as follows:

u_1—card reader unit

u_2—printer

m—number of listings required

x—optional

x or blank—first character of the card record is not interpreted as a carriage control function; it is printed as a data character

1 to 99999—first character of the card record is interpreted as a carriage control function

Request Function

The CL request functions as follows:

- Card deck input at the card reader unit u_1 is written m (1 to 10) times on the printer of unit u_2.

- The mode of the data transfer is assumed to be 1700-formatted binary/ASCII mode used for the input card deck. Non-ASCII input data results in a garbled printout.

- A paper eject occurs at the end of each listing.

- The end of the information occurs when the card reader hopper is empty.

- On completion of the request, the number of records (one record = one card) is typed.

MAGNETIC TAPE TO PRINTER REQUEST

This request transfers data from magnetic tape to the printer.

Format

The format is:

ML,u_1,u_2,R/F,n,m,x

Parameters

The format parameters are defined as follows:

u_1—magnetic tape unit

u_2—printer

R—n is the number of records to be transferred

F—n is the number of files to be transferred

n—number of records or files to be transferred (refer to R,F)

m—number of listings required

x—optional

0 or blank—first character of magnetic tape record is not interpreted as a carriage control function; it is printed as a data character

1 to 99999—first character of magnetic tape record is interpreted as a carriage control function

Request Functions

The ML request completes the following functions:

- The magnetic tape on unit u_1 is read in, and the specified number of records/files are printed on printer u_2.

- Each end-of-file mark encountered on magnetic tape causes a page eject.

- At the end of each listing, a page eject occurs.

- The number of records or files transferred is less than the specified number if the end of tape is detected before all specified input data has been read in.

- The mode of data transfer is 1700-formatted binary/ASCII mode of data on magnetic tape. The magnetic tape is read by a format read and the data is written in the mode of the input data. The printout of binary input data, however, is garbled.

- On completion of the request, the number of records or files transferred is typed.

MAGNETIC TAPE TO MAGNETIC TAPE REQUEST

The MM request transfers data from magnetic tape to magnetic tape.

Format

The request format is:

MM,u_1,u_2,R/F,n,m

Parameters

The following parameters are used:

u_1—input magnetic tape unit
u_2—output magnetic tape unit
} u_1 and u_2 can refer to the same unit

R—n is the number of records to be transferred (1 to 99999)

F—n is the number of files to be transferred (1 to 99999)

n—number of records or files to be transferred (refer to R,F)

m—number of copies to be made (1 to 10)

Request Function

The MM request is used for the following functions:

- The magnetic tape in unit u_1 is read in, and the specified number n of records or files are written m times on magnetic tape unit u_2. Units u_1 and u_2 can refer to the same unit.

- The mode of data transfer is that of the data on the input magnetic tape. The magnetic tape is format read in 1700-formatted binary/ASCII mode, and the data is written in the format mode of input data. At the end of each copy of data, an end-of-file mark is written on the output magnetic tape. Each end-of-file mark detected on the input magnetic tape is counted as one record when R is specified. The number of records or files transferred is less than the specified number if the end-of-tape mark is sensed on the input magnetic tape before n is satisfied.

- When an end-of-tape mark is detected on the output magnetic tape u_2, a diagnsotic is typed.

Data Verification Requests

The data verification requests are discussed in the following paragraphs.

CARD AND CARD REQUEST

The VCC request is used for card-and-card verification.

Format

VCC,u_1,u_2,x

Parameters

The parameters for this request are as follows:

u_1—card reader unit 1

u_2—card reader unit 2

x—optional

0 or blank—format of data to be compared is 1700-formatted binary/ASCII

1 to 99999—format of data to be compared is 80-column binary card image

Request Function

The VCC request functions as follows:

- The two card decks to be compared are input at card reader units u_1 and u_2.

- The IOUP program compares each card, and in case of any discrepancy in the data, a diagnostic is printed. The comparison continues until the end of data on unit u_1/u_2 is detected. When verification is completed, the total number of records checked is printed (one card = one record).

CARD AND MAGNETIC TAPE REQUEST

Verification between cards and magnetic tape is accomplished via the VCM request.

Format

The request format is:

VCM,u_1,u_2,n,x

Parameters

u_1—card reader unit

u_2—magnetic tape unit

n—number of records to be compared (1 to 99999)

x—optional

0 or blank—format of data compared is 1700 binary/ASCII

1 to 99999—format of data compared is 80-column binary

Request Function

The following functions are done by the VCM request:

- The card to be compared with is read on unit u_1, until either n cards are read or the reader hopper is empty, whichever occurs first. This data is compared with each magnetic tape record until either an end-of-tape mark is detected or n (actual number of cards/records of data are read) whichever occurs first.
- At the end of verification, the total number of records checked is printed.
- If any discrepancy occurs in a record, a diagnostic is typed. An end of file on magnetic tape is counted as one record.

MAGNETIC TAPE AND MAGNETIC TAPE

Verification between magnetic tape and magnetic tape is via the VMM request.

Format

VMM,u_1,u_2,n

Parameters

The format parameters are as follows:

u_1—magnetic tape 1 unit } u_1 and u_2 can refer
u_2—magnetic tape 2 unit } to the same unit

n—number of records of magnetic tape to be compared (1 to 99999)

Request Function

- u_1 Equal to u_2

  Magnetic tape one on unit u_1 is read by a format read in the mode of the data on the tape until the specified number of n records of data is read. If the end-of-tape mark is detected on tape 1 before the specified number n has been read, the number of records actually read is the number of records to be compared. An end of file counts as one record.

  Magnetic tape 2 on unit u_2 is format-read in the mode of the data of the corresponding record on tape 1, and is compared for a match in the number of words and word-by-word equality in each record. A discrepancy causes a diagnostic to be typed. At the end of verification, the total number of records checked is typed. Verification also ends when the end-of-tape mark is detected on tape before the specified number of records have been checked.

- u_1 Not Equal to u_2

  The comparison of the data is one record from each of the two units. The verification ends when either the specified number of records are checked or the end of tape on u_1 and u_2.

Motion Control Requests

The motion control requests are discussed and defined in the following paragraphs.

ADVANCE UNIT NUMBER OF FILES REQUEST

To advance a unit number of files, the TAF request is used.

Format

The request format is:

TAF,u,n

Parameters

u—magnetic tape unit

n—number of files to be advanced (1 to 4095)

Request Function

The TAF request function is:

- Magnetic tape on unit u is advanced n number of files. When the end of tape is detected before all the n files have advanced, the tape motion stops at the end-of-tape marker. A typed message indicates the total number of files advanced.

ADVANCE UNIT NUMBER OF RECORDS REQUEST

This request advances the unit a number of records.

Format

The format is:

TAR,u,n

Parameters

The following parameters apply to this request:

u—magnetic tape unit

n—number of records to be advanced (1 to 4095)

Request Function

The TAR request functions as follows:

- Magnetic tape on unit u is advanced n number of records. When the end-of-tape mark is detected before all n records are advanced, the tape motion stops at the end-of-tape mark. A message is typed to indicate the total number of records advanced. An end-of-file mark is counted as one record.

BACKSPACE UNIT NUMBER OF FILES REQUEST

This request backspaces the unit by a number of files.

Format

The format is:

TBF,u,n

Parameters

The TBF request uses these parameters:

u—magnetic tape unit

n—number of files to be backspaced (1 to 4095)

Request Function

The TBF request functions as follows:

- The magnetic tape on unit u is backspaced n number of files. If the load point is detected before all the specified files are backspaced, tape motion stops at the load point. A typed message indicates the number of files backspaced.

BACKSPACE UNIT NUMBER OF RECORDS REQUEST

This request backspaces the unit a number of records.

Format

The format is as follows:

TBR,u,n

Parameters

These parameters are used for the TBR request:

u—magnetic tape unit

n—number of records to be backspaced (1 to 4095)

Request Function

The TBR request functions as follows:

- The magnetic tape on unit u is backspaced n number of records. An end-of-file mark is counted as one record. If the load point is detected before all the specified records are backspaced, tape motion stops at the load point. A typed message indicates the number of records that are backspaced.

REWIND UNIT REQUEST

The rewind unit request is TRW.

Format

The format is:

TRW,u

Parameter

The TRW parameter is defined as follows:

u—magnetic tape unit; the magnetic tape unit u is rewound to its load point

UNLOAD UNIT REQUEST

The TUL request is the unload unit request.

Format

The format is:

TUL,u

Parameter

The following parameter applies to the TUL request format:

u—magnetic tape unit

WRITE END-OF-FILE MARK ON UNIT REQUEST

This request is used to write the end-of-file mark on magnetic tape at unit u.

Format

The format is:

TEF,u

Parameter

The TEF parameter is:

u—magnetic tape unit

SET DENSITY OF UNIT REQUEST

This request sets the density of the magnetic tape on unit u to the specified density.

Format

The format is:

TSD,u,d

Parameters

These parameters are used for the TSD request:

u—magnetic tape unit

d—density code

0—do nothing

5—select 556 bpi

8—select 800 bpi

16—select 1,600 bpi

COSY PROGRAM

The 1700 COSY program provides a means of compressing information in source decks by replacing three or more blanks on a card with two special ASCII characters. COSY compresses Hollerith source decks and converts the Hollerith code to ASCII code. The resulting deck, called a COSY deck, is in COSY format. COSY reduces average deck size by about 60 percent.

A COSY library consists of a group of COSY decks. Each COSY deck is preceded by a COSY deck identifier card and is terminated by an end-of-deck character. The COSY library may be written on magnetic tape or punched cards. The library is terminated by an END/ card followed by an end-of-file mark.

The COSY program is called from mass storage by typing *COSY and by pressing the RETURN button at the teletypewriter console or by using a *COSY punched card. There are no parameters for the teletypewriter call to COSY or for the *COSY card. A COSY revision deck follows the call to COSY. COSY revision decks allow the user to prepare, revise, or copy COSY decks. The revision decks also provide for the preparation, update, or copy of COSY libraries. COSY may be used with any source language that does not use COSY control statements. COSY output may be in Hollerith or COSY (compressed ASCII) format and may be listed or punched or the Hollerith may be sent to a compiler or assembler.

COSY Cards

COSY revision decks are comprised of COSY control cards and new source cards. There are seven COSY control cards (MRG/, DCK/, CPY/, DEL/, INS/, REM/, and END/) and two deck identifier cards (HOL/ and CSY/). The fields for all COSY control and identifier cards (except DEL/ and INS/) are in the standard format shown in figure 8-2.

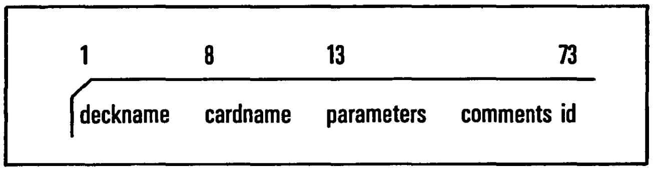


Figure 8-2. Standard Field Format for COSY Cards

In figure 8-2, the following parameters apply:

deckname—columns 1 through 6; the name of a deck in a COSY library that is to be modified or copied. Deckname is used only on DCK/, CPY/, HOL/, and CSY/ cards. The field is blank on all other COSY cards.

cardname—columns 8 through 11; name of COSY control card

parameters—start in column 13; parameters are terminated by a space

comments—can start in any column after the terminating space for parameters; comments may run through column 72 and are optional

id—columns 73 through 75; a three-character deck name identifier; used only on DCK/, HOL/, and CSY/ cards

The control card fields for DEL/ and INS/ cards are shown in figure 8-3.

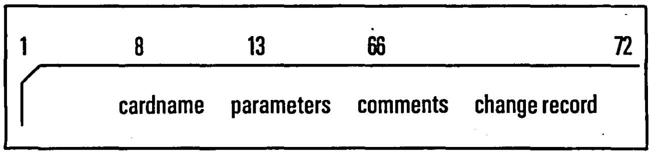


Figure 8-3. DEL/ and INS/ Card Field Format

The cardname, parameter, and comment field are the same for DEL/ and INS/ as for the standard card in figure 8-2, except that the comment field ends in column 65. A change record field is added to these cards to add change identification information. The change record field is a seven-character field (columns 66 through 72) which is used to identify the type, nature, or data of a change. COSY writes an asterisk in column 73 and in the contents of the change record field in columns 74 through 80 of each new source card following the INS/ or DEL/ card. This provides a means of identifying new or changed source cards when a COSY deck is listed. Adding change record information on an INS/ or DEL/ card is a user option. It is not required input to COSY.

MRG/ CARD

An MRG/ card directs COSY to merge two revision decks.

Format

The MRG/ card format is shown in figure 8-4.

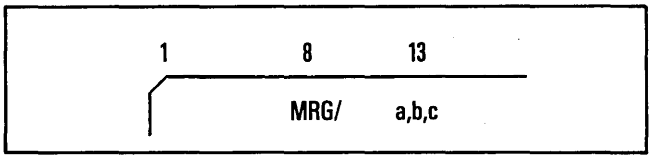


Figure 8-4. MRG/ Card Format

Parameter

The card format parameters are:

a,b,c—This specifies the actions to be taken. This card directs COSY to merge the revisions deck on logical unit a with the revisions deck on logical unit b and write a merged revisions deck on logical unit c.

If revisions between a and b conflict, revisions from a are used. The conflicting revisions from b are listed with asterisks in columns 2 through 5 on the standard print device and are not written on unit c. If either a or b is missing or zero, COSY assumes that the decks are on the standard input device. If c is missing or zero, the standard output device is used.

If a and b are the same logical unit, the first revisions deck is written onto mass storage and is then merged with the second revisions deck on the logical unit. Revisions on mass storage have priority if conflicts occur.

The DCK/ card in the merged deck is the DCK/ card from unit a. The merge terminates when the END/ card on both decks is read.

COSY locates a DCK/ card on unit a and searches unit b until the deck names match. Intervening decks on unit b are copied to unit c. If COSY reaches the end of the revisions deck on unit b before obtaining a match, it treats all the remaining decks on unit a as new decks and inserts them at the end of the merged deck. If revisions are to be input from different input devices, logical units must be specified on the MRG/ control card.

DCK/ CARD

A DCK/ card identifies the COSY or Hollerith deck to be updated or created and specifies the actions to be taken with the new deck.

Format

The DCK/ card format is shown in figure 8-5.

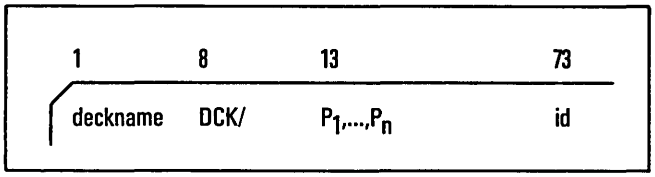


Figure 8-5. DCK/ Card Format

Parameters

The following parameters are used in the DCK/ card field format:

deckname—names the COSY or Hollerith deck to be processed

$P_1,...,P_n$—specifies the actions to be taken. All parameters are optional, can be in any order, and are separated by commas. Blanks are not allowed in the parameter field. Parameters have the form p, or p = lu, or D = deckname where p is I, C, H, or L, and lu is the logical unit on which input or output occurs. Deckname specifies a new deckname for the COSY output.

I parameter—input (I = lu)

I specifies the logical unit containing the COSY or Hollerith source deck(s) to be updated or created. If the parameter is absent or just I, COSY assumes the source deck is on the COSY standard input device.

If I = lu is used, and lu is the system standard input unit, COSY assumes that a new deck is being added to the COSY library. If the first card after the DCK/ card is a source deck identifier, COSY processes the deck until an END/ card is read. Additional new source decks may follow. Each new deck must begin with a source deck identifier card and must end with an END/ card. The card which follows the END/ card must be a DCK/ card, MRG/ card, or another END/ card to mark the end of the revision deck.

If the first card after the DCK/ card is not a COSY or Hollerith source-deck identifier card, COSY assumes that the cards which follow the DCK/ card are revision cards and the COSY source deck will follow the revision cards. COSY reads the revision cards and places them on the mass storage scratch area until an END/ card is read. Then COSY reads the new COSY source deck, which must follow the revision cards, and modifies the new deck according to the revision cards.

If I = lu is used, and lu is not the system standard input, COSY reads the revision cards from the system standard input unit and the source deck which is specified by the DCK/ card from unit lu. Then COSY updates the source deck according to the revision cards.

C parameter—COSY output (C = lu or C)

This specifies the device which is to receive COSY output. If C is absent, there is no COSY output. If just C is used, COSY output is on the standard output device. C cannot be equated to the unit which contains the current COSY library.

H parameter—Hollerith output (H = lu or H)

This parameter specifies the device which is receiving Hollerith output. If H is absent, there is no Hollerith output. If just H is used, Hollerith output is on the COSY standard output device.

D parameter—deckname (D = name)

The D parameter changes the name of the COSY or Hollerith deck. COSY uses the six characters (including blanks and commas) following D = for the new deckname.

NOTE

If name is fewer than six characters and an I, C, or H parameter follows it, COSY misinterprets name.

id parameter (id)

This three-character field changes the COSY or Hollerith deck identifier. If id is blank, the old deck identifier on the HOL/ or CSY/ card is used.

L parameter—list (L = lu or L)

The L parameter specifies that a listing, in decompressed Hollerith form, of the deck is to be made on logical unit lu. If just L is used, the listing is on the COSY standard list device.

DEL/ CARD

COSY deletes a specified number of cards from a previously defined input deck and inserts any Hollerith source cards immediately following the DEL/ card up to the next COSY control card. A DEL/ card has two forms as shown in figures 8-6 and 8-7.

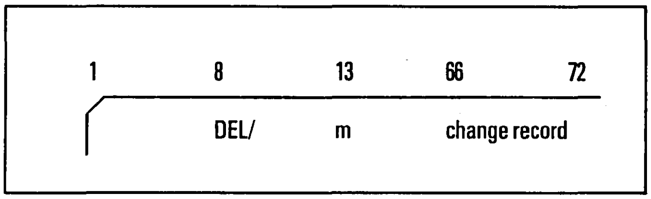


Figure 8-6. First DEL/ Card Format

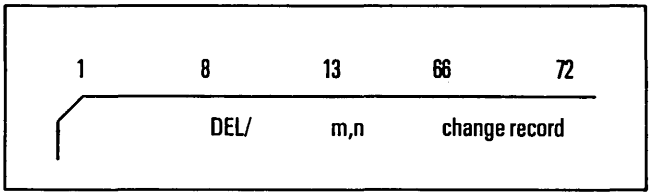


Figure 8-7. Second DEL/ Card Format

In figure 8-6, card m is deleted; in figure 8-7, cards m through n are deleted. The unsigned decimal numbers m and n are the sequence numbers in columns 76 through 80 of the Hollerith source cards. Sequence number m must be less than n.

The number of Hollerith cards following a DEL/ card need not equal the number of cards being deleted.

INS/ CARD

COSY inserts the Hollerith source cards immediately following an INS/ card (figure 8-8) into the new COSY or Hollerith deck.

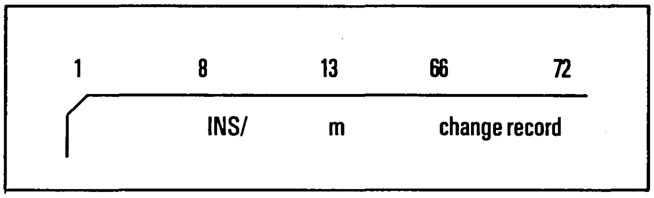


Figure 8-8. INS/ Card Format

The Hollerith source cards are inserted after sequence number m and are found in columns 76 through 80 of the Hollerith source cards.

REM/ CARD

The REM/ card is used to remove the DEL/ or INS/ card and all Hollerith source cards that follow. This operation occurs only when two revisions decks are being merged. A REM/ card has the two forms shown in figures 8-9 and 8-10.

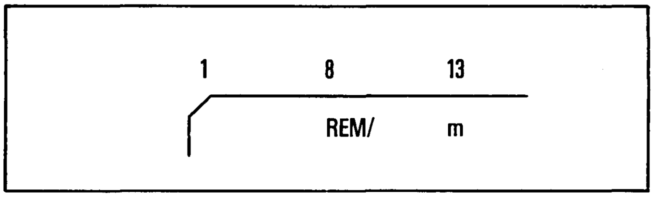


Figure 8-9. First REM/ Card Format

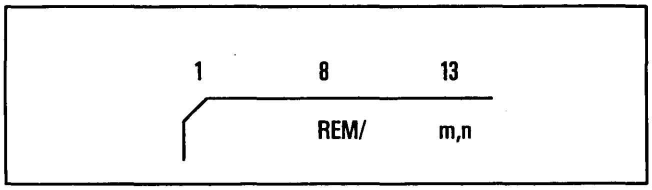


Figure 8-10. Second REM/ Card Format

The sequence numbers m and n must match the sequence numbers on DEL/ or INS/ control cards in the revisions deck that is being merged.

A REM/ card detected when COSY is not merging is ignored.

CPY/ CARD

The CPY/ card causes the COSY library to be copied onto a logical output unit. The CPY/ card has the two forms shown in figures 8-11 and 8-12.

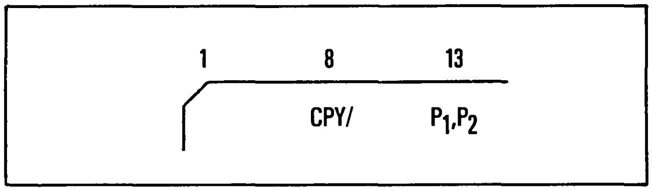


Figure 8-11. First CPY/ Card Format

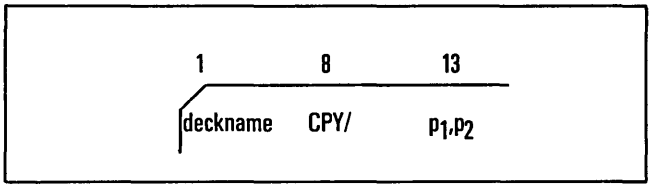


Figure 8-12. Second CPY/ Card Format

The first form, without the deckname, causes the COSY library to be copied from its current position to the end of the library. The second form, with a deckname specified, causes the COSY library to be copied from its current position through the named deck. COSY places an END/ card at the end of the new library, followed by an end-of-file mark.

The COSY library can be positioned at the beginning of any deck by the use of a CPY/ card on which only the deckname and the I parameter are specified. This card positions the COSY library to the beginning of the deck which immediately follows the named deck.

Parameters

The p parameters specify the logical I/O units that are used to copy the COSY library. These parameters can occur in any order and are in the form p = lu where:

p = I or C

lu = a logical I/O unit

I = lu—I specifies the logical unit, lu, from which
or I the COSY library is copied. If the I parameter is omitted or just I is used, the COSY library is copied from the COSY standard input device.

C = lu—C specifies the logical unit, lu, to which the
or C COSY library is copied. If just C is used, the COSY library is copied onto the COSY standard output device. If C is omitted, there is no COSY output.

As each COSY deck is read from input unit I and is copied on output unit C, the deckname is listed on the COSY standard print device. For example:

```
Deckname    CSY/    *COPIED*
```

For each deck that is read but not copied, the *COPIED* notation is omitted. For example:

```
Deck 1      CSY/    *COPIED*

Deck 2      CSY/    *COPIED*

   .          .         .

   .          .         .

   .          .         .

Deck 9      CSY/    *COPIED*

Deck 10     CSY/

   .          .         .

   .          .         .

   .          .         .

Deck 14     CSY/

Deck 15     CSY/    *COPIED*

Deck 16     CSY/    *COPIED*

   .          .         .

   .          .         .
   .          .         .
```

Decks 10 through 14 were read, but they were not copied.

END/ CARD

The END/ card (figure 8-13) terminates Hollerith input decks, COSY libraries, Hollerith input libraries, and revisions decks.

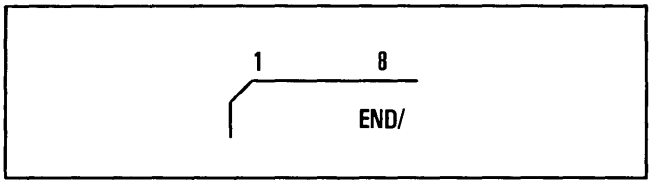


Figure 8-13. END/ Card Format

HOL/ CARD

When a Hollerith deck is input, the first card must be a Hollerith deck identifier (figure 8-14).

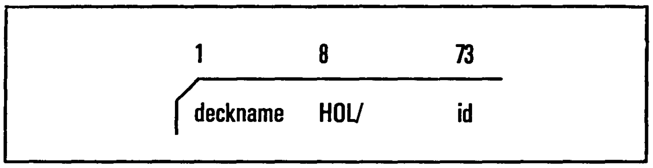


Figure 8-14. HOL/ Card Format

Parameters

The HOL/ card parameters are:

deckname—names the Hollerith deck being processed

id—three-character deck identifier

A Hollerith deck identifier is not produced for a Hollerith output deck.

CSY/ CARD

When COSY output is requested on the DCK/ card (figure 8-15), COSY generates a COSY deck identifier card as the first card of the COSY output deck. COSY deck identifiers must also precede COSY decks on input.

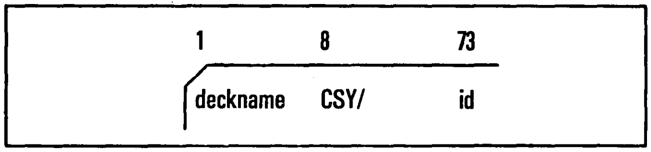


Figure 8-15. CSY/ Card Format

Parameters

The CSY/ card format parameters are as follows:

deckname—names the COSY deck being processed

id—three-character deck identifier of original deck

Sample COSY Revision Decks

The following sample COSY revision decks illustrate the use of COSY control cards.

GENERATING A COSY LIBRARY

The example in figure 8-16 generates a COSY library from two Hollerith source decks and places the library on output unit 13. The system standard input unit (card reader) is unit 10.

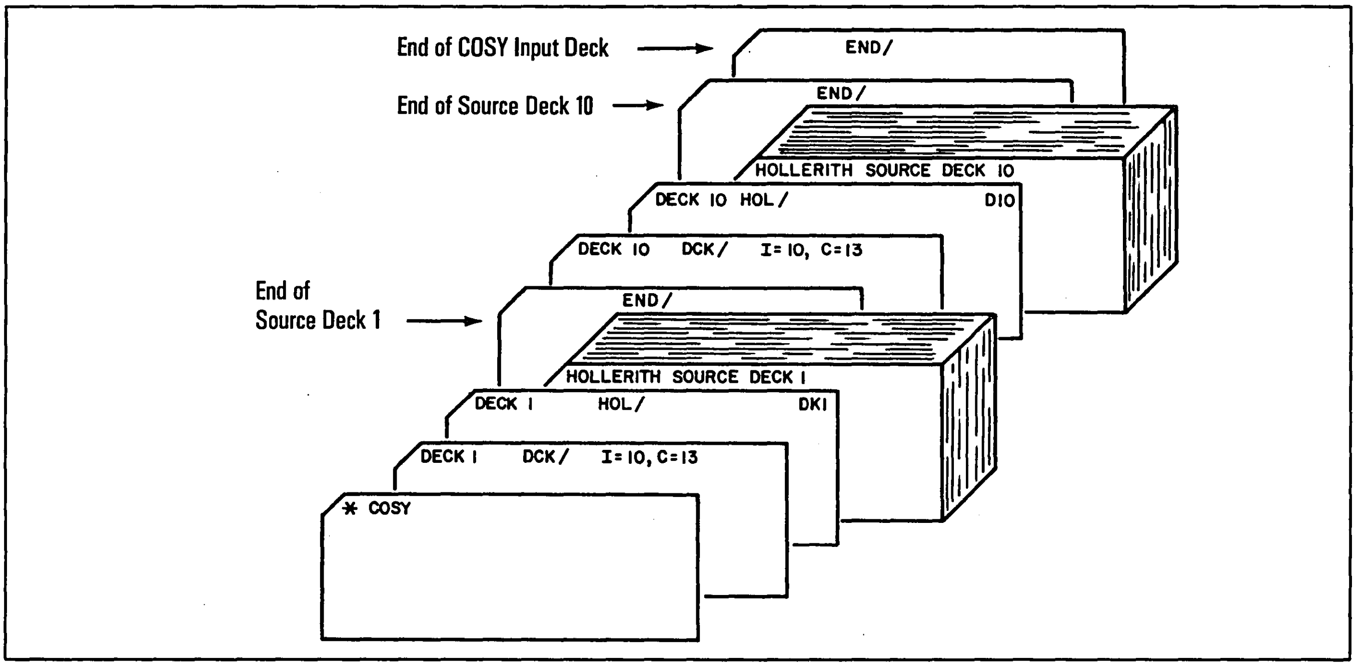


Figure 8-16. COSY Library Generation

UPDATING COSY DECKS

The example in figure 8-17 updates three COSY decks and places the updated decks on logical unit 7. Two of the COSY decks are on logical unit 6 and the third deck (deck 3) is input following the revision decks. The system standard input unit (card reader) is unit 10.

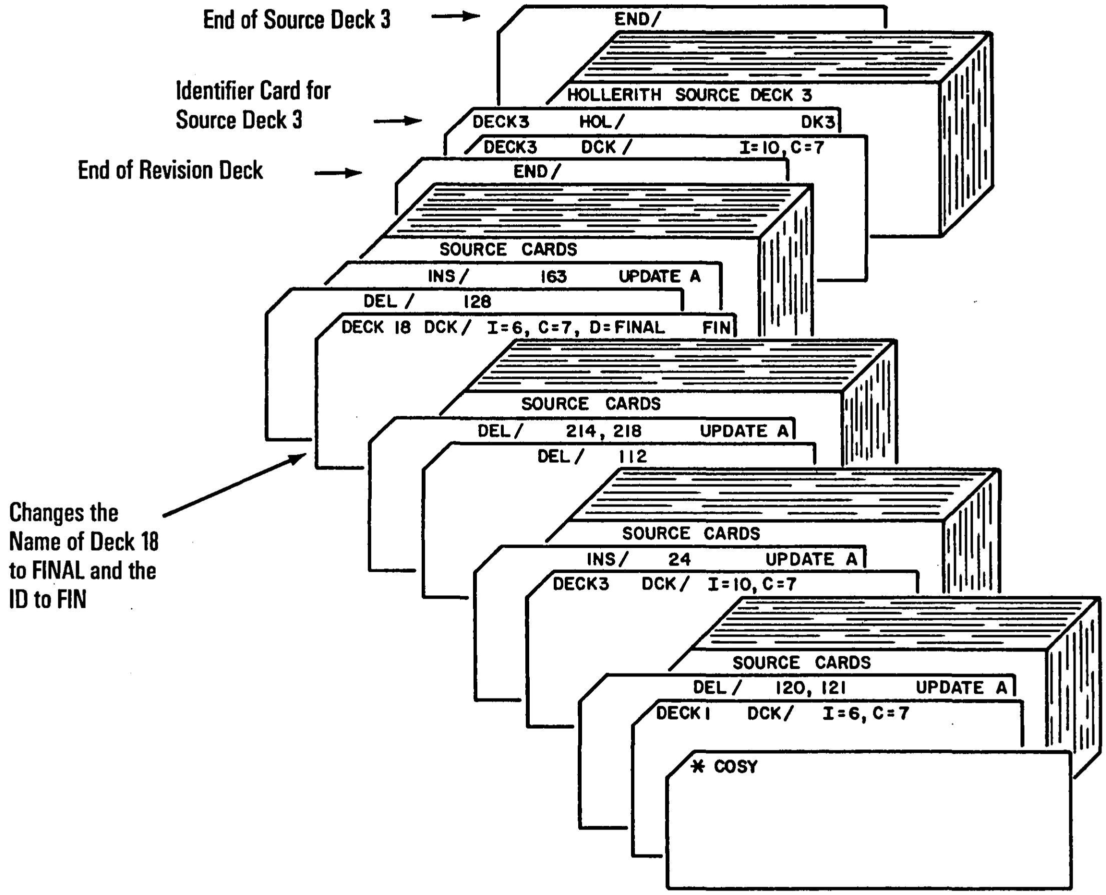


Figure 8-17. COSY Deck Updating

Using the CPY/ Card to Update a COSY Library

The example in figure 8-18 shows how to update a COSY library by using the CPY/ card to replace five old COSY decks with five new COSY decks. Logical unit 13 contains the old COSY library (decks 1 through 24) and logical unit 10 contains five replacement decks. The new COSY library is output on logical unit 6.

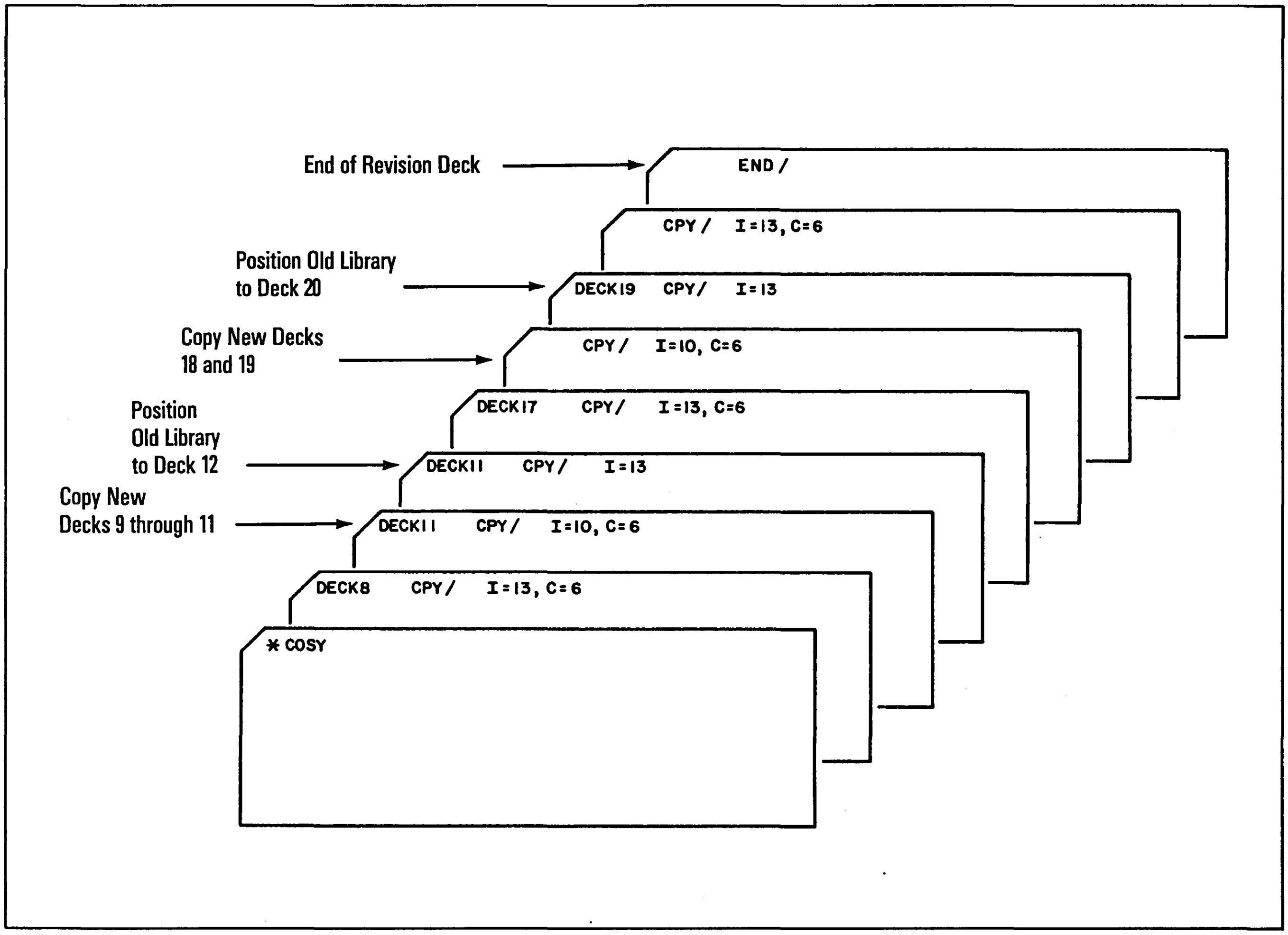


Figure 8-18. Updating COSY Library Via CPY/ Card

MERGING TWO REVISION DECKS

The following are two examples of merging revision decks. Example 1 merges two decks which both appear on the same input unit. Example 2 merges two decks that appear on different input units.

Example 1

This job merges two revision decks (A and B) which appear on logical unit 10 (card reader) and writes the merged output as a revision deck in Hollerith format on logical unit 13. Figure 8-19 shows the two revision decks, and figure 8-20 shows the new (merged) revision deck.

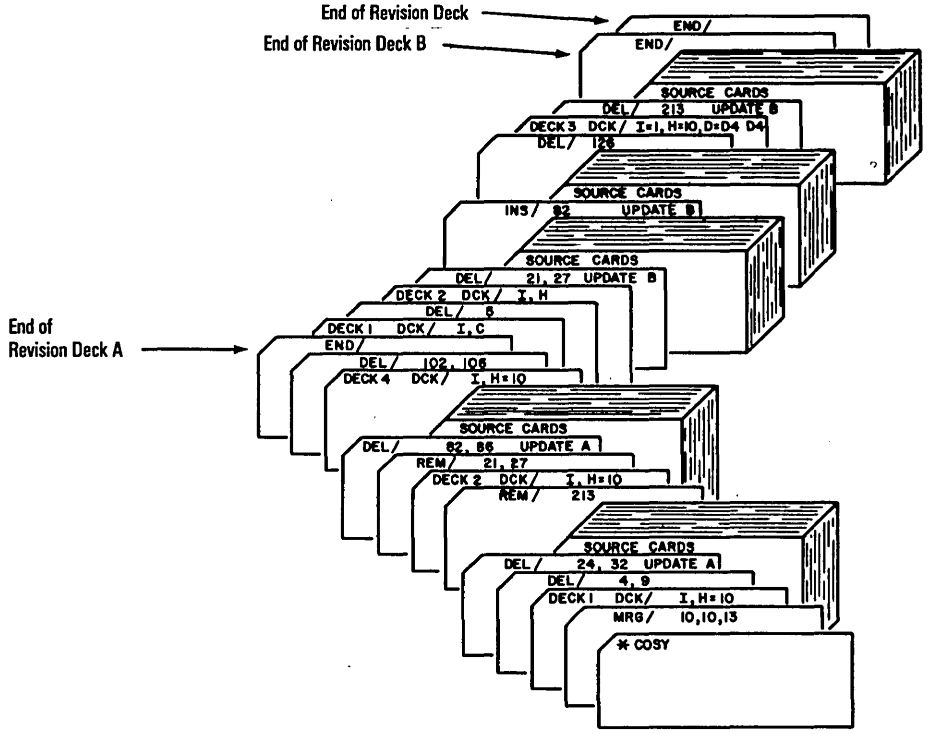


Figure 8-19. First Example of Merging Two Revision Decks

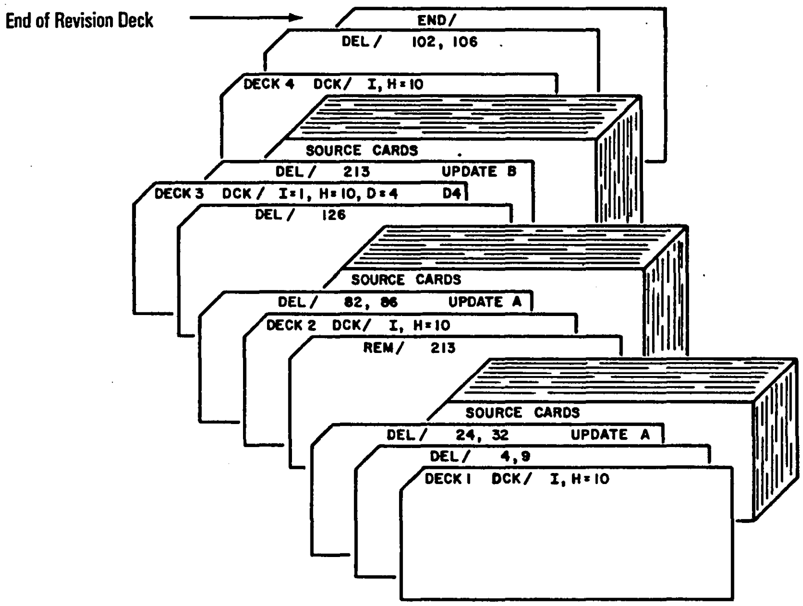


Figure 8-20. Resultant Merged Revision Deck

Example 2

This job (figure 8-21) merges two revision decks (A and B) and writes the merged revision deck on logical unit 13 (magnetic tape). Revision deck A is on logical unit 10 (card reader), and revision deck B is on logical unit 6 (magnetic tape).

Revision deck B, which is input from magnetic tape, is shown in figure 8-22.

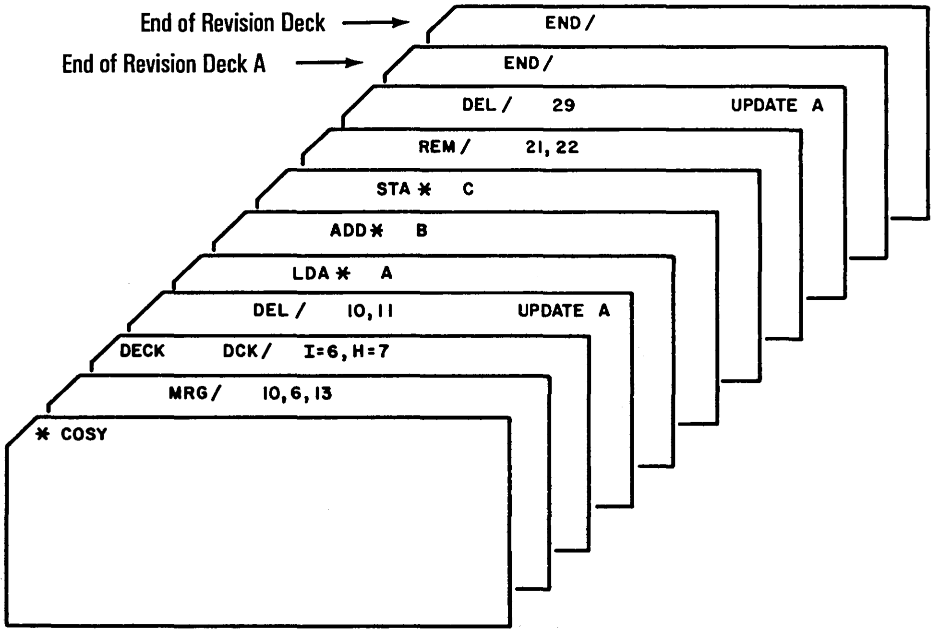


Figure 8-21. Second Example of Merging Two Revision Decks

```
DECKA          DCK/   I = 6, H = 7
               DEL/   10,11          (update B)
               LDA*   A
               SUB*   B
               STA*   C
               DEL/   21,22          (update B)
(source cards for insertion between COSY cards 20 and 25)
               END/                  (end of revision deck B)
```

Figure 8-22. Revision Deck B for Example 2

The merged revision deck, which is output on magnetic tape, is shown in figure 8-23.

Since revision deck A was the primary deck, the DEL/ 10,11 card and the insert cards that follow it in DECKA take precedence over the DEL/ 10,11 card and the insert cards in revision deck B. Also, the REM/ 21,22 cards in revision deck A remove the DEL/ 21,22 card and the following source cards from revision deck B. The DEL/ 29 card from revision deck A is added to the merged revision deck.

CONVERTING COSY DECKS TO A HOLLERITH LIBRARY

The example in figure 8-24 shows a job that converts three COSY decks into a Hollerith library and writes the Hollerith library on logical unit 6. COSY decks 1 and 2 are on logical unit 13, and COSY deck 3 is on logical unit 7.

```
DECKA          DCK/   I = 6, H = 7
               DEL/   10,11          (update A)
               LDA*   A
               ADD*   B
               STA*   C
               DEL/   29
               END/                  (end of merged revision deck)
```

Figure 8-23. Example 2 - Merged Revision Deck

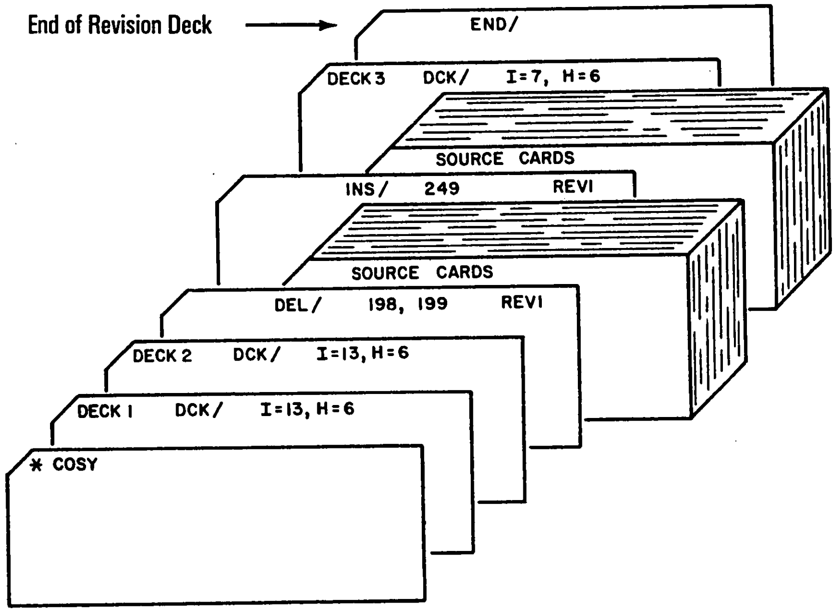


Figure 8-24. COSY Deck Conversion to Hollerith Library

COSY Library

The COSY library consists of one or more COSY decks terminated with an END/ card. The COSY deck is a series of compressed source statements that are written in ASCII format. Each COSY deck begins with a COSY deck identifier and ends with an end-of-deck character followed by an end-of-file mark.

COSY compresses a card image by inserting a special ASCII character and value for three or more sequential blanks as follows:

$5F_{16}$—special ASCII character indicating compression

$5Fxx_{16}$—indicates 3 to 62 consecutive blanks where

$$21_{16} \leq xx \leq 5D_{16} \text{ except } 26_{16}$$

$5F5E_{16}$—end of card image

$5F5F_{16}$—end of deck

The COSY library is kept on magnetic tape. The block size is 192 words, and all blocks are completely filled. Card images may be split across blocks.

Hollerith Input

A Hollerith input library is a group (one or more) of Hollerith source decks which is terminated by an END/ card. Each Hollerith source deck begins with a Hollerith deck identifier card and ends with an END/ card. The Hollerith input decks may be input from cards, a magnetic tape, or a teletypewriter.

Hollerith Output

Hollerith output consists of source statements in uncompressed Hollerith code produced from COSY input. Columns 73 through 75 of the source card images contain a deck identifier. Asterisks appear in this field if the source output was inserted by a revision deck.

Columns 76 through 80 of the Hollerith source card images contain a decimal sequence number. If new source statements are inserted with revisions that contain a DEL/ or INS/ card, COSY writes an asterisk in column 73 of each new source card image and writes the change record field (contents of columns 66 through 72 on the DEL/ or INS/ card) in columns 74 through 80 of the new source card images. If the change record field was blank on the INS/ or DEL/ card, COSY fills columns 73 through 80 with asterisks.

Hollerith output is terminated with an end-of-file mark. COSY writes an end-of-file mark and rewinds the tape on completion of the COSY run.

Revision Deck

A revision deck is a group of COSY control cards and new source cards which are used to update or revise an existing COSY library. The first card of a revision deck must be a DCK/, MRG/, or CPY/ control card; and the last card must be an END/ control card. The new source cards, if used, must follow an INS/ or a DEL/ control card. All cards are in Hollerith code.

The revision deck is input to COSY on the system standard input device. If the source deck, which is to be revised, is on the system standard input device, COSY stores the revision deck on mass storage scratch until the source deck has been read. The revision deck is stored as card images with 40 words per sector.

Listings

Under normal operation, COSY lists revisions from the revision deck as they occur on input. When merging two revision decks, however, COSY lists the final merged revision deck on the standard print device. Asterisks in columns 2 through 5 indicate the card was not used in the COSY operation. Columns 6 through 85 contain the revision input card. If the L parameter is not present on the DCK/ card and revision cards follow the DCK/ card, the revision cards are listed.

Messages

COSY error messages are written on the COSY standard list device. The format is:

COSY Cxx

In the format, xx indicates the error code. Refer to appendix E to interpret COSY error messages.

At the end of a COSY job, the following message is written on the COSY standard list device (only if errors exist):

xx ERRORS

In this message, xx is a decimal count of errors in the COSY job.

At various times during a COSY job, the following message may be written on the system standard comment device:

REWIND LU xx

In this message, xx (decimal) indicates the logical unit that is to be rewound. The operator must enter any value through the system standard input comment device after rewinding the unit to tell COSY that the unit has been rewound.

SKELETON EDITOR (SKED)

A skeleton is a file which consists of requests to the installation file builder program, LIBILD. These requests specify the order and identification of the binary programs that are to be retrieved from a set of library programs and included in the installation file which is being built. The skeleton itself contains no binary programs; it merely has commands that specify which programs are to be put onto the installation file and which is the output from LIBILD. The skeleton may also contain LIBEDT and system initializer control statements which will be included in the installation file.

Sample SKED Statements

Some examples of typical skeleton statements are given in table 8-1.

Table 8-1. Typical SKED Statements

Statement	Function
*B'SYSDAT'	Includes binary program SYSDAT
*S,ENDOV4,7FFF	Assigns value 7FFF to entry point name ENDOV4
*YM,EFILE,1	Defines system directory entry EFILE as ordinal
*L	Loads part 0 core resident programs
*M	Loads system library ordinal
*K,I5	Changes standard input to LU 5
*P	Produces absolute record
*END	End of the skeleton file

SKED Function

The purpose of the skeleton editor (SKED) is to provide the means for modifying a skeleton to allow changes to an existing system. SKED cannot be used to modify the SYSDAT program, but it can be used to redefine the order and content of the program and the system libraries. In addition to the modification commands, SKED has the facility of listing part or all of the skeleton, resequencing the record numbers, and allowing tape motion control.

Generating or Modifying SKED

To generate a skeleton of a system, it is necessary to read the system installation file into SKED via a BUILD command. To change the skeleton, the user has a number of commands which enable him to add, delete, or change the statements in the file. The modified skeleton may then be output on a convenient medium and may be used as input to LIBILD.

Executing SKED

SKED runs under the control of the job processor. It can be brought into execution by entering a *SKED command. On the console, SKED identifies itself by typing:

SKED IN

This is followed by:

NEXT

SKED types NEXT whenever it is ready to receive a command. The user then enters a valid command, followed by a carriage return. A valid command consists of a unique command name followed by any necessary arguments. The command name may be abbreviated to the least number of letters that permit the name to remain unique. It may not contain more letters than the forms given in table 8-2. No imbedded blanks are allowed in the command; however, commas are required to separate the arguments.

Table 8-2. Valid Editing Command Formats

Command	Function
LIST	Types out a brief description of the valid commands
COMAND,LU	Changes the command input device to LU
BUILD,LU	Reads the installation file from device LU and builds the skeleton file in the scratch area
LOAD,LU	Reads the skeleton file from device LU and transfers it to the scratch area
CATLOG,N1,N2	Lists records numbered N1 through N2 from the skeleton file
DELETE,N1,N2	Deletes records N1 through N2 from the skeleton file
INSERT,N,LU	Reads new skeleton records from LU and inserts them in the file immediately following record number N
DUMP,LU	Writes the skeleton file onto device LU
CHANGE,ILU1,LU2	Finds all *K records which specify LU1 as the input device and changes LU1 to LU2
EXIT	Exits from SKED and returns control to the job processor; can also be accomplished by responding to the NEXT statement via a carriage return

The tape motion commands have the same formats as those in DEBUG. Pseudo tape motion of the skeleton file is accomplished by specifying SK as the logical unit. These commands are given in table 8-3.

Table 8-3. Pseudo Tape Motion Commands for SKED

Command	Function
REW,LU	Rewinds LU
UNL,LU	Unloads LU
ADF,LU,N	Advances N files on LU
BSF,LU,N	Backspaces N files on LU
ADR,LU,N	Advances N records on LU
BSR,LU,N	Backspaces N records on LU
WEF,LU,N	Writes N file marks on LU

Additional SKED Commands

The use of SKED involves additional commands which are discussed in the following paragraphs.

COMAND COMMAND

If LU is a device other than the standard input comment device, the comments are output on the standard print output device.

BUILD COMMAND

When an end-of-file condition is detected, the user is asked the following question:

ANY MORE INPUT. ENTER LU

If there is more input, the operator is to type the logical unit number from which the information will be read, followed by a carriage return. If there is no more input, the operator should enter a carriage return. An *END record is appended as the last record of the skeleton file; the installation file must not have an *END record in it.

LOAD COMMAND

The LOAD command works the same as the BUILD command with the exception that an *END command may be present as the last record in a skeleton that is being loaded. If the *END record is present, however, it must be followed by an end-of-file record. Otherwise, these two records are optional.

CATLOG COMMAND

There are three forms of the CATLOG command (table 8-4).

Table 8-4. CATLOG Commands

Command	Function
CATLOG	Resequences and lists the entire skeleton file
CATLOG,N	Lists record number N of the file
CATLOG,N1,N2	Lists records numbered N1 through N2

The only to resequence the file, except for the DUMP command, is by the simple command CATLOG. Prior to resequencing, there will be gaps in the sequence numbers where records have been deleted, and any inserted record appears in proper position but without sequence numbers.

DELETE COMMAND

No record, which has been deleted or inserted since the file was last resequenced, may be referenced. A maximum of 500_{10} record deletions is allowed before the file must be resequenced. When the number of deletions goes over 500 on a certain command, the DELETE command is ignored, and the user receives a message asking him to resequence the file.

The last record in the file is the *END record, which may not be deleted. The command has the two forms in table 8-5.

Table 8-5. DELETE Commands

Command	Function
DELETE,N1	Deletes record number N1
DELETE,N1,N2	Deletes records N1 through N2, where N1 N2

INSERT COMMAND

No record may be referred to which has been deleted or inserted since the file was last resequenced. No insertions may be made following the last record.

DUMP COMMAND

The skeleton is dumped onto the specified device with an end-of-file mark written at the end. The file is automatically resequenced and listed after the dump is complete.

Error Conditions and Messages

If a device failure occurs, the appropriate standard device failure message is printed on the console. This condition can commonly occur with the device being used to build or load the skeleton file; e.g., the card reader has read all the cards without detecting an end-of-file mark. When the failure is not really an error, the operator should respond to the message ACTION with a CU. The CU has the same effect as an end-of-file. The program then continues. In this case, the user is asked if there is any more input.

The SKED error messages are described in appendix E.

LIBRARY BUILDER (LIBILD)

The library and installation file builder (LIBILD) provides the following capabilities:

- Merges input libraries of relocatable binary programs into a single output library, discarding duplicated programs
- Produces an installation file suitable for building a system via the initializer or LIBEDT
- Conversational control statements
- Batch control statements
- Absolutized file input and output to installation file
- Input and output devices fully selectable via logical unit designations
- Substantial recovery features
- Diagnostics in English
- Prompting messages and pauses at appropriate times to allow the operator to mount tapes, etc.

OPERATION PHASES

Operation phases are required or are optional as follows:

- Control statements — required
- Library input — required
- Library output — optional
- Definitions input — optional
- Skeleton input and installation output — optional

Control Statement

This statement is the operator's response to a query made by LIBILD. The response may be entered via the standard input comment device or via another device which is specified by the operator.

Input Library

The input library consists of a set of relocatable binary programs and subprograms which are terminated by an *END record or an end-of-file indicator or by an I/O error answered by a CU. An input library can have any number of system initializer and LIBEDT control statements; therefore, an existing installation tape can be used as a library.

Absolute files such as MACSKL can be a part of a library.

Output Library

The output library is produced from one or more input libraries. It is a set of relocatable binary programs and subprograms which are terminated by an *END record.

Duplicate Program

The duplicate program is a relocatable binary program or subprogram where the NAM block contains a name and identification equal to a previously input program or subprogram. In the case of absolute files, only the name is used as a basis of comparison.

Absolute File

The absolute file consists of a set of binary records where the first record is not a NAM block or an asterisk, preceded by a record of the form *N,XXXXXX,,,B, and terminated by a record having an asterisk (*) as the first character. If the macro skeleton, for example, is on an input library it must take the following form:

```
    *N,MACSKL,,,B
                        absolute
                        binary
                        records
    *(anything)
```

The name given to the set of absolute binary records is that of the name field of the *N record; in this case, MACSKL.

Skeleton

This is a set of ASCII records, each of which has an asterisk as the first character and all of which are terminated by an *END or end-of-file indicator or an I/O error answered by CU. These can be system initializer and LIBEDT control statements, LIBILD control statements, or anything else supported by COS The skeleton defines the logical sequence and content of information written to the installation file. Records recognized as LIBILD statements and their usage are discussed in the following paragraphs.

*B RECORD

The *B statement appears in the following form:

    *B 'A---A' 'BB---B'

The *B statement directs LIBILD to retrieve a relocatable binary program, subprogram, or absolute file. It also directs LIBILD to write the entire program on the installation file. The name of the program is specified by a one-to-six-character name enclosed by single quote marks; the program identification (if any) is specified by a one-to-48-character string enclosed by single quote marks. The ident field allows differentiation between programs that have the same name. Leaving the ident field entirely blank (not even quote marks) causes the first copy of several copies or the only copy of a program to be retrieved. The *B record for an absolute file should have a blank ident field.

An example of the *B record is:

    *B 'JOBENT'

Quote marks may begin anywhere after column 2; however, note that embedded blanks are significant.

*N RECORD

The *N record appears as follows:

    *N,AAAAAA,,,B

This statement is used by LIBEDT to place an absolute file in the program library. This record is mentioned here only to the extent that it signifies that the following records could be an absolute file. The name field (AAAAAA) begins in column 4, and embedded blanks are significant. This record is written to the installation file.

*WEF RECORD

This statement appears as:

    *WEF

The *WEF record directs LIBILD to write an end-of-file indication on the installation file.

*USE RECORD

The *USE statement is as follows:

    *USE A

This record is the counterpart to a 1700 assembler MACRO call and is used to *call out* or specify that the records grouped under a previous *DEF A record are to be inserted in the skeleton at this point. Nesting of the *USE record is permitted to a depth of six levels; for example, figure 8-25 is a six-level nesting.

```
*DEF A
*B 'PROGRM'                    Defines symbol A
*TER
*DEF B
*USE A — Insert A definition          Defines symbol B
*B 'PROGRA'
*TER
*DEF C
*USE B — Insert B definition          Defines symbol C
*TER
*DEF D
*USE A
*USE B                         Defines symbol D
*USE C
*TER
```

Figure 8-25. Six-Level Nesting of *USE Record

When a *USE record is specified, the *DEF and *TER of the corresponding symbol are not inserted in the skeleton.

*END RECORD

The statement format is as follows:

*END

This record signifies the end of the skeleton. It is written to the installation file.

* RECORD

The * record format is:

*(none of B,WEF,DEF,TER,or USE)

This is a record used by other programs in the system and has no meaning for LIBILD. The record is written to the installation file; for example:

*K,I5,P8

*LIBEDT

*T

*Z

*U

Definitions

The definitions consist of a set of ASCII records, each of which has an asterisk as the first character and all of which are terminated by an *END or end-of-file indication or an I/O error answered by CU. A definition group or set begins with a *DEF record, contains only valid skeleton records, and ends with an *END record. The definition records use the following general format:

*DEF A

This record directs LIBILD to set up an internal directory entry to be identified by the single character A, which must be unique for each *DEF record. Subsequent records are read until a *TER record is detected. Those records are then to be referred to as a group by the symbol A. This process is exactly analogous to a 1700 assembler MACRO definition. Every record in the group must begin with an asterisk; for example:

*DEF F

*B 'RTNSPC'

*B 'DEFFIL'

*B 'RELFIL'

*B 'RTVSEQ'

*B 'STOSEQ'

*TER

The ident field may contain any characters with the exception of single quotation marks which are used as delimiters.

Installation File

When this file is produced by LIBILD, it consists of the set of relocatable programs, subprograms, and absolute files specified by the *B records in the skeleton plus all other records in the skeleton that have no meaning to LIBILD. Usually the installation file is suitable input to the system initializer or LIBEDT.

LIBILD Operation

LIBILD operates according to the following steps:

1. LIBILD resides in the program library and is executed by the job processor in response to the following control statement:

   *LIBILD

2. The program always requests its first control statement from the standard comment input device as follows:

   CONTROL LU =

   This query is used for determining the logical unit from which the program will read subsequent control statements, such as a card reader. If the operator types only a carriage return, the queries and control statements remain on the standard comment device. If a logical unit number is entered, each query is printed on the standard list device prior to reading the control statement, which is also printed. A negative reply to any query is either a carriage return on the comment device or a card image having a blank in column 1, whichever is appropriate. If a card is used, the characters which follow the first blank are ignored except that the entire card is printed on the list device following the query message. All positive replies are followed by a carriage return or a blank.

3. The next statement is used for specifying the logical unit from which definitions will be read. The statement appears as:

   DEFS LU =

4. The next statement is used to specify the logical unit on which the installation file is to be written. The statement is:

   INSTALL LU =

5. The next statement specifies the output library logical unit as follows:

   NEWLIB LU =

6. Up to nine input libraries may now be specified. The first query of this type is as follows:

   LIB 01 LU =

   A logical unit number should be typed at this point. The next query is:

   LIB 02 LU =

   On this and subsequent replies, the negative reply signifies that there are no more library logical units. The number which follows the symbol LIB is just a library number counter.

7. The next query is produced when the user has finished specifying library logical units. The query appears as:

   SKELETON LU =

   Following this reply, the first library is read.

   During the library input phase, each program name and ident field is printed on the list device.

   Note

   The skeleton LU must be different from the install LU.

8. The following query appears after the first library and each subsequent library is input:

   LOAD LIBRARY INPUT XX ON LU YY.
   CR WHEN READY.

   In the query, the symbol XX is the library count, and the symbol YY is the logical unit number. All such numbers in this message occur in the same order as they were defined during the control statement phase. When this message is printed, the operator may change tapes, reload the card reader, etc. When the desired library is finally on the device, pressing the carriage return key causes that library to be input.

9. The following message appears when an output library option has been selected:

   LOAD OUTPUT LIBRARY.
   CR WHEN READY.

   When the device is ready, pressing the carriage return key causes the programs to be alphabetized by name. Then those relocatable binary programs are written to the device.

10. During the definitions input phase, the following message allows the operator time to load and ready the device:

    LOAD DEFS, CR WHEN READY.

    Pressing the carriage return key causes the definitions to be input.

11. During the definitions input phase, each definition record is printed on the list device.

12. During the skeleton input phase, the following message always appears on the comment device:

    LOAD SKEL/INSTALL, CR WHEN READY.

    As in the case of libraries, this allows the operator time to load and ready both the skeleton input device and the installation file output device. Pressing the carriage return causes skeleton record to be output concurrently.

13. During the skeleton input and installation file output phase, skeleton records are printed on the list device normally under one of two conditions. If the record has no meaning for LIBILD, it is written to the installation file. If the record has meaning to LIBILD, the appropriate substitution or action is performed. Refer to the discussion of skeleton.

14. The following message appears whenever a *WEF record is encountered.

```
END FILE MARK WRITTEN
```

15. At the completion of skeleton input phase, the following message is output on the comment device:

```
LIBRARY BUILD COMPLETE
TYPE *Z TO TERMINATE OR
TYPE *C TO CONTINUE WITH CURRENT
SKELETON AND/OR OUTPUT LIBRARY
LU'S
```

If the operator types *Z followed by carriage return, LIBILD exits to the job processor. If he types *C, the routine recycles to the point at which the skeleton and installation file devices are loaded and set ready. In other words, a different skeleton and installation file may be loaded but only on the same devices used by the first skeleton and installation file. Also, the same set of library programs is retained. This feature could be used after skeleton errors are detected and corrected.

There is no program limit to the number of times this feature may be exercised.

Recovery From Errors

During the library input phase, if a diagnostic message (NAM record not first record of deck) is output and the operator thinks there may have been a transient hardware problem (e.g., tape-positioning or card-feed problems), he may manually backspace the library and try again by typing 3 in response to the message. Backspacing the card reader is straight-forward; simply refeed the card. Backspacing the magnetic tape presents problems, however. If the operator has the time, he may rewind the tape and reprocess it, thereby obtaining diagnostic messages for every program which was successfully loaded on the previous pass of that tape. This technique is entirely valid; however, if the library input is very large, that technique is time-consuming. It would require a long time to reject 900 programs.

A slightly more complicated but faster process is as follows:

1. Manually backspace the tape a few feet. The tape is now positioned in the middle of a record, at the beginning of a record that is not a NAM block or a record beginning with an asterisk, or at a NAM or asterisk record.

2. Type 3 followed by carriage return, and a record will be read.

3. If the tape position was at an asterisk or NAM record, processing of the input library proceeds with only a few reject messages.

4. If the tape was positioned elsewhere, the following message is output again:

```
NAM RECORD NOT 1ST
```

5. Now, typing 3 will cause LIBILD to slew over records until a NAM record or a record beginning with an asterisk is encountered. Processing this input library proceeds with only a few reject messages.

6. During the skeleton input phase, if skeleton format errors are detected or certain skeleton records are out of order or missing, the operator can correct or rearrange the skeleton deck and try again after the following message is output:

```
LIBRARY BUILD COMPLETE
TYPE *Z TO TERMINATE OR
TYPE *C TO CONTINUE WITH CURRENT
  SKELETON AND/OR OUTPUT LIBRARY
  LU'S
```

7. During the control statement phase, if the operator accidentally specified more input libraries than he really had and all the libraries have been input (in lieu of starting over), he can complete steps 8 through 10.

8. When the following message occurs for the nonexistent library, make sure the device is not ready and type carriage return. The operating system outputs a failure message for the device.

```
LOAD LIBRARY INPUT xx ØN LU yy.
CR WHEN READY.
```

9. Type CU.

10. LIBILD defaults an *END at this point and proceeds as though a library had been input. Do steps 8 to 10 for each nonexistent input library.

Restrictions and Limitations

The following limitations must be adhered to for LIBILD:

- Maximum number of unique program/ident combinations — 1024
- Maximum number of input libraries — 9
- Maximum number of definition groups or sets — 20
- Maximum number of recursive levels in a definition — 6
- Maximum number of times a given program name/ident can appear on a *B record — no limit
- Maximum number of times a given program name can have a different ident — 1024
- Maximum number of times the same program name/ident will be rejected if encountered more than once — no limit
- Maximum number of records in a skeleton — no limit
- Maximum number of skeletons input — no limit

Changing any of these limits requires a major program modification.

Special Notes

The name and identification information of a binary program is obtained during the BUILD operation from the NAM record of that program. It may be desired that certain information which appears in the identification field of the NAM card (columns 25 to 72) is to not be included in the identification field of the skeleton record that is generated because of possible problems that may occur later when the library builder program, LIBILD, is being run. When a program on one input library is being replaced with a program on another input library, for example, LIBILD only recognizes which program to replace by the fact that the program names and identification fields in their respective NAM records are identical. If any information is different, such as the date, LIBILD treats the two programs as though they were unique. Both programs, therefore, are included in the installation file.

GENERAL CASE

Any information on the NAM card that is not to be included in the identification field of the skeleton record should follow the information that is to be included by at least four blank columns.

SPECIAL CASE

If none of the information in the identification field is to be included, at least eight blank columns must be left at the beginning of the identification field; that is, the identification information on the card should start in or after column 33 on the NAM card.

PAULA PROGRAM

The program PAULA allows an install-type tape (relocatable binary format) to be updated. This type of tape includes the system install tape and the program library install tape. On release install tapes, these are combined on one tape; however, the user may wish to keep them on separate tapes for convenience.

PAULA Capabilities

PAULA has the following general capabilities:

- Delete relocatable binary modules
- Insert relocatable binary modules
- Delete commands imbedded in the tape
- Insert commands into the tape

PAULA Operation

PAULA operates in two passes. On the first pass, an install tape image of the desired format is created on disk. On the second pass, this image is copied to an output tape.

On the first pass, input is from one or more tapes (usually the current install tape) with PAULA commands entered from either the system console or from a card reader.

In all the cases in table 8-6, X is either the six-character name of a relocatable binary module or a command that may be on the install tape. The install tape command is only compared up to the first comma.

Table 8-6. PAULA Commands

Command	Comments
L,X	Lists all module names and commands on the system list device; input is from the input tape as defined in command C,YZ
S,X	Skips all modules and commands through X; only X is listed
T,X	Transfers all modules and commands through X from the input tape to disk
E,X	Transfers modules and commands from disk to the output tape as defined in command C,YZ; transfer is complete when either X has been transferred or when end of file is reached, whichever comes first
D,Z	Select the start-sector for an E,X command; Z = absolute sector number (hexadecimal); if this command is not used, an E,X command begins with the file's first sector
M,YZ	Tape motion request Where: Y–logical unit number Z = 1–backspace 2–write file mark 3–rewind 4–rewind and unload 5–skip one file forward 6–skip one file backward
C,YZ	Change logical unit numbers Y = I–input unit P–output unit Z–logical unit number
I,	Inserts one statement onto disk; statement that follows this command is inserted
A,	Returns to job processor
R,	Repeats E
W,	Replaces one program The device which contains the old installation system is specified with: *K,PX The device which contains the new program is specified by: *K,IX This routine allows chaining of program names, which are separated through commas. Other programs as specified are ignored without comment. If the input name does not exist in the system, the tape searches until ENDTAP (last program name in system) and rewinds. An error message is given, and the tape skips to the last correct program.
V,Z	Changes PAULA command input device Where: Z–logical unit
U,	Restores PAULA command input device to system comments input

PAULA Example

The example in table 8-7 shows how a special test may be inserted in the install tape. The special test module being added is named ST021 and has an entry point of SPT21. Use the current install tape as an example.

Table 8-7. Example of PAULA Program

Command	Comments
C,I6	Sets input to be from logical unit 6, which has the current install tape
T,ST005	Transfers all modules and commands to disk through ST005, which is the last special test currently on the install tape
C,I7	Sets input to be from logical unit 7, which has a tape containing the relocatable binary for module ST021
T,ST021	Transfers the new special test module to disk
C,I6	Sets input to be from logical unit 6, which still has the current install tape
T,*U	Transfers the rest of the install tape to disk
M,64	Rewinds and unloads logical unit 6; mounts a scratch tape on 6 to contain the new install
C,P6	Sets output logical unit to 6
E,*U	Transfers disk image to output tape; PAULA will output: OUTPUT UNIT IS 6 TYPE IN OK OR PX X = NEW LU NUMBER Reply to this message with: OK
M,62	Writes file mark on output tape
A,	Returns to job processor

In table 8-7, the PAULA command input is from the system console. An alternative method is to punch the commands on cards and place them in the card reader. Then the only PAULA command entered through the system console is V,5 which tells PAULA to get its commands from logical unit 5 (which is the card reader). Any errors cause PAULA to return control to the system console. Control can also be returned explicitly by using the command U,.

*BOOT COMMAND

The *BOOT command loads a magnetic tape (seven- or nine-track) bootstrap into core, starting at location 0 to read one file from physical tape unit 0. The file is loaded, starting at location $4000. BOOT is called in under the job processor by:

```
*BOOT
```

In addition, memory is zeroed out, interrupts are disabled, and another routine is loaded at $100 with a pointer to $100 at location $FE. This other routine handles three instances:

- Q = 0—zeros core from approximately $250 to end of core; prints message to mount disk and/or tape; gives option of skipping files on tape; allows teletypewriter console control of reading in absolute files from tape

- Q < 0—tape error found during read or file advance; status is printed on teletypewriter; routine jumps to Q = 0 option

- Q > 0—file has loaded correctly and file mark has been found; message printed to that effect; starting and ending load locations are printed; gives option to execute the program or START OVER (start over at Q = 0 option)

The bootstraps loaded by BOOT have been modified slightly to jump to this other routine in case of error or for normal completion. The bootstrap integrity is maintained such that it may be manually run by pushing MASTER CLEAR and GO.

Complete instructions are given by the routine on execution.

NOTE

> No checking is being done to test the size of the file being loaded; i.e., it is possible to wrap around memory and/or store data randomly throughout memory after location $7FFF if a wrong tape has been mounted and loaded.
>
> Location $1 contains the starting address for the load; it may be changed. After a change, set Q = 0 (or MASTER CLEAR) and then P to $100 for normal operation (or P to $0 for manual operation). Suggestion for changes — do not load before $300.

*PIC COMMAND

The *PIC command allows the user to pick COSY update decks from an input file (tape) by specifying the appropriate deck name to pick or to copy through.

The output or picked decks go onto the punch file. At the same time, a listing of these decks is output to the standard list device.

The input, punch, and list units may be specified after entering PIC. These unit assignments then remain local to PIC. The unit values may also be changed by using the *K system command before entering PIC.

Since the program COSY expects update checks to end with an END/ card, the user should add this card to the end of the punch file before rewinding it.

The program PIC gives instructions on how to use all the options available to the user. It is called in under the job processor by:

```
*PIC
```

CYFT COMMAND

The COSY format program is used to insert the proper COSY control cards in assembly language programs so that the generated output can be input to build COSY source programs. The input source is from the standard input device (cards, paper tape, magnetic tape). If input is from magnetic tape, the last deck of the input source must have an end-of-file mark.

CYFT Execution

The procedure to execute this program is:

```
*JOB
J
*K,Ilu,Plu,Llu
J
*CYFT
```

Parameters

The parameters for CYFT are as follows:

I—assigns the logical unit for the source program input

P—assigns where COSY control cards and source programs are to be punched (written)

Example:

```
xxxxxx  DCK/       I,C  (CYFT generated)
xxxxxx  HOL/            (CYFT generated)
        NAM xxxxxx

        END
        END/            (CYFT generated)
```

L—assigns the logical unit for a listing of COSY DCK/ cards to be listed

Example:

```
xxxxxx  DCK/       I,C
xxxxxx  DCK/       I,C
```

If the logical unit fails on completion of input, type in CU. This generates the final END/ for COSY.

SUP COMMAND

Refer to appendix F for the SUP command.

ASSEM COMMAND

Refer to appendix G for the ASSEM command.

SECTION 9

ENGINEERING FILE

Engineering File

INTRODUCTION

The engineering file is provided to preserve driver error information for system maintenance.

The engineering file is divided into three sections:

- Failure data formatting and collection
- Failure storage on mass memory
- Failure listing

DEVICE FAILURE HANDLING

When an I/O driver determines that an error condition has occurred, it reports the error to the error-logging routine in EFDATA. The entry is:

RTJ+ LOG

Where the following registers are defined:

- Q register—figure 9-1

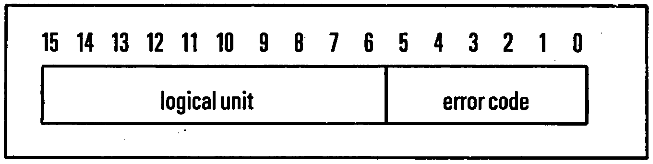


Figure 9-1. Q Register

- I register—driver physical device table (PDT) address

To this data, the current data, time, and hardware status (PDT word 12) are added relevant to the failure. If the failure is on a non-mass-memory device, the data is saved in a push-down pointer table (capacity of five 5-word entries). Each entry is of the form shown in figure 9-2.

When the first failure is placed in the table, the mass memory failure storage module is scheduled to move the failure data to mass memory. Return is made to the driver caller. The LOG sequence is reentrant.

If the failure is on a mass memory device, the failure is saved in a 10-entry push-down table. Each five-word entry of the table is of the same form as figure 9-2. This failure data is saved in core on the premise that mass memory is not reliable because of the failure.

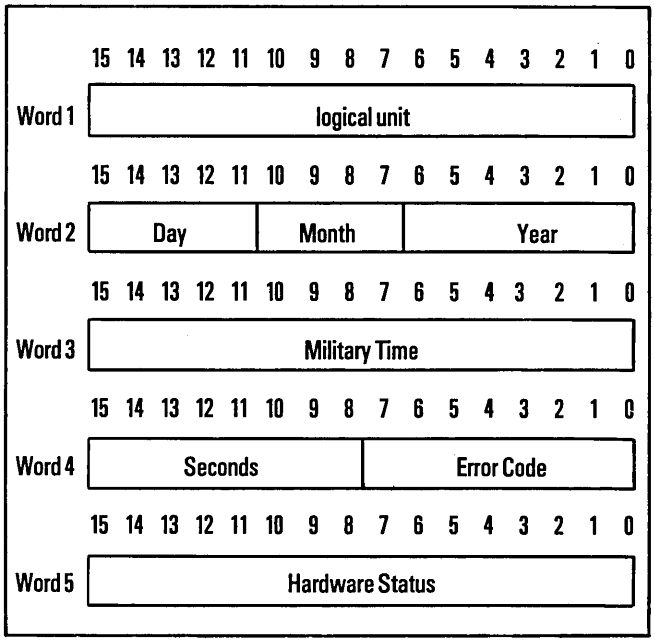


Figure 9-2. Push-down Pointer Table Example

For mass memory failures, EFDATA also logs the failure on the system comment output device with the following message:

MM ERR XX LU=YY T-HHMM:SS S=ZZZZ

Where:

XX	— error code
YY	— logical unit
HH	— hour
MM	— minute
SS	— second
ZZZZ	— hardware status

If the failure occurred during a system directory program read, the allocated space is released. The mass memory diagnostic section of EFDATA operates on the mass memory failure table at a low priority after return to the driver.

DEVICE FAILURE STORAGE

The system initializer defines a mass memory area of 99 sectors (preset to 0) for the engineering file. The address of this area is found in word 19 of the extended core table. One sector is allocated for each possible system logical unit. For each logical unit, 24 failures are saved as four-word entries in a push-down/fall-off table. The first entry is the most recent failure.

The EFDATA program schedules the EFSTOR program from the system library to log failures for non-mass-memory devices. The EFSTOR program takes data from the five-entry failure table and moves it to mass memory in the engineering file area for the failed logical unit. Each table entry is composed of words 2 to 5 of the core-resident table. The five-entry holding table is examined, and errors are processed until all entries are zero.

The core-resident mass memory device failure table is examined for new entries, and all new entries are moved to the engineering file. The philosophy of this data relocation at this time is that the system is operating following a mass memory error if further system I/O errors can be logged.

If the five-entry holding table overflows, a diagnostic message is printed and the system must be restarted with an autoload.

Device Failure Listing

The device failure listing program EFLIST, is entered via the following manual interrupt mnemonic codes:

- EF—lists all engineering file data for system logical units
- EFLU—lists engineering file data for a specified logical unit
- EFMM—lists data from the core-resident mass memory failure table for all errors in the table

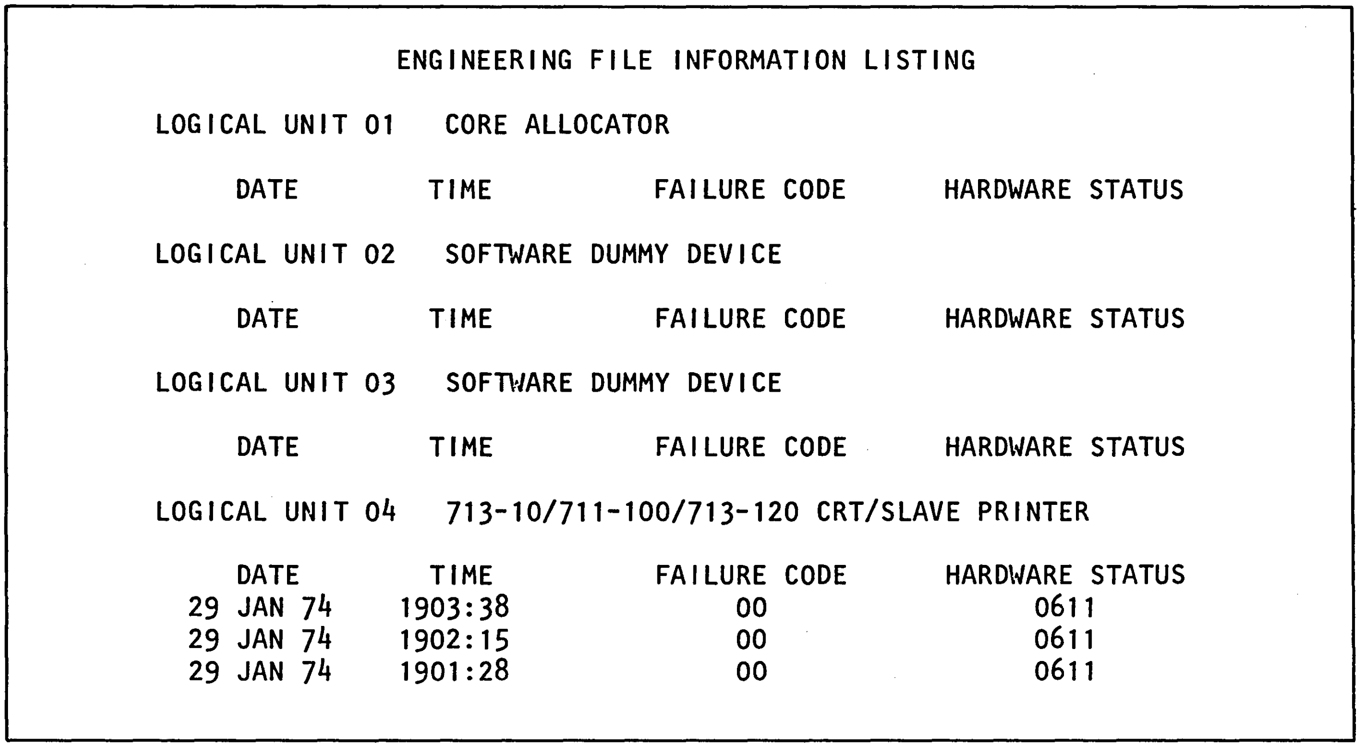
The EF entry produces a listing of the format of figure 9-3.

The EFMM entry produces a listing of the same form; however, it contains only failures from the core-resident mass memory failure table.

The EFLU entry requests the logical unit number as follows:

ENTER LOGICAL UNIT (XX)

The data for the specified logical unit (XX) is printed in the same form as for EF.



```
                 ENGINEERING FILE INFORMATION LISTING

     LOGICAL UNIT 01   CORE ALLOCATOR

          DATE        TIME          FAILURE CODE      HARDWARE STATUS

     LOGICAL UNIT 02   SOFTWARE DUMMY DEVICE

          DATE        TIME          FAILURE CODE      HARDWARE STATUS

     LOGICAL UNIT 03   SOFTWARE DUMMY DEVICE

          DATE        TIME          FAILURE CODE      HARDWARE STATUS

     LOGICAL UNIT 04   713-10/711-100/713-120 CRT/SLAVE PRINTER

          DATE        TIME          FAILURE CODE      HARDWARE STATUS
        29 JAN 74    1903:38             00                 0611
        29 JAN 74    1902:15             00                 0611
        29 JAN 74    1901:28             00                 0611
```

Figure 9-3. Engineering File Listing Format

APPENDIX A

SYSTEM INSTALLATION

System Installation

GENERAL PROCEDURE

Installing the CYBERDATA operating system from magnetic tape to disk involves the following major steps:

1. Load the seven- or nine-track bootstrap.
2. Use the bootstrap to load the disk initializer from tape.
3. Execute the disk initializer to write address tags and data on the disk.
4. Use the routine to read the system initializer from tape.
5. Execute the system initializer to install the operating system.
6. Autoload the new operating system and use DEBUG to put in any necessary patches.
7. Use LIBEDT to set system priorities and to load the program library from tape.

DETAILED PROCEDURE

These general steps are explained in detail in the following paragraphs.

Loading the Bootstrap

The seven- and nine-track bootstraps are on the program library of the CYBERDATA disks under the name BOOT. If no system disk is available, the appropriate bootstrap must be loaded manually at the computer console. Refer to the discussion on entering the magnetic tape bootstrap on page A-4. After entering, continue at the step to load the disk initializer.

LOADING BOOTSTRAP FROM A CYBERDATA DISK

To load the bootstrap from a CYBERDATA disk, enter the job processor, and call the bootstrap loader by typing:

```
*BOOT
```

Instructions are printed on the teletype. After instructions are typed, turn off the program protect. Type a carriage return. The selected bootstrap is then loaded, starting at location 0.

Load the Disk Initializer

Be sure that the tape is on-line on unit 0. Press MASTER CLEAR and then GO to read the file from tape into core, starting at location 4000.

Execute the Disk Initializer

Type E to execute, and then follow the instructions given on the teletype.

NOTE

> Be sure the disk to be initialized is the one that is mounted.

Load the System Initializer

When the disk initializer completes, it types a message. Type R to read the next file from tape. This loads the system initializer.

Execute the System Initializer

Type E to execute, and then follow the instructions given on the teletype.

To input system parameters:

1. Set the initializer list device (default is teletypewriter):

   *C,7,0201 — for printer

   *C,8,0000 — for no list

2. Set core memory size if other than 28K (28K is default):

   *S,MSIZV4,XXXX

Where:

32K — 7FFF
36K — 8FFF
40K — 9FFF
44K — AFFF
48K — BFFF
52K — CFFF
56K — DFFF
60K — EFFF
64K — FFFF

3. Set END0V4 if other than 28K and/or if it includes self-scan 32-character station:

   *S,END0V4,XXXX

Refer to table A-1 for hexadecimal values used with END0V4.

Table A-1. END0V4 hexadecimal Values for Install

Core Size	32-Character Only	480-Character Only	32- and 480-Character
28K	3A8F	36AF	32FF
32K	4A8F	46AF	42FF
36K	5A8F	56AF	52FF
40K	6A8F	66AF	62FF
44K	7A8F	76AF	72FF
48K	7FFF	7FFF	7FFF
52K	7FFF	7FFF	7FFF
56K	7FFF	7FFF	7FFF
60K	7FFF	7FFF	7FFF
64K	7FFF	7FFF	7FFF

4. Set BGNMON if other than 28K and/or if it includes the self-scan 32-character station:

   *S,BGNMON,XXXX

   Refer to table A-2 for hexadecimal values used with BGNMON.

Table A-2. BGNMON Hexadecimal Values for Install

Core Size	32-Character Only	480-Character Only	32- and 480-Character
28K	3A90	36B0	3300
32K	4A90	46B0	4300
36K	5A90	56B0	5300
40K	6A90	66B0	6300
44K	7A90	76B0	7300
48K	8A90	86B0	8300
52K	9A90	96B0	9300
56K	AA90	A6B0	A300
60K	BA90	B6B0	B300
64K	CA90	C6B0	C300

NOTE

Values may be changed if special test binaries are added.

END0V4 must be smaller than BGNMON, and not larger than $7FFF.

Examples:

32K system (480 key stations only)

*S,MSIZV4,7FFF
*S,END0V4,46AF
*S,BGNMON,46B0

44K system (480 key stations and 32 key stations + 300-hexadecimal word special test)

*S,MSIZV4,AFFF
*S,END0V4,6FFF
*S,BGNMON,7000

5. Set the ending sector for CYBERDATA data files (default — end of disk for dual-density disk drives):

   *S,SIESEC,XXXX

   Where:

   XXXX — 5B88 for single-density disk
   — B780 for double-density disk

   This uses all the remaining disk area on fixed and cartridge disks.

To specify only the removable part of the disk, the value should be cut in half. Thus, for a double-density disk but only the removable portion, XXXX = 5B88.

6. Initiate input from the install tape:

   *V

   System install runs to completion and informs the user that he may autoload.

   NOTE

   Do not rewind install tape.

Autoload New System and Enter Patches

To autoload a new system and to enter a patch:

1. Set the system key to N (normal) mode.
2. Reload the system.
3. Enter the time and the date.
4. Load the patches.

   Refer to the discussion on system modification and update on page A-4 for more information on DEBUG.

   To enter a patch, press MANUAL INTERRUPT. After the system responds with MI, type DB. The system responds with DEBUG IN.

   At this time, patches should be entered. The patches are of the form:

   LHM,,MMMM,N/NNNN,NNNN,...NNNN

   or

   LHC,MMMM,N/NNNN,NNNN,...NNN

   Where:

   MMMM is a module name

Before entering the patch, replace the module name with the value that appears beside it in the installation listing. After replacing the module name in the patch, enter it on the console. DEBUG prints old and new contents and waits for verification from the user. If the command appears correctly, type OK, then press RETURN. If it is incorrect, do not type OK, and then press RETURN. The system then rejects the command. After all patches have been entered, type OFF to leave the DEBUG routine.

System Library Priorities and Loading Program Library

To set system library priorities and to load the program library:

1. Press MANUAL INTERRUPT and type *BATCH. Control is passed to the tape, and library installation is initiated. Near completion of this procedure, the teletype types the following message:

   TYPE *E WHEN E APPEARS

   When system types E, operator types:

   *E $\boxed{\text{CR}}$

   When completed, a message informs the operator to reautoload. The system build is now complete.

SETTING CYBERDATA PARAMETER

The terminal configuration is set as follows:

1. Autoload the system, and enter the time and the date.
2. Press MANUAL INTERRUPT, and type VLTP. The system responds with VLTP IN.
3. Define the terminal configuration. Refer to the CYBERDATA System Reference manual on Supervisory Command and Control: authorizing an operator to enter Supervisory Commands and Cancelling an Authorization.
4. Type *Z to terminate.

To set the system for automatic CYBERDATA load:

1. Press MANUAL INTERRUPT, and type VLOS. The system responds with VLOS IN.
2. Type STD. The system responds with VLOS OUT. For more information concerning this parameter, see the discussion on disk and memory allocation.
3. Reautoload the system.

ENTERING MAGNETIC TAPE BOOTSTRAP

The following procedure is used to enter the magnetic tape bootstrap into the data entry processor:

1. If power is not on, press START. Allow the autoload cycle to finish before going to step 2.
2. Turn the autoload console key to M (horizontal position).

NOTE

> Steps 3 to 11 are performed at the data-entry console.

3. Press MASTER CLEAR.
4. Press X.
5. Set ENTER/SWEEP switch to ENTER (up).
6. Set INSTRUCTION/CYCLE switch to INSTRUCTION (up).
7. Enter the first/next word of the seven- or nine-track bootstrap (figures A-1 and A-2) into the console register.
8. Press GO.
9. Press CLEAR.

NOTE

> Do **not** press MASTER CLEAR.

10. Repeat steps 7, 8, and 9 until all words have been entered.
11. Return the ENTER switch to the center position.
12. Return the INSTRUCTION switch to the center position.

The bootstrap is in core. To check the bootstrap:

1. Press MASTER CLEAR.
2. Press X.
3. Set ENTER/SWEEP switch to SWEEP (down).
4. Set INSTRUCTION/CYCLE switch to INSTRUCTION (up).
5. Press GO.
6. Compare the display with the bootstrap code.
7. Repeat steps 5 and 6 until all entries have been checked.
8. Return the ENTER/SWEEP switch to the center position.
9. Return the INSTRUCTION/CYCLE switch to the center position.

NOTE

> To execute the bootstrap, ensure that the magnetic tape unit is mounted and ready. Push MASTER CLEAR and GO.

SYSTEM MODIFICATIONS AND UPDATES

The system is modified via the commands given in the following paragraphs.

Patches

Simple system changes can sometimes be made with patches to the absolutized system modules on the disk. This method does not require system reinstallation. Detailed information on the use of patches can be found in chapter 6. The purpose of this appendix is to describe patches as they may be received to update the system.

There are two patch commands used to update a system. One is the LHC command, which is used to patch core-resident system modules. The other is the LHM command, which is used to patch mass-resident system modules.

The format of the released system patch is as follows:

$$\left\{\begin{array}{c}\text{LHC}\\ \text{LHM}\end{array}\right\}\text{,aaaaaa(1),nn(2)/nnnn(3),...,nnnn(7)}$$

Where:

(1)—systems module name

(2)—bias of patch into the module

(3) to (7)—patch in hexadecimal

Before using the patch, the module name in the patch must be replaced with the value that it represents.

LHC COMMAND

For the LHC command, the module name must be replaced with the memory location where that module begins. This value can be found in the install listing. The first part of the listing shows the core-resident modules, and the second part of the listing shows the mass-resident modules. The beginning address of each core-resident module is shown immediately to the right of the module name in the listing.

LHM COMMAND

For the LHM command, the module name must be replaced with the beginning disk sector number for the module. This value can be found in the install listing immediately to the right of the module name.

Entering the Command

Initiate the debug module when the job processor is not active. An example of entering a patch, with all the operator responses underlined, is as follows:

- Operator: Press MANUAL INTERRUPT
- TTY: MI
- Operator: DB CR
- TTY: DEBUG IN
- Operator: LHC,47EF,110/B00 CR
- TTY: VERIFY
- Operator: OK CR (if patch is acceptable)

  NO CR (if patch is not correct)
- Additional patches may be entered here.

The DEBUG module is terminated by the following:

- Operator: OFF CR
- TTY: DEBUG OUT

REASSEMBLIES OF MASS-RESIDENT MODULES

CYBERDATA mass-resident modules (with the exception of VLOSOP, VLSTRT, VLTYPE, VLBGOP, and VLSTTS) can be replaced in the systems through the use of the CYBERDATA supervisor command LPR. Changing modules in this manner does not require reinstallation of the system. Since each module must be handled individually, this method is not appropriate for replacing a large number of modules.

The LPR command requires that the module, which is to be loaded, must be in an absolutized form in the system scratch area on disk. This can be accomplished by assembling the module under the job processor and then absolutizing the module onto the scratch disk by using the LIBEDT command *P. Refer to chapter 7 for details.

```
                BOOT9  EQU  BOOT9(*)
P000  C810      START9 LDA* N44C
P001  E80E             LDQ* N382
P002  03FE             OUT  -1                 CONNECT TO TAPE UNIT
P003  0D01             INQ  1
P004  0A11             ENA   FWAM1-BOOT9       ADDRESS OF FWA-1 OF BUFFER
P005  03FE             OUT -1
P006  0DFD             INQ  -2
P007  C807             LDA* N100               READ MOTION
P008  03FE             OUT  -1
P009  0A00      STATUS ENA  0
P00A  02FE             INP  -1                 INPUT STATUS
P00B  0FCB             ALS  11                 EOP (BIT 4) TO BIT 15
P00C  0135             SAM  BUFF               IS IT EOP-
P00D  18FB             JMP* STATUS             NO, WAIT.
P00E  0100      N100   NUM  $100               READ MOTION
P00F  0382      N382   NUM  $382               EQUIPMENT CODE
P010  044C      N44C   NUM  $44C               UNIT SELECT, 800 BPI, ASSY/D-ASSY, BINARY
P011  0072      FWAM1  ADC  LWA+1              FWA-1 OF BUFFER - CONTAINS LWA+1 OF BUFFER
                BUFF   EQU  RUFF(*)            BUFFER START

      0071             EQU  LWA(BUFF+95-BOOT9) BUFFER END
```

Figure A-1. Printout of Nine-Track Bootstrap

```
                            EQU   BOOT7(*)
P0000 C831                  LDA*  USEL7
P0001 E832                  LDQ*  EQUIP7
P0002 03FE                  OUT   -1                  SELECT UNIT 0, 800 BPI, BINARY
P0003 0DFE                  INQ   -1
P0004 C82E        LOOPM7    LDA*  RMOTN7
P0005 03FE                  OUT   -1                  START READ MOTION
P0006 0DFE                  INQ   -1
P0007 0844                  CLR   A
P0008 0223        LOOP7     INP   REJ1-*              INPUT FRAME (6 BITS)
P0009 581C                  RTJ*  ROUT1
P000A 0FC4                  ALS   4
P000B 7C29                  SPA*  (HOLD7)
P000C 02FE                  INP   -1                  INPUT FRAME (6 BITS)
P000D 0821                  TRA   M                   SAVE IT
P000E 0F42                  ARS   2
P000F BC25                  EOR*  (HOLD7)
P0010 7C24                  SPA*  (HOLD7)             1ST WORD 6,6,4
P0011 D823                  RAO*  HOLD7
P0012 0A03                  ENA   3
P0013 08AC                  LAM   A                   RECOVER LAST 2 BITS (FOR 15,14)
P0014 5811                  RTJ*  ROUT1
P0015 5810                  RTJ*  ROUT1
P0016 0FC2                  ALS   2
P0017 7C1D                  SPA*  (HOLD7)
P0018 02FE                  INP   -1                  INPUT FRAME (6 BITS)
P0019 0821                  TRA   M                   SAVE IT
P001A 0F44                  ARS   4                   GET FIRST 2 BITS (FOR 1,0)
P001B BC19                  EOR*  (HOLD7)
P001C 7C18                  SPA*  (HOLD7)             2ND WORD 2,6,6,2
P001D D817                  RAO*  HOLD7
P001E 0A0F                  ENA   $F
P001F 08AC                  LAM   A                   GET LAST 4 BITS (FOR 15-12)
P0020 5805                  RTJ*  ROUT1
P0021 5804                  RTJ*  ROUT1
P0022 7C12                  SPA*  (HOLD7)             3RD WORD 4,6,6
P0023 D811                  RAO*  HOLD7
P0024 18E2                  JMP*  LOOP7-1             RETURN FOR NEXT 8 FRAMES
P0025 0B00        ROUT1     NOP   0
P0026 0FC6                  ALS   6
P0027 7C0D                  SPA*  (HOLD7)
P0028 02FE        LOOP1     INP   -1                  INPUT FRAME
P0029 BC0B                  EOR*  (HOLD7)
P002A 1CFA                  JMP*  (ROUT1)
P002B 0B00        REJ1      NOP
P002C 0D01                  INQ   1
P002D 02FE                  INP   -1                  GET DIRECTOR STATUS 1
P002E 0FCB                  ALS   11                  EOP TO BIT 15
P002F 0135                  SAM   BUFF7               END OF OPERATION?
P0030 18D5                  JMP*  LOOP7-2             NOT EOP


P0031 040C        USEL7     NUM   $40C                UNIT 0, 800 BPI, BINARY
P0032 0100        RMOTN7    NUM   $100                READ MOTION
P0033 0382        EQUIP7    NUM   $382                EQUIP NO. + D2
P0034 0035        HOLD7     ADC   BUFF7-BOOT7         POINTER TO BUFFER AREA


                            EQU   BUFF7(*)            BUFFER START
```

Figure A-2. Printout of Seven-Track Bootstrap

REASSEMBLY OF CORE-RESIDENT OR MASS-RESIDENT MODULES

The following method requires reinstallation of the system. This method can also be used for mass-resident modules.

All modules that are to be changed must be assembled such that their relocatable binarys appear serially on a tape in the same sequence that they appear on the install tape.

Through the use of program PAULA or LIBILD, these modules can be inserted into the current install tape. After updating the install tape, the system must be reinstalled.

INSTALLATION TAPE MODIFICATION

Modification of the installation tape, other than straight replacement of system modules, should only be attempted by analysts who have a thorough knowledge of both CYBERDATA and the operating system.

APPENDIX B

SPECIAL AND OWN-CODE TESTS

Special and Own-Code Tests

GENERAL

Special tests and own-code tests are user-written subroutines which enable nonstandard tests to be performed. The only exceptions are the four check digit verification (CDV) tests, which are provided as standard special tests. Four different CDV tests are available.

SPECIAL TESTS

Special tests are integrated into the CYBERDATA System at system generation time and reside permanently in computer memory. They are linked as subroutines; i.e., a return jump is made to the special test entry point (SPTxx; where xx is the test number). They are, therefore, less time-dependent than the own-code tests. A special test may not perform input/output (I/O) operations, but it may use any CYBERDATA System routine to access data. The CYBERDATA System can accept a maximum of 99 special tests, but only one special test can be specified per field.

Special test numbers 1 through 9 are reserved for check-digit verification. Numbers 1 through 4 are currently used; therefore, numbers 5 through 9 are user check-digit verification identifiers.

Special test numbers 10 through 20 are reserved for future CDC-developed standard tests which will be available to all users.

Special test numbers 21 through 50 have pointers assigned in the special test jump table, which are then available for customer use. Test numbers 51 through 99 must be linked by pointers.

CDV tests are standard with the system. These tests check the accuracy of a base number which has been assigned a check digit. They may be performed on pure numeric fields comprising up to 15 digits as data is entered. The system supervisor may modify the modules and weights; however, the method of calculation remains unchanged. Modules may be in the range 1 to 15, and weights may be in the range 0 to 15.

OWN-CODE TESTS

Own codes are disk-resident subroutines. The CYBERDATA System provides for two types of own-code tests: END OF FIELD (EOF) and end of batch (EOB). They may perform I/O operations by using standard I/O routines. The maximum subroutine length is 2K words. The maximum number of own codes in the system is established by an installation parameter and may not exceed 127.†

The EOF own code is specified in the field format information for each field which uses the own code. More than one own code may be called by a given format, but only one may be called per field.

The EOB own codes are specified on a job basis and are called at the end of batch in Entry and Verify modes. These routines may have access to all the standard functions of the CYBERDATA System. Own codes must be written in run-anywhere code. As a disk-resident program, the own code must begin with a standard header as shown in figure B-1.

```
OCHDR   VFD N1/1,N1/1,N3/0,X11/ENDOC-*
        VFD X10/TIME,N6/1
        NUM 0,0
        ADS START-OCHDR
```

Figure B-1. Own Code Standard Header

In figure B-1, the words are defined as follows:

- OCHDR VFD N1/1,N1/1,N3/0,X11/ENDOC-* (Word 1)

  Bit 15—I/O status

  1—program may not be moved
  0—program may be moved

  This bit is set to 1 on the disk and is set/unset at the discretion of the user (see SETMOV). The bit must be set while input/output is being performed into a buffer in the program. It must be set if the program calls a subroutine using disk input/output.

  The bit must be cleared by the last user exiting from the subroutine.

  Bit 14

  0—area free if user count is zero

  1—area not free even if user count is zero (see the following discussion of the time parameter)

†For customers, the own-code numbers 1 through 40 for the standard system installation parameters are linked. If numbers 41 through 127 are to be used, an installation parameter must be defined to permit the use of these numbers.

Bits 10-0—length

Number of words in program including header, excluding backwards pointer (word 0 of DRP); ENDOC is equated to (*) at end of program.

- VFD X10/TIME,N6/1 (Word 2)

Bits 15-6—interval in seconds after which program is to be released

Bits 5-0—user count

Number of users currently referencing this routine; on the disk this is set to 1.

- NUM 0 (Word 3)

Disk address—this word is zero on the disk.

When the program is loaded into core, its disk address is saved in this word until the program is released.

- NUM 0 (Word 4)

Time—this word is zero on the disk.

It represents the time in seconds when the program is to be released. It is set by EXDFUN. If the interval (see word 2) is not equal to zero, the program is not considered free to be released until the time specified in this word has arrived. If the interval equals zero, this word is disregarded.

- ADC START-OCHDR (Word 5)

This and following words (if any) in the header is the displacement from the beginning of program header to the first executable instruction (START). If there is more than one entry point, an ADC is required for each entry point. The value of each ADC is the displacement from the first word of the header (OCHDR) to the entry point.

When ENTPRG is called, the value in the A register is used as an index to these ADC's. For a program with only one entry point, A = 0 selects the single ADC.

An example of a header, therefore, would be figure B-2.

```
        NAM EXAMPL
         .
         .
         .
OCHDR   VFD N5/$18,X11/ENDOC-*  Word 1
        NUM 1,0,0               Word 2,3,4
        ADC START-OCHDR         Word 5
         .
         .
         .
START   LDA- FCNT,I     START OF PROGRAM
         .
         .
         .
```

Figure B-2. Own Code Header Example

The own code is retained in core for a time proportionate to the time control factor. If it is equal to zero, the area in the memory is immediately released for other system use when the execution of the routine has been completed, providing the user count is zero. Programs under time control with a user count of zero are normally retained in core until the time is elapsed, but if an outstanding core request is waiting and CAM cannot otherwise get enough core, it releases these areas even though time has not elapsed.

START Instruction

START is the first own-code instruction. Exit from the own code is standard and must comply with the coding in figure B-3.

```
EQU   EXDFUN($9F),EXTSR($92)
 .
 .
RTJ-  (EXDFUN)
JMP-  (EXTSR)
 .
 .
 .
EQU   ENDOC(*)       LAST CARD OF PROGRAM
END
```

Figure B-3. Own-Code Exit

If an error exit has to be made from the own code, the error routine is as follows:

```
ENQ     $B
JMP-    (ERROR)
EQU     ERROR ($96)
```

Own-code installation does not require system regeneration. It may be added to the own-code library by using the load program (LPR) supervisor command.

LPR Command

The LPR command is used to incorporate a new system function/own code or to replace an existing one in the PRD. Refer to the PRD table and also to the listing of the PRD program. To load a new function, the program is assembled, and a relocatable binary punch is obtained (on tape for example). Using the system routine LIBEDT, an absolutized punch (*P command in LIBEDT) is placed on the scratch area of the disk; for example, the following statements are typed in:

```
*LIBEDT

*K,I6,P8        (Assume relocatable
                 binary on 6)

*P

*Z
```

Get into supervisor mode, and enter the LPR command (refer to the CYBERDATA System Reference manual). The system function number is determined by referring to the PRD program listing for the appropriate index value. The program number (and system function number) for any own-code routine should fit the limit requirements as set at the beginning of this appendix and should not conflict with any routines already installed in the system. Refer to the PPR command in the CYBERDATA System Reference manual.

Examples of the LPR command are as follows:

- To replace the present supervisory command module THW:

  The PRD listing label for the ADC SISTHW is S11. The following command will put the new binary version of THW on the library:

  ```
  LPR,11,S
  ```

- To add a magnetic tape own-code, enter:

  ```
  LPR,1,T,O
  ```

  This loads the magnetic tape own code 1.

TEST RESOURCES

The following resources are available to special and own-code tests.

Active Terminal Table (ATT)

This table contains information that relates to the terminal which is executing the own-code or special test. The I register points to this table when a test routine is activated.

Get Current Character (GBYTE) (GSE)

The character to be returned is defined by RCNT (Character count in current record — word 4 in ATT). The calling sequence is:

```
EQU      GBYTE($94)
RTJ-     (GBYTE)
```

On exit:

A—7-bit ASCII character

Q—character including error indicator (bit 7)

Store Current Character (SBYTE) (GSE)

This routine stores the character found in word 2 (bits 0 through 7) of ATT into the location in the current record as defined by RCNT. The SBYTE routine uses the first temporary location in system status table (SST). The calling sequence is:

```
EQU      SBYTE($95)
RTJ-     (SBYTE)
```

ADD (SUB) Routine (AKMH)

This routine adds (subtracts) two four-word BCD† numbers for which the respective first word addresses (FWA) are AA and BB. See PACK and UNP.

On entry: Q - A = I

A—FWA of four-word buffer for BCD number AA (subtrahend for SUB routine)

Q—FWA of four-word BCD number BB (minuend for SUB routine)

I = 0—result not passed to user's buffer

≠ 0—I = FWA of four-word user's buffer for answer

†A BCD number is a packed decimal number, using the low-order 4 bits of the BCD code for the number. Exceptions are the digit zero, which is represented by a binary zero, and the plus and minus signs. The four-word BCD numbers then, have 15 digits plus a sign. The sign is the high-order 4 bits of the first word of the four-word buffer. A plus sign is all zeros (binary 0000), and a minus sign is all ones (binary 1111).

On exit from ADD (SUB)

Q—FWA of a four-word result buffer in ARITH module

Q-5 = FWA (A) Q-10 = FWA (Q)

The I register ~~must be~~ IS restored to the ATT address at the completion of these functions.

Calling sequence:

```
EXT     ADD,SUB
.
.
RTJ+    ADD
.
.
.
RTJ+    SUB
```

5

PACK Routine (ARITH)

This routine packs up to 15 digits of ASCII into a four-word BCD number. The field to be packed is the current field in the current record as defined in the ATT.

On exit from PACK:

Q—FWA of a four-word buffer containing 4-word BCD number (buffer is internal to ARITH module)

Calling sequence:

NON NUMERIC CHARACTERS ARE CONVERTED TO ZERO

```
EXT     PACK,UNP
.
.
RTJ+    PACK
.
.
.
RTJ+    UNP
```

UNP Routine (ARITH)

This routine changes a packed four-word BCD number (16 characters) into its equivalent ASCII value. See the PACK routine for calling sequence.

On entrance to UNP:

A—FWA of BCD number (four-word buffer)

On exit from UNP:

Q—FWA of an eight-word internal buffer in ARITH module containing the eight-word ASCII character string

(Q+8) = ORIGINAL A

Decimal to Binary (DB) (ARITH)

5B, 5S 5D

This routine converts decimal to binary.

On entering DB:

A—the BCD number in the range 0000 to 9999

On exit from DB:

A—binary (hexadecimal) equivalent of BCD number

Q = A on return

Calling sequence:

```
EXT     DB,BD
.
.
RTJ+    DB
.
.
.
RTJ+    BD
```

4

Binary to Decimal (BD) (ARITH)

5B, 5C.

This routine converts binary to decimal.

On entering BD:

A—binary number in the range 0000 to 9999_{10}

On exit from BD:

A—BCD equivalent of the binary number

Destroys Q = 0

OWN CODE TEST RESOURCES

The following resources are available for own-code tests only.

ENTSR, EXTSR Routines (ENEXSV)

These routines are used to make a subroutine reentrant. When using these routines, exit from the subroutine should be through EXTSR. The call to ENTSR must immediately follow the subroutine entry point. ENTSR and EXTSR must be used for all routines which perform input/output, including display.

ENTSR CLEAR RELATIVE STACK ADDRESS SET.
EXTSR SET RELATIVE STACK REC FROM ADDR [illegible]

Calling sequence:

```
        EQU   ENTSR($91),EXTSR($92)
        .
        .
        .
    NAM ADC   0       ENTRY POINT
        RTJ-  (ENTSR) Q, A, I, SAVED
              NO OTHER TEMPS USED
        .
        .
        .
        .
        JMP-  (EXTSR) A,Q,I SAVED
```

RTNSR Routine (ENEYSR)

This routine returns the last address from the exits stack and removes it from the stack (see ENTSR).

On exit:

Q—last address in exits stack

Calling Sequence:

```
    EQU     RTNSR($98)  A,I SAVED
    .
    .
    RTJ-    (RTNSR)
```

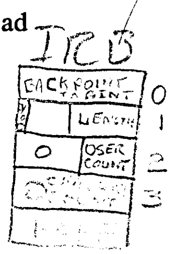
GETBUF Routine (PIO)

This routine allocates the user (interrogate) buffer and sets the interrogate buffer pointers in the ATT.

On entry:
A—number of characters to be read

Calling sequence:

```
    EXT     GETBUF
            .
            .
            .
    RTJ÷    GETBUF
```



BUF IS FILLED WITH $FFFF

CHAR COUNT FOR ABOVE [illegible]

GETSTR Routine

This routine initiates a read request from a key display into an interrogate buffer to read in a string of characters.

On entry into GETSTR:

A—number of characters to be read; GETSTR does a call to GETBUF and exits to Dispatcher; actual character transfer is done by MPC; return to caller on the following conditions:

1. Requested number of characters read
2. REL key pressed
3. CANCEL key pressed

Calling sequence:

```
    EQU     GETSTR($A2)
    RTJ-    (GETSTR)
```

GETBIN, GETALP, and GETCOM Routines (SCANV)

The get binary number (GETBIN), get one alphanumeric character (GETALP), and get comma (GETCOM) routines scan the interrogate buffer starting at the current character position as defined in SST + STEMP (word 20 of SST). GETBIN accepts only 0 to 9, comma, and EOR. Any other character causes all input to be rejected. Blanks are treated as part of the input stream.

Calling sequence:

```
    EQU     GETBIN($99)
    EQU     GETALP($9B)
    EQU     GETCOM($9C)
    .
    .
    .
    .
    RTJ-    (entry)         entry - name of module
```

GETBIN exit conditions

$A \geq 0$	$Q = 0$	Comma (A—binary number)
$A \geq 0$	$Q < 0$	EOR (A—binary number)
$A < 0$	$Q < 0$	Error: nonnumeric

GETALP exit conditions:

$A \neq 0$	$Q \neq 0$ =1	A—ASCII character
$A = 0$	$Q = 0$	Comma
$A \neq 0$ FF	$Q \gtrless 0$	EOR (A = $FF)
$A \lessgtr 0$	$Q \lessgtr 0$	Error (not ASCII)

GETCOM exit conditions:

A = 0	Q = 0	Comma
A ≠ 0	Q < 0	EOR (A = $FF)
A ≠ 0	Q ≠ 0	A—nonblank character
A < 0	Q ≠ 0	Error: non-ASCII character

GFCORE Routine

The get format (GFCORE) routine releases the current format which is in use, gets a new format as specified, and sets up linkage in ATT.

On entry:

Q—format number (bits 14 to 0)

bit 15 = 0—return to ERROR in case of error
= 1—return to caller regardless; set A=0, if error

Calling sequence:

```
EQU       GFCORE($9D)
.
.
RTJ-      (GFCORE)
```

CAM Routine (CAM)

The core allocation module (CAM) routine honors requests for core space by allocating a contiguous block of core in the formats records programs (FRP) buffer area. If a contiguous block of sufficient length is not available, CAM attempts to create an adequate block by squeezing active blocks in the FRP buffer.

On entry:

Q—block length requested (includes three-word header)

On exit from CAM:

A—0, if space was allocated

—1FFFF, if not

Q—beginning address of allocated space, if space was allocated

—block length, if not

0 BP = 0
1 LTH = Q-1
2 USER = 1

Calling sequence:

```
EQU       CAM($AE)
.
.
RTJ-      (CAM)
.
.
```

RELFOR Routine (GFCORE)

The release format (RELFOR) routine is used to release a buffer allocated by CAM. RELFOR decrements the user count in the buffer header by one. If the user count ≠ 0, RELFOR exits to the caller; otherwise, it sets the location addressed by the backwards pointer to 0, sets bit 14 of word 1 in the SST (the storage move flag) to 1, and exits to the caller.

On entry:

Q—FWA of buffer to be released

Calling sequence:

```
EQU       RELFOR($AD)
.
.
RTJ-      (RELFOR)
```

EXIT Q = CONTENTS B.P. IF
USER COUNT = 0

SETMOV Routine (EN[illegible])

This routine allows program movement by CAM by clearing bit 15 of word 1 of the program header (I/O status bit). It also sets bit 15 (relative/absolute) of word 21 in the ATT. ENTSR checks this bit to see if a return address should be relative or absolute. This routine is not necessary for one-shot routines.

Calling sequence:

```
EXT       SETMOV
.
.
RTJ+      SETMOV
.
.
```

EXDFUN Routine ([illegible])

The exit disk function (EXDFUN) routine allows release of a disk-resident program from core and must be called prior to exiting to the dispatcher or ~~the~~ use CLRSR. When exiting to ERROR, EXDFUN is called by that routine. EXDFUN decrements the user count by 1, and if the program is under time control, the time until release is updated. If the program is not under time control, the program area is released. EXDFUN is an internal subroutine in ENTPRG.

Calling sequence:

```
    EQU        EXDFUN($9F)
    .
    .
    RTJ-       (EXDFUN)
    .
    .
```

ERROR Routine (65E)

This is the error exit routine for CYBERDATA routines.

On entry:

Q—error number
    Q = 9 is special test error
    Q = $B is own-code error

Calling sequence:

```
    EQU        ERROR($96)
    .
    .
    ~~RTJ-~~   (ERROR)
    JmP
```

ERROR CODES

Error codes currently defined for CYBERDATA are listed in table B-1.

MOVREC Routine (P7C0)

This routine moves a block from one core address to another. There are two forms available for the MOVREC routine. Form 1 moves the block from a relative address to another relative address. The second form moves the block from an absolute address to an absolute address.

Table B-1. CYBERDATA Error Codes

Code	Index	Meaning
A1	$1	Data type error X
A2	$2	Sign missing
A3	$3	Illegal character X
A4	$4	Illegal function key
A5	$5	Mismatch ch X (Y)
B1	$6	Beginning of data
B2	$7	End of data
C1	$8	Limit error
C2	$9	Special test error
C3	$A	Check digit error
C4	$B	Own-code test error
C5	$C	Nonzero error
D1	$D	Invalid operator number
D2	$E	Invalid format number
D3	$F	Invalid document number
D4	$10	Invalid job name
D5	$11	Invalid batch number
E1	$12	Illegal request
E2	$13	Illegal parameter
F1	$14	Format number in document error
F2	$15	Own code is not on disk
F3	$16	Disk full
F4	$17	Core full, please wait
F5	$18	Function busy, please wait
LL	$19	System lock wait
PP	$1A	Parity error
**	$1B	Field with error flag
C6	$1C	Balancing error
D6	$1D	Invalid auto sequence count
F6	$1E	Disk deselected

NOTE RETURN FROM I/O A = ESTATZ
Q = V BITS

Form 1 calling sequence:

```
EXT     MOVREC
RTJ     MOVREC
ADC     (FROM-*)
ADC     (TO-*)
NUM     NUMWRD    NUMBER OF WORDS
```

Form 2 calling sequence:

```
EXT     MOVREC
RTJ     MOVREC
ADC     FROM
ADC     TO
NUM     NUMWRD
```

NOTE

These forms can be mixed.

WRITEC Routine (PIO)

The write current record (WRITEC) routine is used to write out a record from the terminal record buffer (TRB) which has not been written before.

Calling sequence:

```
EQU     WRITE($A3)
RTJ-    (WRITEC)
```

REWRIT Routine (PIO)

This routine rewrites the current record in TRB after corrections have been made to it.

Calling sequence:

```
EQU     REWRIT($A4)
RTJ-    (REWRIT)
```

READP Routine ([illegible])

Read the previous record as specified in TRB. If the previous disk address (PREVDA) is negative, then the current record is the first record of the batch.

Calling sequence:

```
EQU     READP($A5)
RTJ-    (READP)
```

READN Routine

This routine is used to read the next record as in TRB. If the next disk address (NXDA) bit 15 is set, the current record is the last record of the batch.

Calling sequence:

```
EQU     READN($A6)
RTJ-    (READN)
```

ENTPRG Routine ([illegible])

This routine loads disk-resident programs as defined in the PRD module. Prior to calling:

A = INDEX INTO ENTRY POINTS

```
Q15—0 —load and go
    1 —load only
Q14—12—program group
       0—miscellaneous functions
       1—key function
       2—interrogate function
       3—supervisor function
       4—magnetic tape function
       5—undefined
       6—undefined
       7—own code                 SEE BIT 9
Q11—0 —if core is not available, return to caller
       ~~with (Q) = 0~~ or, to REJSUP if a
       supervisor function
    1 —wait till core is available
Q10—0 —called routine is not a subroutine
    1 —called routine is a subroutine CALL MUST BE CORE RESIDENT
Q9 ~~Q8~~—7—unused  1 GO TO ERROR ROUTINE
                    0 RETURN TO USER (Q) = 0
Q6—0—index into program group
A ~~Q~~7—0—entry-point number (0 for own code)
```

This routine uses temporary storage words 2 to 5 (STEMP) in the ATT. An example of one own code passing control to another is in figure B-4.

SETS I/O BIT, BUMPS USER COUNT

```
EQU ENTPRG($9E),EXDFUN($9F),EXTSR($92)

Own code number 1 exit

RTJ-  (EXDFUN)    release this routine

RTJ-  (ENTPRG)    call next own code

Own code number 2 exit

RTJ-  (EXDFUN)    release this routine

RTJ-  (EXTSR)     exit subroutine
```

Figure B-4. Example of Own Code Passing Control

CLCDWA Routine (PIO)

This routine calculates a disk word address for use with unformatted reads or writes (see REDWRD and WRTWRD).

On entry:

(A)—sector number
(Q)—offset in number of words

On exit:

(A)—least-significant bits
(Q)—most-significant bits

Calling sequence:

```
EQU     CLCDWA($90)
RTJ-    (CLCDWA)
```

REDWRD Routine (PIO)

This routine reads a string of words, starting at a specified disk word address (unformatted read). If only the sector address and offset are known, use CLCDWA to obtain the word address prior to performing REDWRD. The format is shown in figure B-5.

```
              EQU    REDWRD($A9)

              RTJ-   (REDWRD)

Select   {    ADC    buf
 one     {                    Buffer address
              ADC    (buf-*)  LOWER 32K
                               ONLY
              NUM    msb      Most-significant bits

              NUM    lsb      Least-significant bits

              NUM    nw       Number of words

              NUM    lu       Logical unit number
```

Figure B-5. REDWRD Routine Calling Sequence

WRTWRD Routine (PIO)

This routine writes a string of words, starting at a specified disk word address (unformatted write). if only the sector address and offset are known, use CLCDWA to obtain the word address prior to performing WRTWRD. The format is shown in figure B-6.

```
              EQU    WRTWRD($AA)

              RTJ-   (WRTWRD)

Select   {    ADC    buf
 one     {                    Buffer address
              ADC    (buf-*)  LOWER 32K
                               ONLY
              NUM    msb      Most-significant bits

              NUM    lsb      Least-significant bits

              NUM    nw       Number of words

              NUM    lu       Logical unit number
```

Figure B-6. WRTWRD Routine Calling Sequence

REDSEC Routine (PIO)

The REDSEC routine reads either:

- A specified number of sectors
- The first 96-nwds words of one sector into core (formatted read)

The routine format is given in figure B-7.

```
              EQU    REDSEC($A7)

              RTJ-   (REDSEC)

Select   {    ADC    buf
 one     {                    Buffer address
              ADC    (buf-*)  LOWER 32K
                               ONLY
              NUM    sec      Sector address

Select   {    NUM    nsec     Number of sectors
 one     {
              NUM    -nwds    Minus the number of words

              NUM    lu       Logical unit number
                               0-3
```

Figure B-7. REDSEC Routine Calling Sequence

BUF IN LOWER 32K UNLESS RTJ ABSADD
BEFORE CALL — REDWRD, WRTWRD, REDSEC, WRTSEC

WRTSEC Routine (PIO)

This routine writes either:

- A specified number of sectors
- The first 96-nwds of one sector onto a disk (formatted write)

The format is given in figure B-8.

```
              EQU    WRTSEC($A8)

              RTJ-   (WRTSEC)

Select     {  ADC    buf
 one       {                          Buffer address
           {  ADC    (buf-*)
                LOWER 32K ONLY
              NUM    sec              Sector address

Select     {  NUM    nsec             Number of sectors
 one       {
           {  NUM    -nwds.           Minus the number of words

              NUM    lu               Logical unit number
```

Figure B-8. WRTSEC Routine Calling Sequence

FOR SUPY ONLY

SUPRW/TAPIO Routine (PIO)

The supervisor READ/WRITE and MOTION routine is used for reading from the supervising input unit and writing to the supervisory list unit. The TAPIO routine is for reading/writing magnetic tape.

On entry:

```
   Q = 0—READ ASCII
       1—WRITE ASCII
       2—FREAD ASCII
       3—FWRITE ASCII
       4—MOTION of input unit / ← INPUT O
       5—MOTION of output unit
       8—READ binary
 1     9—WRITE binary
 2     A—FREAD binary
 3     B—FWRITE binary
```

```
BIT 0   0 = READ          1 = WRITE (0=INP 1=OUT)
    1   0 = UNFORMATTED   1 = FORMATTED
    2   0 = I/O           1 = MOTION
    3   0 = ASCII         1 = BINARY
```

TAPE CU MUST BE SPECIFIED

For the following requests, A specifies the code conversion:

```
0   Q = 16₁₀—READ code
1       17₁₀—WRITE code
2       18₁₀—FREAD code
3       19₁₀—FWRITE code

    A = 1—BCD
        2—EBCDIC
        3—ASCII
        4—user code 1
        5—user code 2
        6—user code 3
        7—user code 4
```

Calling sequence:

The calling sequences are based on EQU SUPRW($97),TAPIO($9A). Entry is either SUPRW or TAPIO.

The sequence for data transfer is one of the following:

```
• RTJ-    (ENTRY)
  ADC     BUFF
  NUM     X

• RTJ-    (ENTRY)
  ADC     (BUFF-*)
  NUM     X

• RTJ-    (ENTRY)
  ADC     0
  NUM     X
```

Where:

```
ADC    BUFF     —absolute buffer address
ADC    (BUFF-*)—relative buffer address      I/O BIT
ADC    0        —buffer is interrogate buffer ADMINISTERED BY PIO
NUM    X        —number of words to be
                 transferred
```

The sequence for motion is:

```
RTJ-    (ENTRY)
NUM     X
```

TAPIO REQUESTS AFFECT BCOUNT

Where:

NUM X—standard system motion code word (figure B-9)

15 — 12	11 — 8	7 — 4	3 — 0
p_1	p_2	p_3	dy

Figure B-9. Motion Code Word

In figure B-9, the word is defined as follows:

BLOCK COUNT

p_1, p_2, p_3—motion control parameters

- 0—terminate request
- 1—backspace one record —1
- 2—write end of file 1
- 3—rewind 0
- 4—rewind and unload 0
- 5—skip one file forward -0
- 6—skip one file backward -0
- 7—advance one record 1

dy—density parameter

- 0—no change
- 1—800 bpi
- 2—556 bpi
- 3—200 bpi
- 4—1,600 bpi

Up to three motion requests can be made with the standard word; for example:

```
NUM     $1101
```

This request means to backspace two records on an 800-bpi tape.

If several iterations of a motion request are needed, use the following format:

15	14 13 12 11	0
1	p	n

In this request, the 1 in bit 15 indicates that the request can be repeated, p is the motion control parameter, and n is the number of iterations (maximum is 4095).

Note

1. If the interrogate buffer is used to read data into, and no buffer is available or the buffer is not long enough, the program allocates a new buffer.
2. The supervisor input/list units being a zero indicate a CYBERDATA keydisplay unit.
3. Keydisplay units honor only write end-of-file motion requests which are executed as a clear display.
4. If the list unit is a keydisplay unit, and end-of-display area is indicated by Q = $C000 on completion.
5. If the read unit is a keydisplay unit, function lock is not turned on during the read.

HOW TO WRITE CODE-CONVERSION ROUTINE

ENTERED AS SUBROUTINE

In addition to the system's recording codes (ASCII, BCD, EBCDIC), the customer may add four more recording codes. The recording code routines are installed by the following command:

```
LPR,n,T,C
```

In this command, n is the code number (1, 2, 3, 4). These code numbers are used to identify the recording code in the EMT command. The code-conversion routines receive three parameters:

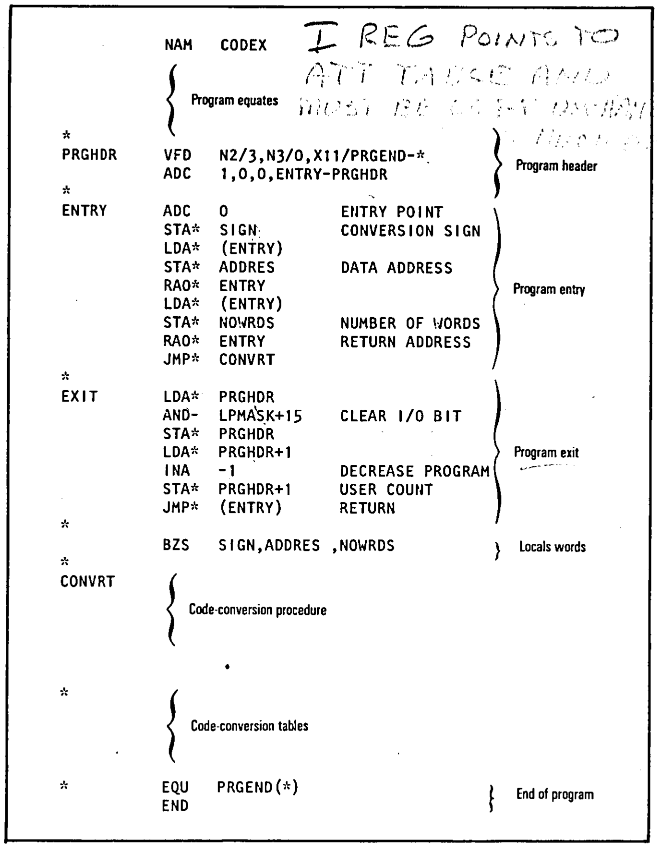
PARAMETER 1
- Address of the data to be converted

PARAMETER 2
- Number of words

A REGISTER
- Sign to convert the data from ASCII to the code (positive number) or from the code to ASCII (negative number)

According to these parameters, the code-conversion routines replace the original data with the converted data.

A code-conversion program should have the structure in figure B-10.



I REG POINTS TO ATT TABLE AND MUST BE LEFT UNCHANGED [illegible]

```
        NAM   CODEX
        {  Program equates
*
PRGHDR  VFD   N2/3,N3/0,X11/PRGEND-*          } Program header
        ADC   1,0,0,ENTRY-PRGHDR
*
ENTRY   ADC   0               ENTRY POINT
        STA*  SIGN            CONVERSION SIGN
        LDA*  (ENTRY)
        STA*  ADDRES          DATA ADDRESS
        RAO*  ENTRY                            } Program entry
        LDA*  (ENTRY)
        STA*  NOWRDS          NUMBER OF WORDS
        RAO*  ENTRY           RETURN ADDRESS
        JMP*  CONVRT
*
EXIT    LDA*  PRGHDR
        AND-  LPMASK+15       CLEAR I/O BIT
        STA*  PRGHDR
        LDA*  PRGHDR+1                         } Program exit
        INA   -1              DECREASE PROGRAM
        STA*  PRGHDR+1        USER COUNT
        JMP*  (ENTRY)         RETURN
*
        BZS   SIGN,ADDRES ,NOWRDS              } Locals words
*
CONVRT  {  Code-conversion procedure

        .
*
        {  Code-conversion tables

*       EQU   PRGEND(*)                        } End of program
        END
```

Figure B-10. Code-Conversion Structure

HOW TO WRITE A TAPE-LABEL PROGRAM

Tape-label programs are disk-resident routines for writing volume headers and trailers as well as file headers and trailers. Labels are written on the tape at the position it is in when the label program receives control. Code conversion is performed according to the table specified by the magnetic tape format which is being used. To write a label program, the programmer must be familiar with the CYBERDATA table structure and the more frequently used internal routines. Of special interest are the following areas as well as the SST, EST, ATT, and INT:

- MTCA—magnetic tape communication area (ATT) contains the magnetic tape format, pointers to the EOD area (header and trailer information), and entry and exit parameters
- EOD—CYBERDATA trailer which contains all relevant information about the last set of data recorded on the tape

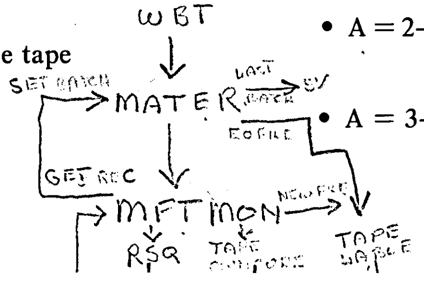
The TLP program header has four fixed words and four entry points. The first entry is used for the volume header, the second for the file header, the third for the file trailer, and the last for the volume trailer. A sample TLP program header is shown in figure B-11.

```
LBL7   VFD    N1/1,N1/0,N3/0,N11/ELBL7-*
       VFD    N10/0,N6/1
       NUM    0
       NUM    0
       ADC    VOLHDR-LBL7          (Volume header)
       ADC    FILHDR-LBL7          (File header)
       ADC    FILTRL-LBL7          (File trailer)
       ADC    VOLTRL-LBL7          (Volume trailer)
```

Figure B-11. TLP Program Header Format Example

Label routines may be designed specially by the analyst, following the previously mentioned guidelines for fixed headers and entry points as well as the following guidelines for exiting from a label program. The program LBL1, which deals with IBM standard labels, contains many subroutines. Two exits should exist in the program:

- Exit to continue writing on the tape
- Error exit



The first exit is done by the following instructions:

```
RTJ-      (EXDFUN)
ENA 0
```

```
RTJ-      (EXDFUN)
ENA       0
LDQ-      MTFEXT,I      (Second word
                        of MTCA;
RTJ-      (ENTPRG)
```

Error exit:

```
RTJ-      (EXDFUN)
ENA       4 —— 3
LDQ-      MTFEXT, I
RTJ-      (ENTPRG)
```

NOTE

When the program *takes* new areas using CAM, it must release those areas, too.

MAGNETIC TAPE OWN CODES

The user has the opportunity to manipulate the data, arrange his own blocking, or add constant data by defining a magnetic tape own code. The system gives control to this routine at three stages of processing, passing a parameter indicating the stage:

- Next record in TRB buffer (MTFENT = -1)
- End of batch—no record in buffer (MTFENT = 1)
- End of tape—no record in buffer (MTFENT = 2)

RETURN TO A = 0

For the processing, MTFENT is word 46_{10} of the ATT and the TRB address is in word 6 of the ATT.

When the magnetic tape own code has completed its own processing, it must load the index of MFTMON (set in word 47_{10} of the ATT and known as MTFEXT) in register Q and perform a return jump to ENTPRG with the A register set to one of the following:

- A = 0—get next record (delete current record from output buffer)
- A = 1—put current record into the output buffer and get next record
- A = 2—write the output buffer to tape, excluding the current record

MTOC RESCHEDULED

- A = 3—terminate this WBT command

As a disk-resident routine, each magnetic tape own code must have the standard header and must perform an RTJ- (EXDFUN) before exiting: AFTER EACH RECORD.

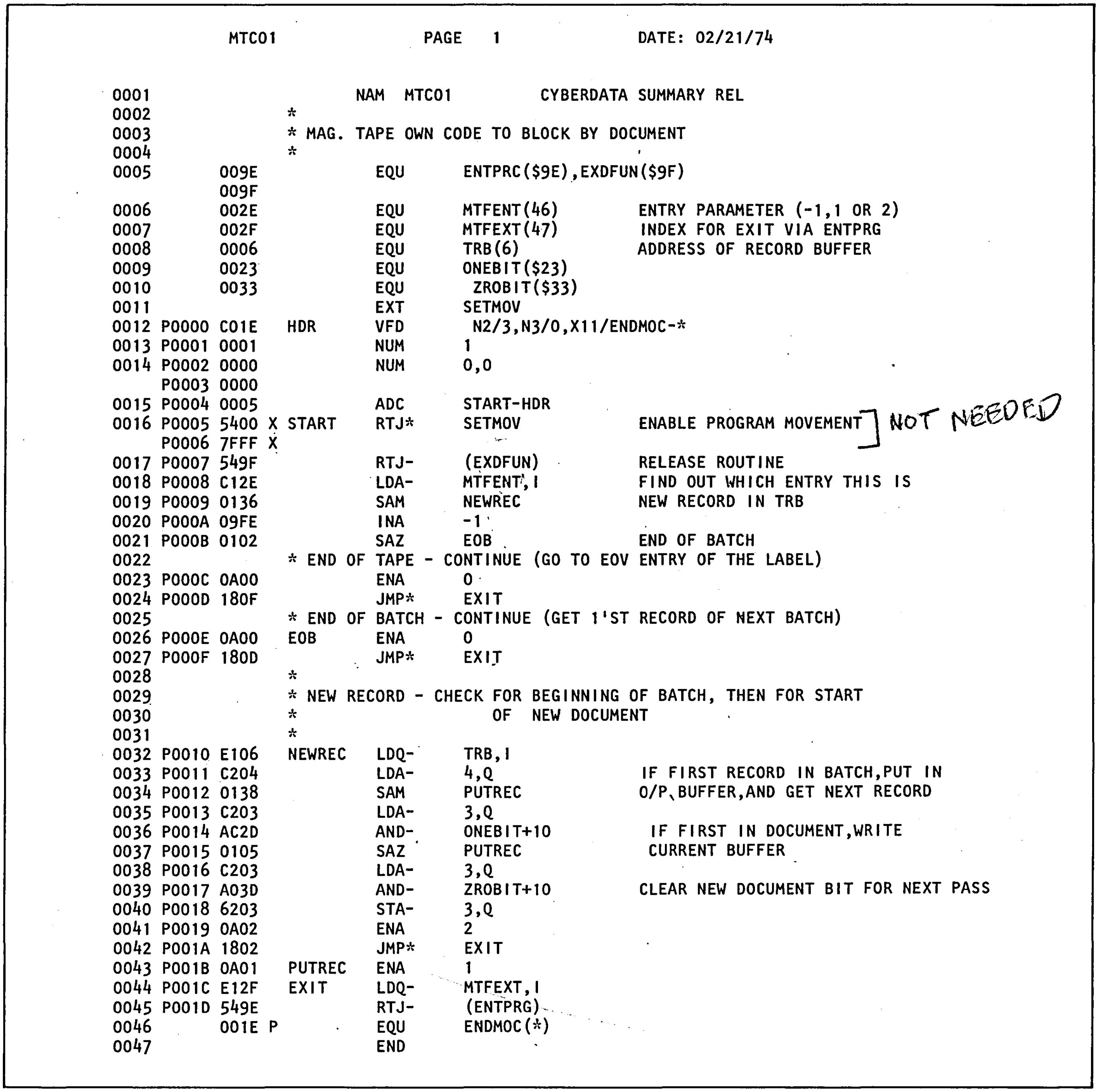
EQU EXDFUN ($9F)

OWN CODE EXAMPLE

Blocking by document is required, but the number of records in a document can vary. Assume that the maximum number of records is 12, each of which consists of 80 characters. Define an MTF with undefined blocking and a maximum block length of 960. Then write a short own code as in figure B-12.

NOTE

RTJ- (EXDFUN) should be performed after any input/output that may be necessary. In the routine in figure B-12, no input/output is performed. To save code, the EXDFUN may be performed earlier.



```
           MTC01                PAGE   1            DATE: 02/21/74

                          NAM  MTC01        CYBERDATA SUMMARY REL
             *
             * MAG. TAPE OWN CODE TO BLOCK BY DOCUMENT
             *
      009E             EQU      ENTPRC($9E),EXDFUN($9F)
      009F
      002E             EQU      MTFENT(46)          ENTRY PARAMETER (-1,1 OR 2)
      002F             EQU      MTFEXT(47)          INDEX FOR EXIT VIA ENTPRG
      0006             EQU      TRB(6)              ADDRESS OF RECORD BUFFER
      0023             EQU      ONEBIT($23)
      0033             EQU       ZROBIT($33)
                       EXT      SETMOV
P0000 C01E   HDR       VFD       N2/3,N3/0,X11/ENDMOC-*
P0001 0001             NUM      1
P0002 0000             NUM      0,0
P0003 0000
P0004 0005             ADC      START-HDR
P0005 5400 X START     RTJ*     SETMOV              ENABLE PROGRAM MOVEMENT] NOT NEEDED
P0006 7FFF X
P0007 549F             RTJ-     (EXDFUN)            RELEASE ROUTINE
P0008 C12E             LDA-     MTFENT,I            FIND OUT WHICH ENTRY THIS IS
P0009 0136             SAM      NEWREC              NEW RECORD IN TRB
P000A 09FE             INA      -1
P000B 0102             SAZ      EOB                 END OF BATCH
             * END OF TAPE - CONTINUE (GO TO EOV ENTRY OF THE LABEL)
P000C 0A00             ENA      0
P000D 180F             JMP*     EXIT
             * END OF BATCH - CONTINUE (GET 1'ST RECORD OF NEXT BATCH)
P000E 0A00   EOB       ENA      0
P000F 180D             JMP*     EXIT
             *
             * NEW RECORD - CHECK FOR BEGINNING OF BATCH, THEN FOR START
             *                       OF  NEW DOCUMENT
             *
P0010 E106   NEWREC    LDQ-     TRB,I
P0011 C204             LDA-     4,Q                  IF FIRST RECORD IN BATCH,PUT IN
P0012 0138             SAM      PUTREC               O/P,BUFFER,AND GET NEXT RECORD
P0013 C203             LDA-     3,Q
P0014 AC2D             AND-     ONEBIT+10             IF FIRST IN DOCUMENT,WRITE
P0015 0105             SAZ      PUTREC                CURRENT BUFFER
P0016 C203             LDA-     3,Q
P0017 A03D             AND-     ZROBIT+10           CLEAR NEW DOCUMENT BIT FOR NEXT PASS
P0018 6203             STA-     3,Q
P0019 0A02             ENA      2
P001A 1802             JMP*     EXIT
P001B 0A01   PUTREC    ENA      1
P001C E12F   EXIT      LDQ-     MTFEXT,I
P001D 549E             RTJ-     (ENTPRG)
      001E P           EQU      ENDMOC(*)
                       END
```

Figure B-12. Sample Own Code for Magnetic Tape

APPENDIX C

DISK AND MEMORY ALLOCATION

Disk and Memory Allocation

GENERAL

This appendix summarizes the operating system and the CYBERDATA memory-resident and disk-resident routines and tables. The discussion includes a description of the allocation of memory.

MEMORY-RESIDENT MODULES

The memory-resident modules are discussed on the following pages.

Operating System

The operating system used for CYBERDATA has the following assigned with special definitions:

- SYSDAT

  This contains system data as described in the CYBERDATA system status table (SST) and the extended status table (EST).

- SPACE

  This is modified to allow different configurations of allocatable core and background depending on flags and values in the CYBERDATA extended status table (EST); detailed discussion in allocatable memory

- Memory-Resident Drivers

  The only operating system drivers that are memory-resident are D17332 (disk drive) and D1711 (TTY driver); all others are disk-resident (The CYBERDATA terminal drivers are not a part of the operating system.)

All other memory-resident operating system modules are in their standard form.

CYBERDATA Memory-Resident Modules

The following special memory-resident modules apply to the CYBERDATA System:

- Display drivers

  These are general display routines and do not apply to a specific type of terminal.

- Self-scan routines

  These routines are used to drive the 32-character, self-scan terminals. If a particular system configuration does not include these terminals, these routines can be removed from the system to reduce the size of the memory-resident portion of the system.

- CRT routines

  These routines are used to drive the 480-character CRT terminals. If a particular system configuration does not include these terminals, these routines can be removed from the system to reduce the size of the memory-resident portion of the system.

- Processing routines

  These are the high-usage routines in the CYBERDATA system. The modules CON029 and CONTTY are necessary for using 029 keypunch keyboards and TTY keyboards on terminals. If one of these keyboards is not used in a particular configuration, its corresponding module can be removed from the system.

- Special test area

  The check-digit tests are special tests and are already in this area on the released system. Additional tests can be added here.

DISK-RESIDENT ROUTINES

The routines, which are disk-resident, are discussed in the following paragraphs.

Operating System

All of the standard routines for job processing, library editing, system maintenance, and utilities are disk-resident.

CYBERDATA Routines

- Start-up routines

  There are five CYBERDATA routines which are accessible through the use of MIPRO. They are VLSTRT, VLTYPE, VLBGOP, VLOSOP, and VLSTTS. These routines are accessed with the mnemonics VL, VLTP, VLBG, VLOS, and VLST. For more information on these routines, see allocatable memory also in this appendix.

- Terminals type table

  This table is modified by VLTYPE and contains information for each terminal in the system.

- Disk buffers

  Buffers are defined and reserved on disk using module DUMMYV, which is a one-word dummy, and various LIBEDT commands.

- Miscellaneous routines

  Routines in this category do not have a direct relationship to function keys, interrogate commands, or supervisor commands. A typical routine in this category is PB2, which is called by the module PBS. These modules may be placed in the library by using the LPR command.

- Key functions

  These routines have a one-to-one correspondence to function keys. These routines may also be placed in the library by using the LPR command.

- Interrogate functions

  These functions have a one-to-one correspondence to interrogate commands. These routines also may be placed in the library by using the LPR command.

- Supervisor functions

  The supervisor functions have a one-to-one correspondence to supervisor commands. They also can be placed in the library by using the LPR command.

- Own-code insertion area

  As an alternative to loading all the own code by using the LPR command each time the system is reinstalled, they can be inserted in the install tape so that they are automatically linked and loaded when the install is done. To load a new own code by using the LPR command, an entry for that own code must have been defined on the install tape in the form *S,SICXXX where XXX is the own-code number. The install tape, as released, contains entries for 40 own codes.

ALLOCATABLE MEMORY

To enhance the capability of running the CYBERDATA operating system with execution of background programs (either simultaneously or separately) several parameters have been specified to control partitioning of allocatable memory. See figure C-1.

At system configuration time, standard COS operation allows the specification of levels in allocatable memory. These levels determine what portion of memory can be available to a program requesting memory at a certain priority.

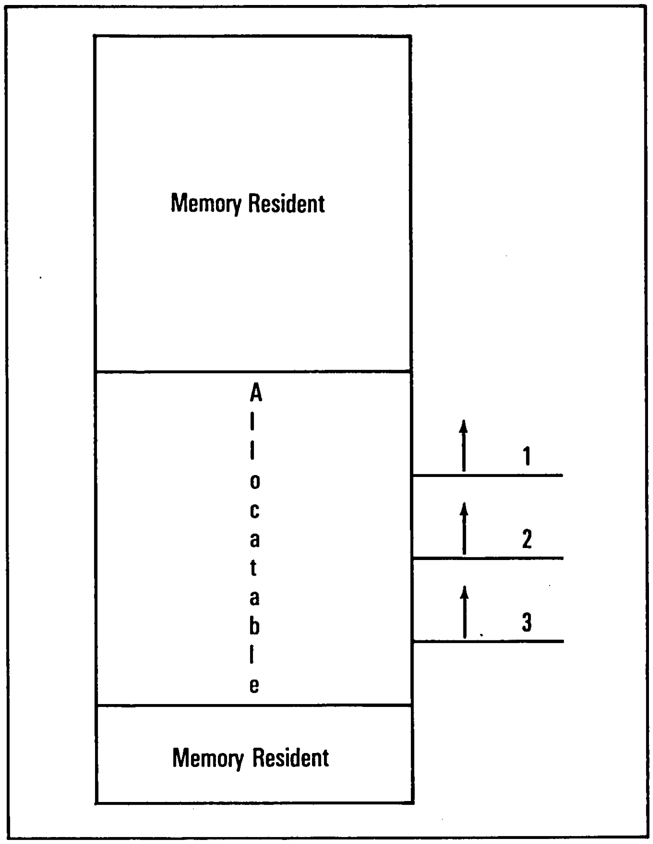


Figure C-1. Example of Allocatable Memory

In the example in figure C-1, only area 1 is available to programs requesting space at priority 1. Areas 1, 2, and 3 are available to programs requesting memory at priority 3.

At startup, the VLSTRT module requests one large partition of allocatable memory for table, data-buffer, and disk-resident program use. This request is made at priority level 5; therefore, the length of area 5 determines in what area CYBERDATA will always be able to receive even if lower-priority programs are running.

The following parameters in SYSDAT determine the length of area 5 and the way in which allocatable memory is requested by CYBERDATA:

- VLOS
- VLBCKG
- VLA5S
- BGCORE
- LFTOVR
- TOTAVL

VLOS Parameter

VLOS is used by the Restart program to determine the size of area 5.

- VLOS = NONE

  Area 5 is set to 0. No area is specially reserved for level 5 programs.

- VLOS = STD

  Area 5 is set to length VLA5S, which is an installation parameter (currently 1000_{16}).

- VLOS = BCKG

  Area 5 is set to the total length of allocatable memory minus BGCORE (an area to be left available to background programs) minus areas 1 through 3 specified at install time.

For both the STD and BCKG options, TOTAVL (total available) is set to the full length of allocatable memory minus LFTOVR. LFTOVR is an installation parameter (currently 280_{16}) that defines an area that will be available for loading DEBUG when CYBERDATA has taken the maximum memory.

The value of VLOS may be changed by using the program VLOSOP. VLOSOP is called with the following sequence:

```
MI    VLOS    CR
```

System response:

```
VLOS IN
```

Enter one of the following:

- STD
- BCKG
- NONE

Any other input results in the message DATA ERROR. The program then exits with the response:

```
VLOS OUT
```

The new value of VLOS takes effect at the next Autoload and Restart.

VLBG Parameter

VLBG is used by the CYBERDATA start-up program to determine the number of words to be requested from allocatable memory for the CYBERDATA buffer.

- VLBG = NONE

  TOTAVL words are requested.

- VLBG = MUST

  Area 5 is requested.

- VLBG = RQST

  An optimum number of words is requested. This figure is based on the NO. OF STATIONS and AVERAGE RECORD LENGTH parameters entered at start-up time.

The value of VLBG may be changed by using the program VLBGOP. VLBGOP is called with the following sequence:

```
MI    VLBG    CR
```

System response:

```
VLBG IN
```

Enter one of the following:

- xxxx,NONE
- xxxx,MUST
- xxxx,RQST

The xxxx is the hexadecimal value desired for the BGCORE parameter. Any other input results in the message DATA ERROR.

The program responds with:

```
VLBG OUT
```

The new value of VLBG takes effect at the next Autoload and Restart.

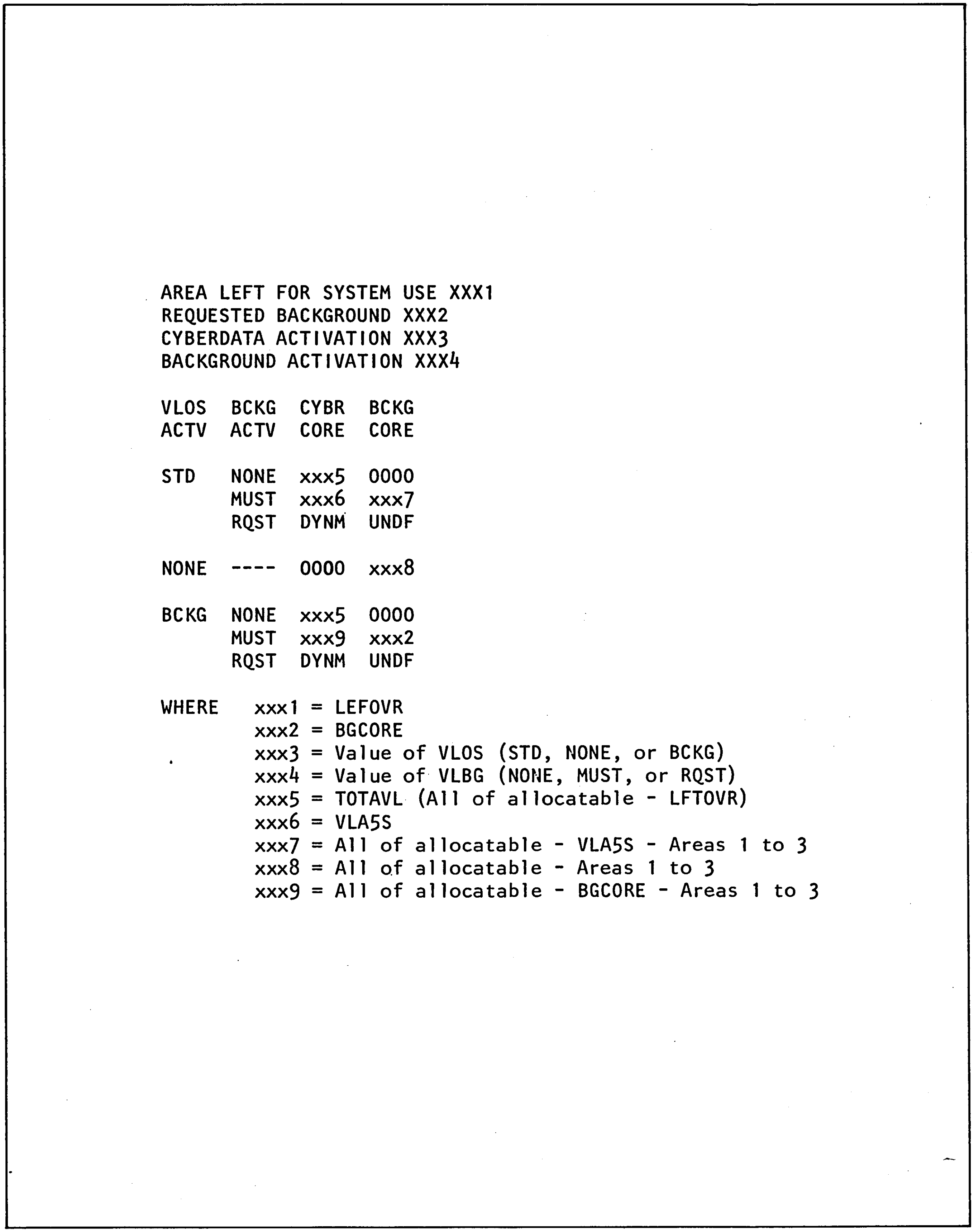
The five examples in figure C-3 show how allocatable memory would be partitioned when given the different values of the parameters VLOS and VLBG.

VLST Program

The program VLST can be called to give a printout of what the result of the different allocation schemes will be. The program is called with the following sequence:

```
MI    VLST    CR
```

The resulting printout is shown in figure C-2.



```
AREA LEFT FOR SYSTEM USE XXX1
REQUESTED BACKGROUND XXX2
CYBERDATA ACTIVATION XXX3
BACKGROUND ACTIVATION XXX4

VLOS  BCKG  CYBR  BCKG
ACTV  ACTV  CORE  CORE

STD   NONE  xxx5  0000
      MUST  xxx6  xxx7
      RQST  DYNM  UNDF

NONE  ----  0000  xxx8

BCKG  NONE  xxx5  0000
      MUST  xxx9  xxx2
      RQST  DYNM  UNDF

WHERE   xxx1 = LEFOVR
        xxx2 = BGCORE
        xxx3 = Value of VLOS (STD, NONE, or BCKG)
        xxx4 = Value of VLBG (NONE, MUST, or RQST)
        xxx5 = TOTAVL (All of allocatable - LFTOVR)
        xxx6 = VLA5S
        xxx7 = All of allocatable - VLA5S - Areas 1 to 3
        xxx8 = All of allocatable - Areas 1 to 3
        xxx9 = All of allocatable - BGCORE - Areas 1 to 3
```

Figure C-2. Sample VLST Printout

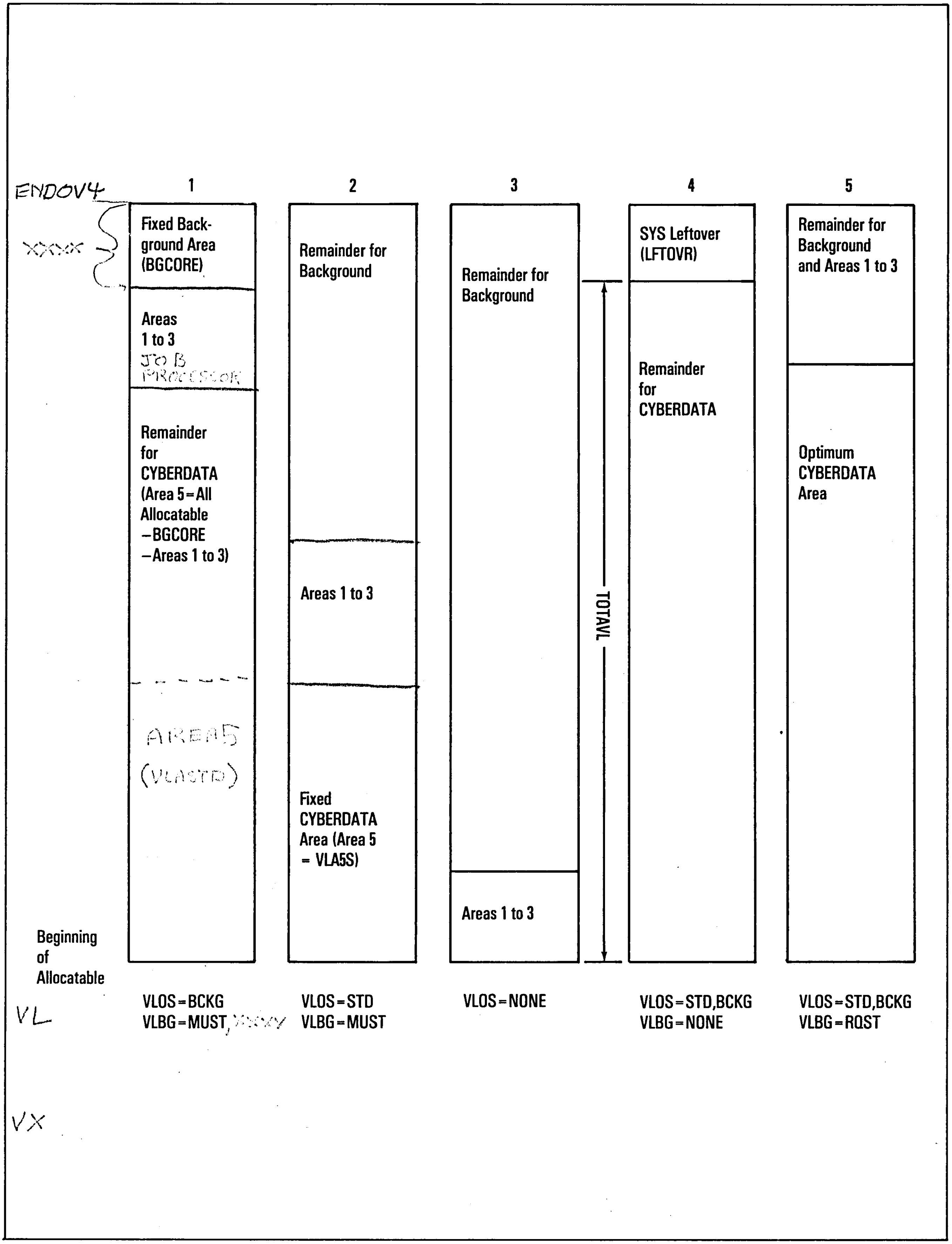


Figure C-3. Examples of Partitioning Allocatable Memory

DISK DATA FILES

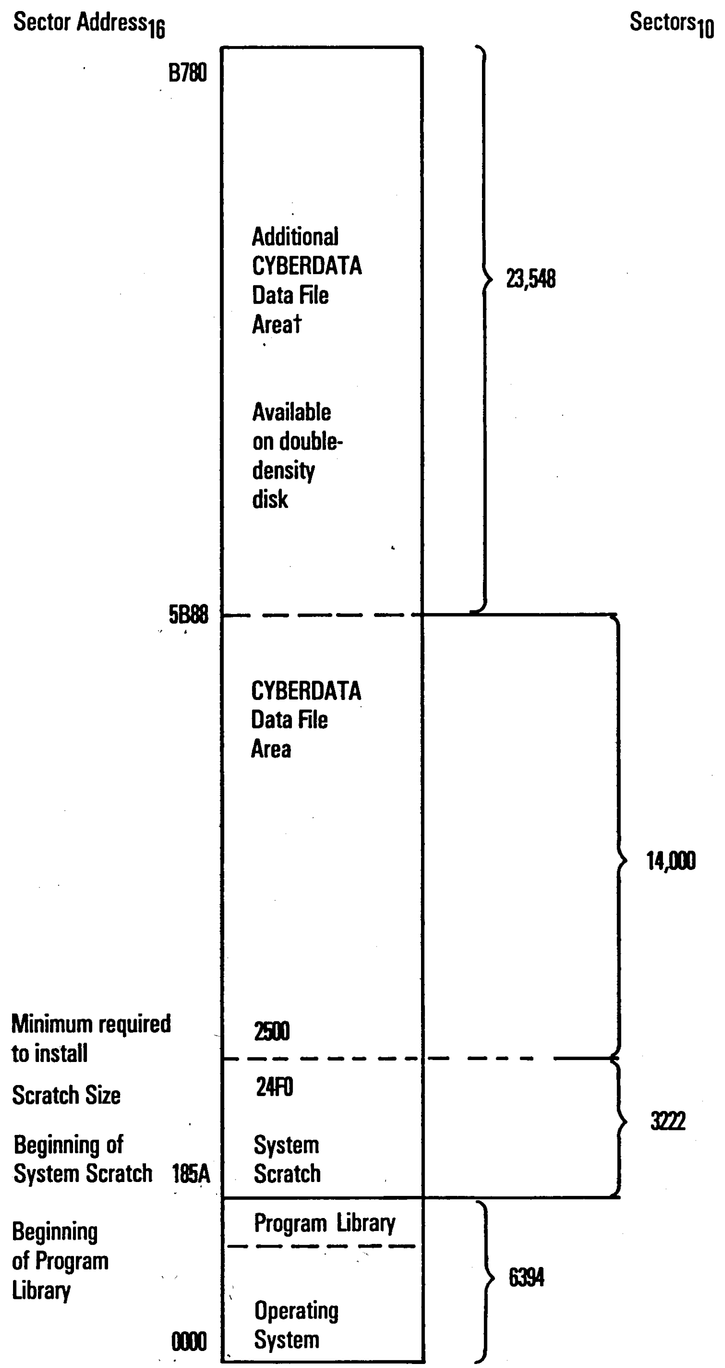
The size of the CYBERDATA data file area on the system disk is determined at installation time. To change the size, the system must be reinstalled.

To add additional disks to the system, SYSDAT must be changed and reassembled and the system must be reinstalled. The SYSDAT changes are to the CYBERDATA system equates, which are in the SYSDAT listings under conditional assembly options. Disks 1, 2, and 3 are added, if needed, by changing the equate cards for DISK1, DISK2, and/or DISK3 from NONE to SINGLE (single density) or DUAL (dual density).

The first 8800_{10} sectors are reserved for the CYBERDATA operating system modules and scratch which is required during installation. This area is not available for data. The remaining area is available for data as follows:

Disk	Sectors for Data	Character Capacity Left
856-2 Single Density	$14{,}624_{10}$	$2{,}807{,}808_{10}$
856-4 Dual Density	$38{,}048_{10}$	$7{,}305{,}216_{10}$

Refer to figure C-4 for the CYBERDATA mass memory (system disk). Figure C-5 shows additional disk mass memory.



†1. Additional disks have a total area available for data (46,970$_{10}$ sectors).

2. Sector 0 and 1 of the first track in a batch contains the table image and other track addresses in batch.

3. If a two-disk system is used, the next batch is assigned to the second disk.

4. Batch limit is 90 tracks or 250,000 characters per batch.

5. Sectors 0 through 5B88 represent the 856-2 single-density disk. Sectors 0 through B780 represent an 856-4 dual-density disk.

6. Scratch size is set at the system installation time.

7. Each record of a batch requires five additional words for linkage.

Figure C-4. COS Mass Memory (System Disk)

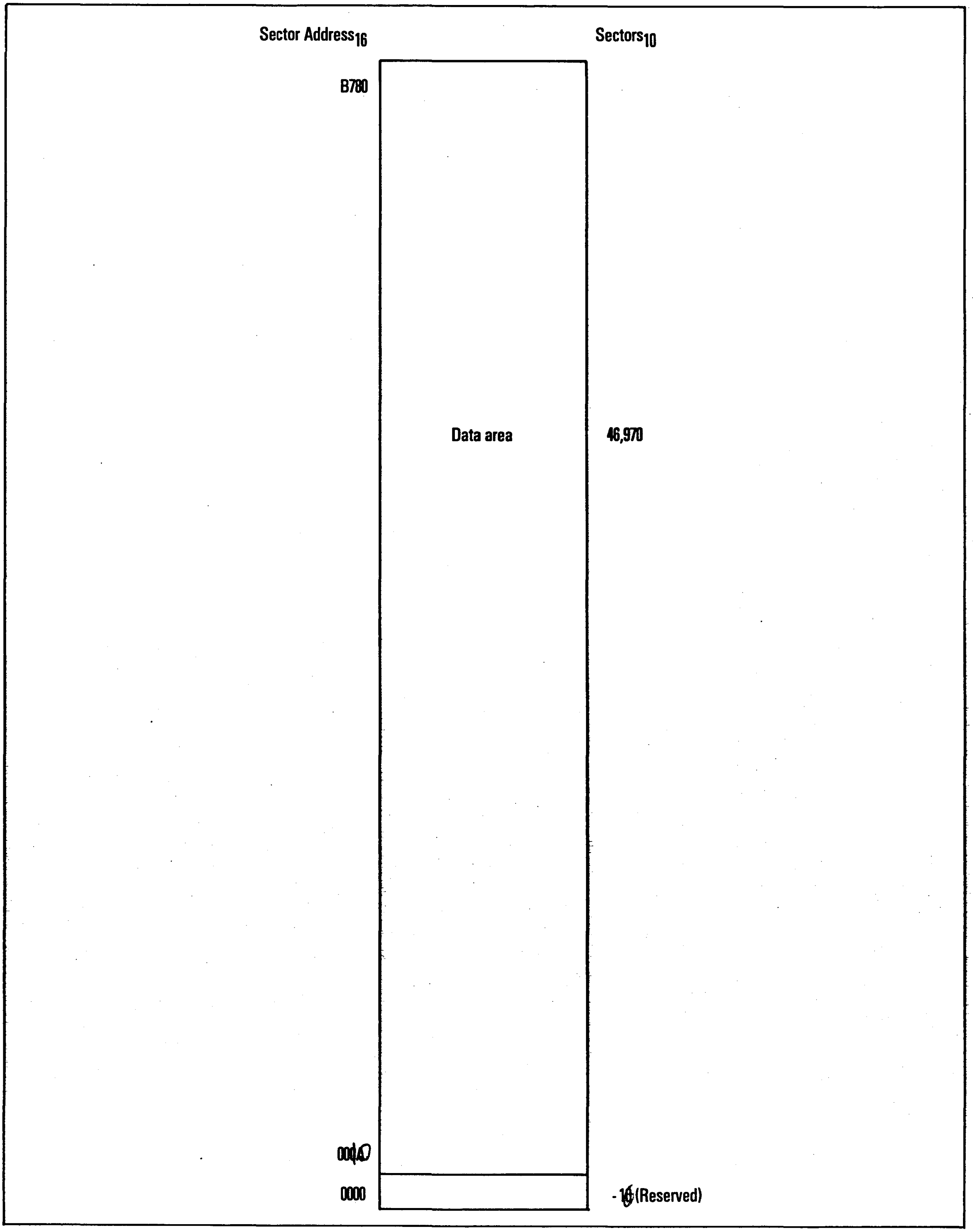


Figure C-5. COS Mass Memory With Additional Disks

APPENDIX D

CYBERDATA STANDARD CODES

CYBERDATA Standard Codes

GENERAL

70

The CYBERDATA codes in table D-1 are represented in hexadecimal.

Table D-1. COS Standard Codes

Symbol	ASCII Code	BCD Code	EBCDIC Code	Symbol	ASCII Code	BCD Code	EBCDIC Code
Space	20	10	40	8	38	08	F8
!	21	2A	5A	9	39	09	F9
"	22	0F	7F	:	3A	0D	7A
#	23	3F	7B	;	3B	2E	5E
$	24	2B	5B	<	3C	3E	4C
%	25	1D	6C	=	3D	0B	7E
&	26	1D	50	>	3E	0E	6E
' (apostrophe)	27	0C	7D	?	3F	3A	6F
(	28	1C	4D	@	40	1F	7C
)	29	3C	5D	A	41	31	C1
*	2A	2C	5C	B	42	32	C2
+	2B	30	4E	C	43	33	C3
, (comma)	2C	1B	6B	D	44	34	C4
-	2D	20	60	E	45	35	C5
.	2E	3B	4B	F	46	36	C6
/	2F	11	61	G	47	37	C7
0	30	0A	F0	H	48	38	C8
1	31	01	F1	I	49	39	C9
2	32	02	F2	J	4A	21	D1
3	33	03	F3	K	4B	22	D2
4	34	04	F4	L	4C	23	D3
5	35	05	F5	M	4D	24	D4
6	36	06	F6	N	4E	25	D5
7	37	07	F7	O	4F	26	D6

Table D-1. COS Standard Codes (cont)

Symbol	ASCII Code	BCD Code	EBCDIC Code	Symbol	ASCII Code	BCD Code	EBCDIC Code
P	50	27	D7	X	58	17	E7
Q	51	28	D8	Y	59	18	E8
R	52	29	D9	Z	5A	19	E9
S	53	12	E2	[	5B	3D	5F
T	54	13	E3	\	5C	1E	E0
U	55	14	E4	]	5D	2D	4A
V	56	15	E5	∧	5E	2F	4F
W	57	16	E6	—	5F	1A	6D

APPENDIX E

DIAGNOSTIC CODES AND MESSAGES

Diagnostic Codes and Messages

EQUIPMENT MALFUNCTION CODES

When a system I/O device driver has detected an error, the alternate device handler is called. The alternate device handler prints the following diagnostic message on the standard comment device if no alternate device is defined:

```
LU,nn FAILED xx
    ACTION
```

Where:

nn—number of the logical unit which has failed
xx—failure code

Respond to the error by typing one of the following:

- RP—to repeat the request
- CU—to report the error to the requesting program; the device is allowed to continue processing requests.
- CD—to cause any future programs calling the device to be informed of the failure by their completion addresses; the error is reported to the calling program, and the device is marked down. no subsequent attempt is made to operate this device.
- DU—to activate CU and suspend job processing; if job processing is not in progress; this action is not taken, and ACTION is retyped. Another option may be selected.
- DD—to activate CD and suspend job processing; if job processing is not in progress, this action is not taken, and ACTION is retyped. Another option may be selected.

When the system initializer device detects an I/O failure, the following message is printed:

```
L,nn FAILED xx (yyyy)
ACTION
```

Where:

nn—initializer logical unit which has failed

xx—failure code

yyyy—last hardware status of the failed device

The error response is one of the following two entries:

- RP—to repeat the request
- CU—to complete the failed operation and cause the initializer to return to the comment device for a subsequent control statement entry

DEVICE FAILURE CODES

The device failure codes for the system or initializer driver are given in table E-1.

Table E-1. Device Failure Codes

Error Code	Error	Description
0	Time-out error	This indicates a failure to interrupt within the allotted time (requires TIMER package) • Teletype 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 is required.

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
1	Lost data	Data was not transferred out of the read register before the next data word appeared. • 1711/1713 Teletypewriter Retype the statement. • Magnetic tape Use the CU option to continue without processing the lost record or abort the read option.
2	Alarm	This indicates the presence of an abnormal condition. • Line printer Paper is out, paper is torn, a fuse alarm is sounded, or interlock is open. Correct the problem, and use the RP option. • 1728-430 Card Reader Interlock is open, or the chip box is full. Correct the problem, and take the RP option. • Pseudo tape This is a failure to fulfill a request due to mass-storage device failure. • COSY driver The first record is not a CSY/ control record.
3	Parity error	• 1711/1713 Teletypewriter Attempt recovery by retyping the command. • Magnetic tape The tape is positioned after the bad record. Either take the CU option to continue processing (the bad record is ignored), or abort the operation. • COSY driver The last record was not an END/ record.
4	Checksum error	(FREAD binary) The sum of the header word and data in a record did not balance to zero when added to the checksum word. • Card readers Holes are not cleanly punched. Check cards for tears between holes. If the cards are all right, attempt recovery; otherwise, perform the following operations:

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
		1. Remove the cards from the input hopper. 2. Single-cycle the card in the transport area to the output stacker. 3. Take the last two cards in the output hopper, and put them into the input hopper ahead of the unread cards. If this is a multicard record, reread all cards in the record. 4. Ready the card reader. 5. Take the RP option. • COSY driver There was no end-of-file mark following the END/ record.
5	Internal reject	The I/O device did not send a reply to the computer within the allotted time. • The computer cannot communicate with the device. Check the hardware address switch and the POWER ON switch. The RP option may be used if the problem has been corrected. • COSY driver A read on the write unit or a write on the read unit occurs before the end-of-deck mark was encountered.
6	External reject	The I/O device has replied to the computer that it is not ready to perform the specified request. • The device is busy or not ready. If the device is not busy, check the ready switch. Attempt to continue by typing RP. • COSY driver Motion request on read unit is encountered after end-of-deck marker.
7	Compare	This is a hardware problem. A compare error occurs when a faulty signal is detected in the area of the punch solenoid and the echo amplifier circuits during an echo check. • 1728-430 Card Reader Remove, and discard the last card punched. Ready the device, and type RP. • 1729-3 Card Reader Attempt recovery as for card checksum error (see error code 4).

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
	Preread	A preread error occurs if all read amplifiers are not off during a dark check. • 1728-430 Card Reader Remove, and discard the last card punched. Ready the device, and type RP. • 1729-3 Card Reader Attempt recovery as for card checksum error (see error code 4).
8	Illegal Hollerith punch	This error occurs when the card reader encounters a punch sequence which does not comply with the Hollerith-to-ASCII conversion table being used by the driver. • Software recovery allows the user to locate the illegal punch by setting an ASCII? in the buffer word for the bad column. Select the reply option to continue or to abort the job and correct the mispunched cards.
9	Sequence error	Cards in a record are not in sequential order. • Abort for read operation and restore sequential order to the record.
10	Nonnegative record length	The first word of a formatted binary record is the complement of the number of records in the record. The word may be a negative number, indicating that the card read was not the first card of the record. • Attempt recovery by using the procedure for checksum error (see error code 4).
11	Read/write mode change	This error indicates a switch from read or write mode. • 1728-430 Card Reader This message is issued only as a warning to the operator. If mode switch is allowable, repeat the request using the RP option.
12	7/9 punch error	The error occurs if a 7/9 punch in column 1 is read when an FREAD ASCII request is specified. • Card reader 1. If column 1 is a 7/9 punch, there is no recovery..Abort operation request is the wrong mode. 2. If column 1 was misread, read card as for checksum error (see error code 4).

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
13	No write ring	An attempt was made to write on magnetic tape without write being enabled. • Insert write ring, and use RP option. • Pseudo tape Attempt to write on the file which was opened to read only.
14	Not ready	Ready the device, and use the RP option.
15	Not used	None
16	Controller seek error	The controller seek error occurs when the controller has failed to obtain the file address that was selected during a read, write, compare, or checkword operation. This is usually an indication of a positioning error.
17	Drive seek error	A drive seek error occurs when the drive unit detects that the cylinder positioner moved beyond the legal limits of the device during a load address, write, read, compare, checkword check, or write address function.
18	Address	This error occurs when an illegal file address obtained from the computer is detected or when the controller has advanced beyond the limits of file storage.
19	Protect fault	The protect fault occurs when an unprotected controller operation attempt is made to write in a protected core location.
20	Checkword error	The checkword error occurs when the controller logic detects an incorrect checkword in data read from file storage during a read, compare, or checkword operation.
21	Not used	None
22	Card output stacker full	Card readers Empty output hopper, and take RP option.
24	Card feed failure	The read ready station does not contain a card after a feed cycle has occurred, and the input hopper is not empty. • Card readers Card feed failure error can occur as a result of warped or damaged cards. If the card reader can be made ready, take the RP option.
25	Card jam	A card transport problem has occurred. It is possible for a card jam to occur in any one or more of four read stations in the 1728 Card Reader. *CAUTION* Do not attempt to single-cycle the machine. Damage to the card transport or punch head may result. Call customer engineering to aid in clearing the jam.

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
		• Jam while reading 1. Examine the transport area. 2. Remove all cards that have completely passed under the read station. 3. The cards that have not completely passed the read station have not been read. Put these cards back into the hopper. Ready the card reader, and repeat the request via the RP option. The cards must be recycled in proper sequence. 4. If the procedure results in failure, abort the read.
26		Not enough file space is available for this request to pseudo tape driver.
27	Not used	None
28	Reserved	None
29	Read error	A read error occurred in reading the mass storage resident driver.
30	Reserved	None
31	Short record	Attempt was made to write a record with a length less than the standard noise-record length.
32	Not used	None
33	Reserved	None
34	Data interrupt	A data interrupt occurred after reading 80 columns. • Card readers This error indicates a hardware failure, possibly due to improper card travel. Reread the card (see the recovery procedure for error code 4).
35	End of operation	An end-of-operation interrupt occurred prior to reading 80 columns. • Card readers Continuous failures may indicate card slippage in feeding. Reread the card as for error code 4.
36	Reserved	None
37	Wrong address	Buffered data channel is using the first-word address other than the address sent by a buffered driver.
40	Repeated the request due to an error	The driver is attempting recovery.
41	Incomplete request	The request was not successfully completed. The driver attempted to repeat the request the maximum number of times.

Table E-1. Device Failure Codes (cont)

Error Code	Error	Description
42	Reserved	None
43	Incomplete directory call of overlay read request	This is due to an irrecoverable error.
44	Guarded address	Error is on write.
45	Reserved	None
46	External reject	Error occurs on output.
47	External reject	The error occurred on input.
48	Controller address error	The controller address status is not the expected value.
49	Drive address error	The drive address status is not the expected value.
50	No ID	This is an ID abort; no ID burst (1732-2).
51	Illegal density	An attempt to select illegal density (1732-2) was made.
52	Reserved	None
53	Reserved	None
54	Reserved	None
55	Reserved	None
56	Mass memory buffer expired	No more buffer space is available (software buffer driver).
57	Buffer transfer error	Mass memory error on buffer transfer (software buffer driver) occurred.
58	Reserved	None

EQUIPMENT STATUS CODES

The hardware status codes listed in tables E-2 to E-8 may appear in the A register if the STATUS request is used. These codes also appear in the engineering file printout.

Table E-2. 1711/1712/1713 Teletypewriter Status Codes

Register A Bits	Status	Description
0=1	Ready	The teletypewriter power switch is in the ON-LINE position; power is applied to the teletypewriter.
1=1	Busy	If the controller is in Read mode, it is in the process of receiving a character from the teletypewriter, or the holding register contains data for transfer to the computer. The busy status drops when the data transfer to the computer is completed, if data has not been lost in the meantime.

Table E-2. 1711/1712/1713 Teletypewriter Status Codes (cont)

Register A Bits	Status	Description
		If the controller is in Write mode, the data register contains data and is in the process of transferring it to the teletypewriter. Busy status drops when the transfer is complete. In either mode, the teletypewriter mode control relays are in the process of switching from one mode to another.
2=1	Interrupt	An interrupt condition exists in the controller.
3=1	Data	If the controller is in Read mode, the holding register contains data for transfer to the computer. The data status drops when the read is completed. One character (located in the lower 7 bits of the A register) is transmitted at a time. If the controller is in Write mode, it is ready to accept another write from the computer. The data status drops when the write is completed.
4=1	End of operation	The clutch in the teletypewriter is disengaged. A change of controller mode may be accomplished at this time. This status is equivalent to a not-busy status.
5=1	Alarm	The ready status is a 0 or the lost data status is a 1. The alarm status drops when the condition it caused is corrected or when the interrupt request is cleared.
6=1	Lost data	The holding register contained data for transfer to the computer, and the teletypewriter began to send a new character sequence. The lost-data status may be cleared by a clear-controller function or a select-write-mode function after the teletypewriter is stopped and the character in the holding register is read or when the interrupt request is cleared.
7	Not used	None
8	Not used	None
9=1	Read mode	The controller is conditioned for input operations.
10=1	Motor on (ready)	This is identical to a ready status; the teletypewriter is turned on.
11=1	End of file	This is used for the 1713 paper tape reader only.

Table E-3. 1728-430 Card Reader/Punch Controller Status Codes

Register A Bits	Status	Description
0=1	Ready	The card reader is operational.
1=1	Busy	The controller is busy whenever a card is being entered into the buffer memory.
2=1	Interrupt	Interrupt status is available if one or more of the selected interrupts has occurred. Other bits must be monitored to determine the condition causing the interrupt.

Table E-3. 1728-430 Card Reader/Punch Controller Status Codes (cont)

Register A Bits	Status	Description
3=1	Data	The card reader is ready to transfer data to the computer.
4=1	End of operation	Last card column was read, or a reload memory function was sent.
5=1	Alarm	Card reader has one or more of the following alarm conditions: • Compare or preread error • Stacker full or jammed • Fail to feed • Separator card transferred to memory • AUTO/MAN switch in manual position
6=1	Lost data	Indicates data was not transferred out of the holding register before the next column that was being read appeared. The status drops when a clear (0=1) is sent to the controller. NOTE When lost data occurs, no further transfers occur from that card. An end-of-operation status is generated.
7=1	Protected	The controller recognizes only the I/O instructions with the protect bit present. Bit 7 is 1 when the PROTECT switch is in the PROTECT position.
8=1	Error	A preread or compare error occurred.
9=1	Motion failure	This indicates that during a card cycle, the transport of the card failed.
10=1	End of file	The end-of-file condition is caused by an empty input tray, unloaded buffer memory, or the END-OF-FILE switch being on. When the input tray does not contain the last card of a file, the switch should be off to inhibit the status.
11=1	End of file	This indicates an end-of-file card has been read. This bit is set by the driver.
12=1	Chip box error	The chip box is full.

Table E-4. 1729-3 Card Reader/Controller Status Codes

Register A Bits	Status	Description
0=1	Ready	The card reader is operational.
1=1	Busy	The card reader is busy.

Table E-4. 1729-3 Card Reader /Controller Status Codes (cont)

Register A Bits	Status	Description
2=1	Interrupt	This indicates the interrupt response generated by the card reader. Other status bits must be monitored to determine the cause of the interrupt.
3=1	Data	This indicates that data transfer may occur. For the reader data, the data hold register contains information ready for transfer to the computer.
4=1	End of operation	End of operation indicates the card reader has completed operation.
5=1	Alarm	An alarm status indicates the presence of an alarm condition.
6=1	Lost data	This indicates that data was not transferred out of the holding register before the next column that is being read appeared. The status drops when a clear (0=1) is sent to the controller. NOTE When lost data occurs, no further transfers occur from that card, and an end-of-operation status is generated.
7=1	Protected	This indicates the protect switch on the card is in the PROTECT position. When it is in this position, the card reader only accepts instructions with a 1 on the program protect line. All other instructions are rejected. A protected instruction is used with either a protected or unprotected card reader.
8=0	Not used	None
9=1	Not ready	This is always the inverse of bit 0.
10=1	END-OF-FILE switch	This status indicates the END-OF-FILE switch is on.
11=1	End-of-file card	This indicates an end-of-file card has been read. The bit is set by the driver.

Table E-5. 1732-2/615-73/615-93 Magnetic Tape Controller Status Codes

Register A Bits	Status	Description
0=1	Ready	The tape unit is connected and ready.
1=1	Busy	The equipment is busy.
2=1	PE warning	None
3=1	PE lost data	None
4=1	End of operation	Data transfer is completed.
5=1	Alarm	An error condition occurred (see other error status lists).
6=1	Lost data	None

Table E-5. 1732-2/615-73/615-93 Magnetic Tape Controller Status Codes (cont)

Register A Bits	Status	Description
7=1	PE transport	The controller is connected to a phase-encoding transport.
8=1	Parity error	A parity error has been detected.
9=1	End of tape	The end-of-tape marker has been sensed.
10=1	Loadpoint	None
11=1	File mark	A file mark or tape mark is sensed.
12=1	556 bpi	The tape is set to 556 bpi.
13=1	800 bpi	The tape is set to 800 bpi.
14=1	Seven-track	None
15=1	Write enable	A write enable ring is present.

Table E-6. 1733-2/856-2/856-4 Cartridge Disk Controller Status Codes

Register A Bits	Status	Description
0=1	Ready	The ready status bit indicates that the drive is available and is ready to operate. The drive becomes not ready for the following reasons: • Disk pack not in drive unit • Disk drive motor not up to operating speed • Read/write heads not in operating position • A fault condition develops in the drive The status condition is affected by the operating program only if it selects a nonexisting device or a device which is not ready. Normally the ready status bit indicates that manual intervention is required at the selected drive unit.
1=1	Busy	The busy status bit indicates that the controller and/or the drive unit is presently involved in the performance of an operation. The bit is set by the acceptance of a load address, write, read, compare, checkword check, or write address function. The busy status bit is cleared when the controller and/or drive unit has completed its operation or when an abnormal condition is detected which aborts the operation. Once initiated, the computer cannot clear the busy condition.
2=1	Interrupt	The interrupt status bit indicates that a selected interrupt condition has occurred. The bit is cleared by the acceptance of any output function.

Table E-6. 1733-2/856-2/856-4 Cartridge Disk Controller Status Codes (cont)

Register A Bits	Status	Description
3=1	On cylinder	The on cylinder status bit is set when the drive positioner is on cylinder. The bit is cleared if the drive unit is presently positioning or if a seek error is detected.
4=1	End of operation	The end-of-operation status bit is set whenever the controller portion of an operation is complete. The busy status bit may remain set if the selected unit is positioning. The bit is cleared by any output function.
5=1	Alarm	The alarm status bit indicates that one of the following abnormal conditions occurred: • Not ready • Checkword error • Lost data • Seek error • Address error • Storage parity error • Protect fault Any output function clears the bit. The not ready condition can be changed by manual intervention.
6=1	No compare	The data received from the computer core storage does not compare with the data read from file storage during a compare operation. The bit is cleared by any output function.
7=1	Protected	The controller is presently reserved for or being operated on by protected computer instructions, or the drive unit is protected and may only be accessed by protected computer instructions. If the controller is reserved or being operated on by a protected instruction, it can be cleared by a protected director function which has the release bit set in register A. If the drive unit is protected by the PROTECT switch on the operator's panel, it can be cleared by changing the PROTECT switch to its off position (down) or by deselecting the unit with a director function which has the proper protect code set in register A.
8=1	Checkword error	The controller logic has detected an incorrect checkword in data read from file storage during a read, compare, or checkword check operation. The bit is cleared by any output function.

Table E-6. 1733-2/856-2/856-4 Cartridge Disk Controller Status Codes (cont)

Register A Bits	Status	Description
9=1	Lost data	The computers direct-access bus has not been able to keep up to the file data transfer rate during a write, read, or compare operation. The bit is cleared by any output function.
10=1	Address error	The controller has detected an illegal file address receivcd from the computer, or the controller has advanced the file address beyond the limits of file storage. The bit is cleared by any output function.
11=1	Controller seek error	The controller has been unable to obtain the file address selected during a write, read, compare, or checkword check operation. This error usually indicates a positioning error. The error can be corrected by doing a status of the drive cylinder, comparing this with the cylinder register (to find out how many tracks and in what direction the positioning error is from the selected file address). The first load address function which follows a controller seek error moves the CDD positioner without changing the cylinder register, and can therefore, correct the positioning error. The bit is cleared by any function which sets the busy status.
12	Drive type	An 856-2 Drive is connected.
13=1	Storage parity error	The controller has received a parity error signal from the direct-storage bus while receiving data or control information. If the error is detected on control information transfer, the operation ends immediately. If the error is detected during a data transfer, the operation ends at the end of the sector which is being operated on. The bit is cleared by any output function.
14=1	Protect fault	An unprotected controller operation attempts to read or write in a protected computer storage area. If the error is detected while control information is being received from storage, the operation ends immediately. If the error is detected while data is being transferred to or from storage, the operation ends at the end of the sector which is being operated on. The bit is cleared by any output function.
15=1	Drive seek	The drive unit has detected that the cylinder positioner has moved beyond the legal limits of the device (below cylinder 0 or above maximum cylinder) during a load address, write, read, compare, checkword check, or write address function. The bit is cleared by any function that sets the busy status.

Table E-7. 1742-30/120 Line Printer Status Codes

Register A Bits	Status	Description
0=1	Ready	The printer is operational.
1=1	Busy	The printer is busy during the transfer and storage of each character. It is also busy after the initiation of a print cycle and remains busy until the content of memory is printed. Paper motion also activates the printer. Transfer of data to memory, however, is allowed.
2=1	Interrupt	The printer indicates an interrupt response. The other status bits determine the cause of the interrupt.
3=1	Data	The printer is ready to receive data. If an interrupt on data has been selected, data status also indicates that the interrupt has occurred.
4=1	EOP	The printer has completed an operation. If the bit is 1, no operation is in progress.
5=1	Alarm	The printer has an alarm condition.
6=1	Error	A parity synchronization or compare error has occurred.
7=1	Protected	The PROTECT switch on the printer is in the protected position. In this position, the printer accepts only those instructions with a 1 on the program protect line. All other instructions are rejected. A protected instruction can be used with either a protected or unprotected printer.
8=1	Load image	The image memory of the line printer must be loaded (1742-120 only). The next 288 characters will be sent to the image memory.

Table E-8. Pseudo Tape Status Codes

Status Bits	Status	Description
0=1	Ready	Always set
1=1	Busy	Always set
2	Not used	None
3=1	Data	Set on completion of read or write
4=1	End of operation	Set at end-of-operation
5=1	Alarm	Set on malfunction/error
6	Not used	None
7	Not used	None
8	Not used	None
9=1	End of tape	The last existing record on the file has been accessed
10=1	Loadpoint	The internal pointers are pointing to the beginning of the file

Table E-8. Pseudo Tape Status Codes (cont)

Status Bits	Status	Description
11-1	File mark	A pseudo file mark has been sensed
12	Not used	None
13	Not used	None
14-1	800 bpi	Always set
15-1	Write enable	The file may be written on

SYSTEM INITIALIZER CODES

Table E-9 defines the system initializer error codes.

Table E-9. System Initializer Error Codes

Message	Significance
ERROR 1	Asterisk initiator missing
ERROR 2	Number appears in name field
ERROR 3	Illegal control statement
ERROR 4	Input mode illegal
ERROR 5	Statement other than *Y or *YM previously entered
ERROR 6	Statement other than *Y previously entered
ERROR 7	*Y not entered prior to first *L
ERROR 8	Name appears in number field
ERROR 9	Illegal hexadecimal core relocation field
ERROR A	Illegal mass storage sector number
ERROR B	Error return from loader module
ERROR C	Unpatched external at conclusion of an *M load
ERROR D	Unpatched external at conclusion of an *L or *LP load
ERROR E	Field terminator invalid
ERROR F	More than 120 characters in control statement
ERROR 10	Ordinal name without ordinal number
ERROR 11	Doubly defined entry point
ERROR 12	Invalid ordinal number

Table E-9. System Initializer Error Codes (cont)

Message	Significance
ERROR 13	Loader control statement out of order; correct order is L, LP, M, MP
ERROR 14	Data declared during an *M load but not by first segment; initialization restarted
ERROR 15	Not used
ERROR 16	Irrecoverable mass storage input/output error
ERROR 17	Irrecoverable loader error; last program loaded was ignored
ERROR 18	Not used
ERROR 19	Not used
ERROR 20	*S,ENDOV4,hhhh not defined before first *L
ERROR 21	*S,MSIZV4,hhhh not defined before first *LP or *MP
ERROR 22	Attempt to load part 1 core resident into nonexistent memory
ERROR 23	The name used in the second field of an *M control statement was not previously defined as an entry point
ERROR 24	The entry point, SECTOR, was not defined at the start of initialization and is not available to the initializer
ERROR 25	Illegal partition number in first field of an *MP statement or illegal number of partitions in second field of statement
ERROR 26	Attempt made to load an *MP program when no partitioned core table exists in SYSDAT

SYSTEM INITIALIZER LOADER ERRORS

The system initializer loader errors are defined in table E-10.

Table E-10. System Initializer Loader Errors

Message	Significance
LOADER ERROR 1	Unrecognizable input
LOADER ERROR 2	Mass storage overflow
LOADER ERROR 3	Out of order input block
LOADER ERROR 4	Illegal data or common declaration

Table E-10. System Initializer Loader Errors (cont)

Message	Significance
LOADER ERROR 5	Core overflow
LOADER ERROR 6	Overflow of entry-point table
LOADER ERROR 7	Data block overflow
LOADER ERROR 8	Duplicate entry point
LOADER ERROR 9	15/16-bit arithmetic error
LOADER ERROR 10	Unpatched externals
LOADER ERROR 11	Insufficient core for both SYSDAT and paging
LOADER ERROR 12	Illegal page number used
LOADER ERROR 13	Undefined transfer address
LOADER ERROR 14	Invalid function for loader
LOADER ERROR 15	Link table overflow
LOADER ERROR 16	External table overflow
LOADER ERROR 17	Entry point absolutized to $7FFF

SYSTEM INITIALIZER DISK ERRORS

The disk errors for the system initializer are given in table E-11.

Table E-11. System Initializer Disk Errors

Message	Significance
DISK ERROR	Address tag write sequence attempted but internal/external reject found
DISK FAILURE xx	Surface test operation caused error xx; refer to device error codes to interpret xx
DISK COMPARE ERROR SECT aaaa WORD bbbb is cccc SB dddd	Surface test pattern error on sector aaaa at word bbbb; only one error will be listed per sector. Data read was cccc but it should be dddd

GENERAL SYSTEM ERROR MESSAGES

Table E-12 lists general system errors.

Table E-12. General System Error Messages

Message	Significance
ALT,no.	This informs the operator an alternate device number has been assigned.
B01,statement	A statement or parameters are unintelligible for the breakpoint program.
B02,address	The specified hexadecimal address cannot be processed by the breakpoint program because it is protected.
B03,address	The breakpoint limit is exceeded. The specified hexadecimal address is the last breakpoint processed.
CHECK TAPE UNIT	Restart the disk-to-tape program.
CHECKING FILE - ERRORS	Errors were detected in the file manager files when checked after autoload.
DATE/TIME ENTRY	Reenter COS date/time.
DB FORMAT INCORRECT	Some part of the remaining portion of request is incorrect for ODEBUG.
DB INVALID REQUEST	The mnemonic does not agree with any known mnemonic for ODEBUG.
DB I/O ERROR	The monitor request returned with the error bit set for ODEBUG.
DB LHO/LHC ERROR	Data written on mass storage does not match LHO/LHC input for ODEBUG.
DB NO CORE AVAILABLE	No allocatable core is available for ODEBUG.
EF STACK OVERFLOW	There is currently no space in the engineering file stack to record this device failure.
EFSTOR LU ERROR	An illegal logical unit has been passed to the engineering file which is outside the range 1 to 99.
EFSTOR MASS MEMORY FAILURE	An error occurred in updating the engineering file on mass memory.
ILLEGAL PARAMETERS SPECIFIED	Disk to tape has detected a nonhexadecimal character for equipment code. Respecify equipment codes.
L,no. FAILED code ACTION	The number of the failed device appears when a driver cannot recover from an error. Type RP to repeat the request; or, type CU to report the error condition to the requesting program and continue. Press RETURN.

Table E-12. General System Error Messages (cont)

Message	Significance
L,no. FAILED code status	This informs operator of device failure in the initializer: no.–logical unit of failed device code–indicates cause of failure (see equipment malfunction codes for the code description) status–hardware status
LU no. DOWN	If a device is marked down and yet is requested by a program, and this device contains no alternate, this message is typed on the comment device only the first time it is requested after being downed. The completion address is always scheduled with error. The requesting program should not continually request downed units.
MI INPUT ERROR	A statement presented to the manual interrupt processor is unrecognizable or the requested program is not supplied.

MM ERR xx
LU=nn
T=hhmm:ss
S=ssss
	xx = Error number
	nn = Logical unit
	hhmm = Hours/minutes
	ssss = Hardware status

OV	This shows an overflow of volatile storage. The message appears on the output comment device. No recovery is possible.
PARITY,address	A memory parity error occurred at the specified hexadecimal location. The message appears on the output comment device. No recovery is possible.
SET PROGRAM PROTECT	The system is waiting for the PROGRAM PROTECT switch to be set.
TIMER REJECT	The timer start up was rejected (SPACE or MIPRO).
TROUBLE WITH THE DISK	Restart the disk-to-tape program.

JOB PROCESSOR ERROR CODES

Table E-13 defines the job processor error codes.

Table E-13. Job Processor Error Codes

Message	Significance
JOB ABORTED	The current batch job has abnormally terminated. If the job card included a job name, that name replaces JOB.

Table E-13. Job Processor Error Codes (cont)

Message	Significance
JP,yyyyyy	yyyyyy is the last program library program that was executed before the job terminated.
JP01,hhhh	This is a program protect violation. hhhh is the current contents of the P register. The message is on the standard comments device.
JP02,address	An illegal request or parameters occurred at the specified hexadecimal address. The message is on the standard comments device.
JP03,statement	An unintelligible control statement is output with the diagnostic. The message is on the standard comments device.
JP04,statement	There are illegal or unintelligible parameters in the control statement. The message is on the standard comments device.
JP05	The statement that was entered after manual interrupt is illegal. The message is on the standard comments device.
JP06	A threadable request was made at level 1 when no protect processor stack space was available, or an unprotected threaded request was made at level 1. The message is on the standard comments device.
JP07	An unprotected program tried to access the protected device. The message is on the standard comments device.
JP08	An attempt was made to access the read-only unit for write or the write-only unit for read, or an attempt to access an unprotected request on a protected unit. The message is on the standard comments device.
JP09	An I/O error occurred while accessing the job processor file directory table.
JP10	An operation was attempted on a file that is not in the file table. Define the file.
JP11	The file name that is being defined already exists for another file. Dump the file table to select a name that was not used previously, or attempt a new define with another name.
JP12	An attempt was made to access a file that has not been opened.
JP13	No files are available for definition. Purge the file table to make available any expired files.
JP14	An attempt to open a previously opened file has been made.
JP15,xxx	The JOB card was not the first control statement in job, or more than one job card was detected in a job. xxx is the control statement in error.

LOADER ERROR CODES

Loader error codes are defined in table E-14.

Table E-14. Loader Error Codes

Message	Significance	Message	Significance
E1	Irrecoverable input/output error; terminates load	E11	Minimum amount of core not available for load; at least 195 words plus the length of the loader must be available; terminates load
E2	Overflow of entry external table reservation on mass storage; terminates load	E12	Overflow of command sequence storage reservation on mass storage; terminates load
E3	Illegal or out-of-order input block; terminates load	E13	Undefined or missing transfer address; this code is not given if the loading operation is part of system initialization; occurs when loader does not encounter a name for the transfer address or the name encountered is not defined in loader's table as entry-point name; loading is terminated
E4	Incorrect common or data block storage reservation; occurs if the largest common storage declaration is not on first NAM block to declare common or data storage or occurs if when protected common or data was being used, the NAM block declared a reservation longer that protected common or data; terminates load	E14	Loader request operation code word illegal; terminates load
E5	Program longer than area or partitions allotted to it; terminates load	E15	Overflow of loader table used to store relocatable addresses that have been absolutized to $7FFF; terminates load
E6	Attempt to load information in protected core; terminates load	E16	Entry-point name not in loader table; operator must type in correct entry-point name
E7	Attempt to begin data storage beyond assigned block; terminates load	E17	Informative diagnostic; relocatable entry point has been absolutized to location $7FFF; if any program in system is testing for an entry-point value of $7FFF to indicate that this entry point is not present, the test is not valid
E8	Duplicate entry point		
E9	High-order bit of a relocatable address is set or negative relocation has been encountered during part 1 load; terminates load		
E10	Unpatched externals; external name is printed following the diagnostic; when all unpatched externals have been printed, the operator may terminate the job by typing in an *T CR or continue execution by typing in an *CR; core-resident entry point tables may also be linked by typing in an *E		

LIBEDT ERROR CODES

The error codes assigned to LIBEDT are defined in table E-15.

Table E-15. LIBEDT Error Codes

Message	Significance	Message	Significance
L01	More than six characters in a parameter name are presented to the library editing program	L03	An improper system directory ordinal was presented to the library editing program.
L02	More than six digits in a number are presented to the library editing program.	L04	An invalid control statement was presented to the library editing program.

Table E-15. LIBEDT Error Codes (cont)

Message	Significance	Message	Significance
L05	An illegal field delimiter in a control statement was presented to the library editing program.	L14	The program which is being loaded is longer than the size of unprotected core but is not longer that the distance from the start of unprotected core to the top of core.
L06	An illegal field in the control statement was presented to the library editing program, or input/output was attempted on a protected device.	L15	An illegal input block is encountered. The last program stored in the library is not complete.
L07	Errors occurred in loading as a result of a library editing program control statement.	L16	An I/O input error occurred. The last program stored is not complete.
L08	A program to be added to the program library has an entry point which duplicates one already in the directory.	L17	An *L program being installed exceeds the capacity of LIBEDT to input from mass storage.
L09	Standard input failed on the first input record following an *N request.	L18	An attempt was made to load a zero-length program during an *M request.
L10	The operator is deleting a program which is not in the library.	L19	No data-base entry point was specified in the system for use by an *A statement and parameters.
L11	No header record is on file input from mass storage.	L20	An irrecoverable error occurred during loading.
L12	On an *L entry statement, either there was an input error or the first record was not a NAM block.	L21	An attempt was made to write beyond the maximum sector number specified for MAXSEC at initialization.
L13	The common area declared by the program being loaded exceeds available common or system common not specified in system when requested.		

COSY ERRORS

Table E-16 lists COSY errors with the appropriate COSY action for the error.

Table E-16. COSY Errors

Message	Significance	COSY Action
no. ERRORS	This message appears at the end of a COSY job if errors exist. The number specified is the decimal count of errors in the COSY job.	
****COSY Cno****		
01	The first card of the revisions deck is not a DCK/, MRG/, CPY/, or END/ control card.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, CPY/, or END/ card.
02	Illegal parameters occurred on the MRG/ control card.	COSY aborts.
03	The first card from merge input is not a DCK/ control card.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/ or END/ card.

Table E-16. COSY Errors (cont)

Message	Significance	COSY Action
04	An MRG/ control card is in revisions decks.	COSY aborts.
05	Illegal parameters occurred on the DEL/, INS/, or REM/ control card.	COSY reads revisions and lists them with asterisks in columns 1 through 4 until it reads next control card.
06	Sequence numbers are out of order in the revisions set.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads next control card.
07	Two sequence numbers occurred on the INS/ control card.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads next control card.
08	The control card does not follow DCK/ card when revisions are being merged.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads next control card.
09	The first card of the source deck is not a CSY/ or HOL/ control card.	COSY aborts.
10	The requested deck is not on the input library.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card.
11	Decknames on the DCK/ and HOL/ cards do not agree when adding a new deck to COSY library.	COSY aborts.
12	The revision card following the DCK/ card is not a control card.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a control card.
13	The DEL/ or INS/ card contains a sequence number beyond the end of the input deck.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card.
14	An illegal parameter occurred on the DCK/ card.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card.
15	A parameter occurred on the DCK/ card twice.	COSY uses the second parameter.
16	A DCK/ card requests both H and C or H and L on the same unit.	The C or L parameter is ignored. Processing continues.
17	A DCK/ card requests input from a logical unit previously used for output.	COSY reads the revisions and lists them with asterisks in columns 1 through 4 until it reads a DCK/, MRG/, or END/ card.
18	COSY output is requested on a unit previously used for Hollerith output, or Hollerith output is requested on a unit previously used for COSY.	The illegal output request is cleared. Processing continues.
19	The maximum number of output units is exceeded.	Output is cleared. Processing continues.
20	The DCK/ card requests output on a logical unit previously used as input.	The output is removed. Processing continues.
21	The DCK/ card requests C and L output on the same unit.	The L parameter is ignored. Processing continues.

Table E-16. COSY Errors (cont)

Message	Significance	COSY Action
22	The CPY/ control card is not the first card of the revisions deck.	The CPY/ control card is listed with asterisks in the first four columns, and the next control card is read.
23	The CPY/ card was not followed by a CPY/ or END/ card.	COSY aborts.
L,lu,FAILED ec	COSY driver errors output by the alternate device handler. All errors are catastrophic:	For protected requests: Type DU For unprotected requests: Type DU
1	Not assigned	
2	First record read not a CSY/ record	
3	END/ card not last card on COSY input	
4	No end-of-file on COSY input	
5	Read request was made to a logical unit assigned to output, or write request was made to a logical unit assigned to input	
6	MOTION request was made to a logical unit assigned to input/output and no end-of-deck marker was encountered	
REWIND LU no.	This message may appear at various times during a COSY job. The specified number is the decimal logical unit to be rewound.	The operator must enter any value through the standard input comment device after rewinding the unit.

MACRO ASSEMBLER

Table E-17 lists macro assembler errors.

Table E-17. Macro Assembler Errors

Message	Significance
xxxxyy**********	This is the format for pass 1 and 2 error messages: Where: xxxx–four-digit line number yy–two-character error code explained in table
******yy**********	This is the format for pass 3 error messages. If the L option is selected, errors in pass 3 precede the source line on the list output. If L is not selected, error messages are output on the standard comment unit.
ABS BASE ERR	The assembler was loaded at a different location from where it was absolutized.

Table E-17. Macro Assembler Errors (cont)

Message	Significance
**DS	A double-defined symbol occurred; a name is in: • Location field of a machine instruction • ALF, NUM, or ADC pseudo instruction • Address field of an EQU, COM, DATA, EXT, BSS, or BZS pseudo instruction
**EX	This indicates an illegal expression; either: • No forward referencing of some symbolic operands • No relocation of certain expression values • A violation of relocation • Illegal register reference • Symbol other than Q, I, or B is specified
INPUT ERROR	An error was returned by the driver when doing a read.
**LB	A numeric or symbolic label contains an illegal character. The label is ignored.
MASS STORAGE OVERFLOW	There is not enough room for the input image on mass storage.
**MC	This is a macro call error: • Illegal parameter list • No continuation card where one was indicated
**MD	A macro definition error occurred.
**MO	An overflow of the load-and-go area has occurred; affects only the X option.
**NN	There is a missing or misplaced NAM statement.
**OP	An illegal operation code is encountered: • Illegal symbol in operation code field • Illegal operation code terminator
**OV	One of the following occurred: • Numeric constant or operand overflow value is greater than allowed • Operand overflowed

Table E-17. Macro Assembler Errors (cont)

Message	Significance
**PP	An error in the previous pass of the compilation assembly has occurred; the output page immediately preceding the first page of listing for pass 1 or pass 2 error message.
**RL	An illegal relocation is found: • Violation of relocation • Violation of a rule for instructions that requires the expression value to either be absolute or have no forward referencing of symbolic operands
**SQ	A sequence error is indicated; this tags instructions with sequence numbers that are out of order. This is not fatal and is not counted in the number of errors reported at the bottom of the symbol table.
**UD	An undefined symbol in an address expression is found.

UTILITY PROGRAMS

Table E-18 defines the utility program errors.

Table E-18. Utility Program Errors

IOUP	
Message	Significance
END OF TAPE LU nnnn ACTION?	An end-of-tape mark is sensed while data is being written on magnetic tape. The operator must respond with either $RES to resume action from the point of last interruption or $END to terminate the request.
FILE BACKD FILE nnnn FILE BACKD RECS nnnn	The specified unit has been backspaced by nnnn files or records.
FILE SKIPD FILE nnnn FILE SKIPD RECS nnnn	The specified unit has been advanced by nnnn files or records.
FORMAT ERROR	This indicates an invalid control statement; reenter the statement.
IN/OUT ERROR LU nnnn	An error occurred in an input/output operation on logical unit nnnn.
MISMATCH REC nn*32768+nnnnn	The indicated record is not the same on both the data being verified. NOTE The nn and nnnn are defined as follows: nn–00 through 03

Table E-18. Utility Program Errors (cont)

IOUP	
Message	**Significance**
	The quotient is obtained by dividing the total number of records by 32768. If nn is 0, only nnnn will be typed out. nnnnn–0 through 32767 The remainder is obtained by dividing the total number of records by 32768.
UT FORMAT INCORRECT	The request is not correctly formatted. Parameters and/or delimiters are incorrect.
UT INVALID REQUEST	The mnemonic request code is illegal.
DTLP	
Message	**Significance**
ILLEGAL PARAMETERS SPECIFIED	The equipment code which was specified for disk to tape contains a nonhexadecimal character.
DISK ERROR (xxxx)	A disk error has occurred. The last status read was xxxx.
TAPE ERROR (xxxx)	A tape error has occurred. The last status read was xxxx.
LIBILD	
Message	**Significance**
INVALID LU	The logical unit is illegal.
INVALID CLASS CODE	The device is incompatible with the function to be performed.
LAST DECK REJECTED - NOT UNIQUE	This refers to duplicate copies of program. The program identification must be unique.
LAST DECK REJECTED - NO XFR RECORD	The binary program does not have a transfer record.
NAME RECORD NOT 1ST RECORD OF DECK. TYPE 1,CR TO TERMINATE EXECUTION. TYPE 2,CR TO PROCEED TO SUBSEQUENT LIBRARY OR SKELETON. TYPE 3,CR TO CONTINUE ON WITH CURRENT LIBRARY	The expected relocatable binary deck did not have a NAM block format.
XFR RECORD MISSING FOR LAST PGM LISTED. PGM DELETED. TYPE 1,CR TO TERMINATE EXECUTION. TYPE 2,CR TO PROCEED TO SUBSEQUENT LIBRARY OR SKELETON. TYPE 3,CR TO CONTINUE ON WITH CURRENT LIBRARY.	The relocatable binary deck was not terminated with XFR block.
TOO MANY BINARY DECKS LOADED. CHANGE LIMIT AND RECOMPILE.	This library has more programs that LIBILD can process.

Table E-18. Utility Program Errors (cont)

LIBILD	
Message	**Significance**
CHECKSUM ERROR NOTED IN LAST PROGRAM. BAD *DEF RECORD. NO IDENT CHARACTER. BAD *DEF RECORD. IDENT CHAR ALREADY USED. IGNORED.	Previously generated checksum does not compare with the current checksum when the program is read from mass memory.
INVALID DEFINITION RECORD. IGNORED. NO DEFINITIONS WERE SUCCESSFULLY LOADED. TOO MANY DEFINITION SETS. IGNORED. PROGRAM SPECIFIED BY THIS RECORD NOT FOUND. PROGRAM HAVING THIS ID INFO NOT FOUND.	*DEF is not the first record of the definition group.
MORE THAN ONE PGM HAS THIS NAME. (NO ID INFO.)	The first program on the library with this name will be written to installation file.
ILLEGAL CHARACTER STARTS IDENT FIELD.	The ident field must start with a single quote.
ILLEGAL IDENT FIELD. RECORD IGNORED.	The *B record was not terminated by a single quote prior to column 73.
ILLEGAL *B RECORD. RECORD IGNORED.	The name field of *B must be enclosed by single quotes.
NULL PROGRAM NAME. RECORD IGNORED.	The name field consists of two single quotes.
PROGRAM NAME TOO LONG. RECORD IGNORED.	The name on *B contains more than six nonblank characters.
NO DEFINITIONS ARE STORED. RECORD IGNORED.	*USE was encountered but no definitions were made.
INVALID *USE RECORD. IDENT FIELD. RECORD IGNORED. INVALID *USE RECORD. MAX IMBEDDED LEVEL IS SIX. RECORD IGNORED.	No nonblank character was detected prior to column 73.
INVALID *USE RECORD. REQUESTED SET IS IN USE. RECORD IGNORED.	This *USE is infinitely recursive.

SKED	
Message	**Significance**
INVALID COMMAND	The command was not valid for SKED.
ERROR IN COMMAND FORMAT	A comma, argument, etc. has been omitted.
COMMAND NAME NOT UNIQUE	Not enough letters were included to uniquely define command.
LU NOT LEGAL FOR COMMANDS	The device type was not valid for the command requiring.
SKELETON NOT LOADED	The skeleton was not previously loaded prior to operation on it.
RECORD NUMBER IS ZERO	The record number of zero is illegal.
INVALID CHARACTER IN NUMBER	Nondecimal characters were specified in the number argument.
INVALID RECORD NUMBER	The record number is out of range, or the second argument is less than the first argument.

Table E-18. Utility Program Errors (cont)

SKED	
Message	**Significance**
RANGE CONTAINS NUMBER ALREADY DELETED	The record has been referenced which has been deleted.
RECORDS HAVE BEEN PREVIOUSLY DELETED	Range of record numbers of the catalog command includes numbers which have been deleted.
NO INSERTION RECORDS FOUND AT SPECIFIED LU	A device which was defined for insertion records does not contain any.
RECORDS NOT DELETED PLEASE RESEQUENCE SKELETON	An attempt was made to delete more than 500 records since the file was last resequenced.
RESPONSE MUST BE LU(CR) OR (CR)	There was an invalid response to message: ANY MORE INPUT ENTER LU
ENTER COMMANDS ON INPUT COMMENT DEVICE	An error has been detected; input command device is not the console.
PAULA	
Message	**Significance**
ILLEGAL INPUT STATEMENT	The command cannot be recognized.
ERROR ON INPUT DEVICE	An irrecoverable error was detected on the device.
ILLEGAL M, STATEMENT	A tape motion request cannot be interpreted.
ILLEGAL D, STATEMENT	There is an invalid sector number parameter.
NAM NOT FOUND	The designated module could not be located.

DEBUG ASSISTANCE

The following paragraphs explain COS debug.

Monitor

A module in CYBERDATA was created to point to a serious system failure if a certain test performed by this module is not satisfied. This module, called MONITR, is a debug routine which checks for corruption of system tables, low core, etc. This module is not always alive in the system.

NOTE

For tracking down bugs and/or for testing own codes, etc., the monitor should be used to help spot problems. To use the monitor in this manner, turn on the SELECTIVE SKIP switch (up position). (This will generate extra system overhead which could affect otherwise heavy system use.)

Table E-19 is a list of error codes and the error printout descriptions.

The system prints out the following message to note the errors which have occurred. See table E-19 for the error codes.

```
T=XX   E=XX   P=XXXX   Q=XXXX

A=XXXX   M=XXXX   I=XXXX   H:MM:SS
```

Table E-19. Monitor Errors

E	T	Q	A	I	Description
01†	Terminal number	Unused	TRB ADR	ATT ADR	Disk address loop (Terminal-accessing disk left locked)
02	Terminal number	Unused	Unused	ATT ADR	Stack full
03	Terminal number	Unused	Unused	ATT ADR	No index (EXDFUN)
04	Terminal number	Unused	Unused	ATT ADR	FRP chain corrupted
05	Unused	Unused	Unused	Unused	Low-core checksum error
06	Terminal number	Unused	Unused	ATT ADR	Nonexistant function called (ENTPRG)
07	Terminal number	Unused	Unused	ATT ADR	Supervisors TIQ non-zero
08	Number of terms	Unused	Unused	Unused	Illegal number of terminals
09	Terminal number	Unused	Unused	ATT ADR	No conversion table
10	Terminal number	Unused	Word/Term‡	ATT ADR	Error in ATT
11	Terminal number	Unused	Unused	ATT ADR	No supervisor when SV called
12					Unused
13	Terminal number	Unused	Unused	ATT ADR	Format number 0 referenced
99†	Unused	Core location	PRD index	ATT ADR	Not an error

NOTE

148/580B

†This printout can be caused with the patch LHC,MONITR,1[illegible]/580E. Each time a disk-resident module is loaded into core the message is printed giving information on the module loaded.

‡High-order 8 bits contain index into ATT; low-order 8 bits contain terminal number.

APPENDIX F

SYSTEM UTILITY PROCESSOR COMMANDS

System Utility Processor Commands

INTRODUCTION

The system utility processor (SUP) is a generalized tape utility for the 1700 series Computer Systems under control of COS. The system utility processor is designed to provide a set of functional operations to process magnetic tape files created on CDC or other manufacturer's equipment. The SUP capabilities encompass the more complex record formats and labeling structures usually found in data processing environments.

DESIGN REQUIREMENTS

The system utility processor is designed to provide a medium through which the user can utilize a 1700 series system to reduce I/O processing such as off-line printing of listable tapes on other systems. The system utility processor also provides data manipulation of 1700-system-created tapes to augment other 1700 series features, such as readable tape dumps and improvement of I/O efficiency, through the blocking and deblocking of data files.

SYSTEM CONFIGURATION REQUIREMENTS

SUP is designed to run under COS control with a minimum of 5K of available storage beyond the basic system. Under COS, SUP is installed in the program library and is executed as a background job.

MINIMUM HARDWARE CONFIGURATION

When using COS, the minimum hardware required for the system utility processor includes a minimum 5K to be available for background processing, a teletypewriter or CRT, tape transports, disk pack, and a printer (if PRINT and DUMP functions are used extensively).

NOTATIONAL CONVENTIONS

Parameters are defined as keyword identifiers and may be entered via the teletypewriter in any order, unless otherwise specified. The OPEN and CLOSE statements are the only exceptions. These two statements require the first parameter following the operator (OPEN, CLOSE) to be either Inn, Onn, VFnn, or PRnn. A value may be associated with the keyword parameter, which may be either numeric (decimal), hex-numeric, or alphabetic. To assign a value to a keyword, the following rules apply:

- Numeric

  A numeric value may be attached to a keyword by following the keyword with a numeric value such as PARM12, where 12 is the associated numeric value. Numeric values may be assigned to a keyword through the equal sign as in PARM=12.

- Hex-numeric

  A hexadecimal number may be assigned to a keyword in the same manner as decimal numerics except a dollar sign ($) is inserted immediately preceding the hexadecimal number such as PARM$C or PARM=$C.

- Alphabetic

  An alphabetic string may be assigned to a keyword via the equal sign, but it must also be delimited by quotes as in PARM='ALPHA STRING'.

The symbols [ ], ( ), ,...., ___, and ' are used to define all the SUP control statements. The symbols are not parts of any statement but are only used to indicate how a control statement should be entered on the console.

The symbols are defined as follows:

- [ ]

  The brackets represent optional parameters. Any parameters enclosed in brackets may or may not be entered on the console depending on the user's needs. If more than one parameter is enclosed in the brackets, only one item can be entered on the console.

- ( )

  The parentheses indicate that a choice must be made between parameters. One of the parameters from the vertical stack must be coded.

- ,....

  This symbol indicates more than one set of parameters may be entered on the console.

- ___

  To select a parameter or operator, the underline symbol must be used. All following capital letters which are not underlined may be included without intervening blanks; however, they may be ignored.

- ’

  This symbol indicates a single quote or apostrophe.

FEATURES

The features of the system utility processor are selected by the user to best satisfy his requirements. The feature package is modularly designed such that optional modules may be included or excluded in any system utility processor. Standard modules are required. The following are features that include both standard and optional modules:

- Copy

  This optional module copies data from tape to tape.

- Dump

  This optional module prints a dump from the tape to the printer in either hexadecimal or character mode.

- Initialize

  This optional module writes volume 1 tape headers with volume serial numbers on any tape.

- Labels

  This optional module provides for the reading and writing of standard tape labels and trailer records.

- Print

  This optional module prints standard listings from a tape.

- Selection

  The selection optional module selects records for processing based on specified criteria.

- Verify

  The verify optional module verifies data on two tape files and checks for equality.

- Blocking

  This standard module allows the processing of either blocked or unblocked records.

- Conversion

  The conversion standard module allows the selection of the following data conversion options:

  — ASCII to EBCDIC
  — EBCDIC to ASCII
  — ASCII to BCD
  — BCD to ASCII
  — EBCDIC to BCD
  — BCD to EBCDIC

- Positioning

  This standard module positions tapes to given records, blocks, or records in blocks, using any of the processing functions.

- Record formats

  This is a standard module in which all functions of SUP can process the following record formats:

  — Variable-length unblocked records
  — Variable-length blocked records
  — Fixed-length records
  — Fixed-length, blocked records
  — Undefined records; for example, fixed- or variable-length records that follow no standard or are intermixed

SYSTEM UTILITY PROCESSOR LOADING PROCEDURE

SUP is loaded into memory, for execution, from the COS program library. The following is the instruction for loading and execution of the system utility processor:

```
*SUPLP
**TAPE UTILITY**
*NEXT:
```

UTILITY CONTROL LANGUAGE FUNCTION

SUP executes through the use of a declarative and operational utility control language. Declarative control statements define file characteristics and internal processing options such as data conversion and blocking. The operational control statements define functional operations such as tape copy and transferring of data on tape to the printer. Control statements consist of an OPERATOR, which is a mnemonic describing the function to be executed, and a list of parameters. Parameters in a control statement may be either optional or required, depending on the function. If a parameter is required in a function but is omitted, a message is generated to request that the parameter be stated. If a parameter is optional in a function but is required because of previous options, a message is generated to request that the parameter be stated.

The general format for all declarative and operation control statements is:

```
OPERATOR,PARM1,PARM2,...,
```

Declarative Control Statements

The two declarative control statements (OPEN and CLOSE) are used to control the following functions:

- Blocking
- Conversion
- Labeling
- Record format
- File termination
- Positioning to files or records
- Data processing

By using the OPEN statement, the user specifies the characteristics of the file or files on which a function is to be applied.

The OPEN statement must be issued for all logical units used by a function prior to the function's use.

After a process is terminated, the user must specify the CLOSE statement. This statement provides for end-of-volume termination, tape marks, labels, and tape-positioning options.

OPEN Statement

The OPEN statement defines the characteristics of the magnetic tape files that are used in a process. The statement is used to assign blocking factors, record lengths, record formats, label formats, conversion information on either input or output data files, and positioning to records or files.

The format for the OPEN statement is:

```
       ⎧Inn  ⎫   ⎡SL⎤    ⎡V ⎤    ⎡B⎤
OPEN,  ⎨Onn  ⎬,  ⎢BL⎥,   ⎢VB⎥,   ⎢E⎥,
       ⎪VFnn ⎪   ⎣NL⎦    ⎢F ⎥    ⎣A⎦
       ⎩PRnn ⎭           ⎢FB⎥
                         ⎣U ⎦

[SFnn ,]   [BRnnnnn,]   ⎡PNAM [='XXXXXX']⎤
                        ⎢TNAM [='XXXXXX'],⎥
                        ⎢SNAM [='XXXXXX']⎥
                        ⎣NAM  [='XXXXXX']⎦

[LBnnnnn,]   [LRnnnnn,]   [LCnnn]
```

Where:

- Either I, O, VF, or PR must be the first parameter entered after the function statement. The remaining parameters may be input in any order.
- Inn—input

  I indicates that the OPEN statement applies to the input device, where nn specifies the one- or two-digit logical number of the input device. If nn is omitted, the standard input logical unit number is assumed.
- Onn—output

  O indicates that the OPEN statement applies to the output device, where nn specifies the one- or two-digit logical number of the output device. If nn is omitted, the standard output logical unit number is assumed.
- VFnn—verify

  VF indicates that the OPEN statement applies to the verify file, where nn specifies the one- or two-digit logical unit number of that file. If nn is omitted, the standard output logical unit number is assumed.
- PRnn—print

  PR indicates that the OPEN statement applies to the list device, where nn specifies the one- or two-digit logical number of that device. If nn is omitted, the standard list device is assumed.

LABEL TYPES

The label types for the OPEN statement are:

- SL

  This means to process labels as standard tape labels.
- BL

  This means to bypass labels.

- NL

  This indicates no labels, meaning perform no label processing.

Definition of the label types are as follows:

- SL—standard

  Input label processing involves the verification of the volume, header, and trailer labels. Output processing requires a volume label and header labels or two tape marks. Label processing ensures all labels are correct and a volume has expired before it is written over. Automatic sequencing to the next input volume and multivolume output processing is included in standard label processing. Labels may be American National Standard labels or EBCDIC standard labels (see conversion option on page F-2). Input label processing requires HDR1, EOF1, or EOV1 labels. Output processing generates HDR1, HDR2, EOF1, EOF2, EOV1, and EOV2 labels.

  If open processing of the HDR2 label is featured, it eliminates the need for specifying block size, record size, or record formats in the OPEN statement.

  The discussion on page F-17 contains a complete description of label processing and label content.

- BL—bypass labels

  If BL is specified the tape is positioned to the first record of the specified file. No attempt is made to verify the volume, header, or trailer labels. BL must not be specified for output files.

- NL—no labels

  If NL is specified the tape is not positioned. If the label parameter is omitted, NL is assumed.

RECORD FORMAT

The record format is defined as follows:

- V—variable-length records

  In the first 32 bits of a record, 16 bits must contain a binary number specifying the record length. The last 16 bits must be zero-filled.

- VB—variable-length, blocked records

  In the first 32 bits of a block, 16 bits must contain a binary number which specifies the block length. The next 16 bits must be zero-filled. The first variable-length record of the block must follow the 32 bits.

- F—fixed-length records

  Each record must be exactly the same length.

- FB—fixed-length, blocked records

  Each block must contain an integral number of fixed-length records. There is no requirement that the block length remain fixed. Fixed blocking usually has a maximum length.

- U—undefined record format

  Each record is treated as an undefined, variable-length record. Blocking cannot be used in the undefined format. Print files may be specified as U only. If the format parameter is omitted, U is assumed. Refer to page F-23 for a description of each record format.

DATA FORMAT

The following parameters specify data conversion. To specify conversion, both the input and output file data formats must be defined. For example, if the input data format is specified as A (ASCII) and the output tape is specified as E (EBCDIC) the conversion of ASCII to EBCDIC takes place. If the data format of the specified input and output files is the same or not given, no conversion takes place.

- B

  B indicates the specified tape file contains BCD data (seven-track only).

- E

  E specifies the tape file contains EBCDIC data. If SL is specified, E implies EBCDIC labels are to be read or written.

- A

  This indicates the specified tape file contains ASCII or binary. ASCII or binary is assumed if a code is not entered.

Label processing uses the conversion code to imply the label type. Since EBCDIC codes are primarily IBM standard, E causes System/360/370, OS/DOS/VS-compatible labels to be generated.

The data format code may be necessary to list or dump certain data tapes since the system printer can accept only one data format. If the file is to be properly printed, therefore, the conversion code must be specified. Binary data does not require conversion. The default is either binary or ASCII. Refer to the conversion tables on page F-24.

POSITIONING AND SELECTIVE RECORD PROCESSING

Parameters, which cause input or output files to be positioned to a given record and file, may be specified. Other parameters may be specified to selectively process records used as input to or output of the COS assembler.

These parameters are:

- SFnn—skip files

  This parameter specifies skip nn files before processing where nn is a one- or two-digit decimal number. Skip files may be used on either input or output files.

- BRnnnnn—bypass records

  This parameter specifies bypass nnnnn records before processing where nnnnn is a one-to-five-digit decimal number not greater than 32,767. Bypass records may be used on either input or output files.

  Note

  > The BR parameter may not be used with the parameters PNAM, TNAM, SNAM, or NAM because the resulting output is unreliable.

- PNAM (='XXXXXX')—position to name

  This parameter specifies positioning of a data file to a name statement or name block which contains the name specified by XXXXXX where XXXXXX is a one-to-six-character name. If XXXXXX is not specified, the first NAM block or statement is used. The PNAM parameter bypasses all records prior to the specification condition. Positioning of files occurs during an operational function; therefore, the input file is not positioned during OPEN processing.

SELECTIVE PROCESSING OF PARAMETERS

Selective processing which uses the SNAM, TNAM, and NAM parameters specifies positioning to given records as in PNAM and termination of the process when the record is found. Selective processing can occur only on input files. Processing files that are not 1700 source data, listable, or relocatable records produce unreliable results. Selective processing does not require a specified type of file (source listable or relocatable).

- TNAM (='XXXXXX')—terminate name

  This indicates the termination of processing on the specified name statement or block. If XXXXXX is specified, termination occurs when the name specified by XXXXXX is located. If XXXXXX is not specified, the file is terminated on the first name statement or block. Where TNAM and PNAM are used jointly, termination tests are not executed until at least one record has been processed. This feature allows the processing of all records between specified name statements and blocks.

- SNAM (='XXXXXX')—select name and terminate

  This specifies the selection of records following and including the specified name statement or block and terminates processing on the next name statement or block. SNAM is similar to using the PNAM and TNAM parameters where PNAM specifies the name block and TNAM does not specify a particular name block. SNAM may be used, therefore, to selectively process one file delimited by name statements.

- NAM (='XXXXXX')—name blocks or statements

  This specifies the selection of records which are only name blocks or statements. If XXXXXX is specified, the selection is limited to name blocks with the name specified by XXXXXX. Usually a name is not specified since the NAM parameter generally is used to determine the names of all programs on a given file. The NAM parameter when used with the PR function, for example, results in a listing of all name statements on the associated input file.

SPECIFICATION OF RECORD SIZES

Before processing a data file, the block size and/or record size should be specified; however, for convenience, block and record sizes have default values of 136 characters. If files are unblocked, either record or block sizes may be specified. If both are specified, they must be equal. Block and record sizes are generally maximum values; however, if fixed-length records or blocks are specified, the record size must be the exact record size. If record and block sizes are incorrectly specified, a warning message is used to indicate the error. Processing is possible if a warning occurs; however, results are unpredictable.

Since records are read and written from allocated storage, it is possible that excessive block or record sizes could exceed the available storage. The OPEN statement precludes this possibility by terminating and issuing a severe error message, indicating insufficient storage. (see page F-14). The following parameters may be used to specify associated block and record sizes.

- LBnnnnn—length of block

  This indicates the maximum block length where nnnnn is a one-to-five-digit decimal number (not greater than 32,767), specifying the number of characters in a block. For undefined-, fixed-, or variable-length records, LB must equal LR. For blocked records, LB must be greater than or equal to LR. For PR files, LB is used as the form's width.

- LRnnnnn—length of record

  This indicates the maximum record length where nnnnn is a one-to-five-digit decimal number (not greater than 32,767) specifying the maximum number of characters in a record. If fixed block is specified, LR is the exact length of each record. For PR files, LR is used as the form's width.

- LCnnnnn—line count per page

  This specifies the number of lines per page before ejecting the page (only valid for PR files). If LC is unspecified, 56 lines per page is assumed. By specifying zero, line counting is inhibited.

  NOTE

  Because of operating restrictions, block and record sizes must be a multiple of two characters.

CLOSE Statement

The CLOSE statement terminates file processing and positions the file for further processing. All output files must be closed to ensure the last block will be written.

NOTE

If SUP is terminated by the EXIT instruction, all open files are closed and unloaded.

The format for the CLOSE statement is:

$$\text{CLOSE,}\left\{\begin{matrix}\text{IN}\\ \text{OUT}\\ \text{VF}\\ \text{PR}\end{matrix}\right\},\left[\begin{matrix}\underline{\text{RW}}\\ \underline{\text{UN}}\text{LOAD}\\ \underline{\text{LE}}\text{AVE}\\ \underline{\text{EOV}}\end{matrix}\right]$$

Where:

- Either IN, OUT, VF, or PR must be specified

- IN

  I specifies the CLOSE statement applies to the input file.

- OUT

  O specifies the CLOSE statement applies to the output file.

- VF

  VF specifies the CLOSE statement applies to the verify file.

- PR

  PR specifies the CLOSE statement applies to the print file.

On termination of a procedure on the tape, the tape may be positioned according to one of the following functions:

- RW—rewind

  The tape is rewound and positioned at the load point. If the tape is an output file, a tape mark is written and all end-of-file labels are written before rewinding. Rewind is assumed if no parameters are specified.

- UN—unloaded

  The same procedure occurs as in rewind, except the tape is unloaded following rewind.

- EOV—end of volume

  The function causes all trailer labels to be written, including end-of-file labels (if tape is to be labeled), followed by a double tape mark. The tape is rewound to load point and unloaded.

- LEAVE—leave tape positioned

  This indicates the tape is left positioned at the next file. On input, the tape is positioned at the next file. On output, the last record is written followed by labels if they are specified. The tape is left positioned following the last tape mark, terminating the file.

Backspace Statement

The backspace function is available for tape control to provide an easy method for positioning back to a name statement, which was just used to complete an operation. The backspace instruction backspaces physical records, and the instruction may cause unreliable results when working with blocked files.

The format for this statement is:

$$\underline{\text{B}}\text{SPACE,} \quad \left\{\begin{array}{c} \underline{\text{I}} \\ \underline{\text{O}} \\ \underline{\text{VF}} \end{array}\right\} , \quad [\text{nnnnn}]$$

Where:

- Either IN, OUT, or VF must be specified.
- IN

  I specifies the input file is to be backspaced.
- OUT
- O specifies the output file is to be backspaced.
- VF

  VF specifies the verify file is to be backspaced.
- nnnnn

  This indicates the number of blocks or physical records to be backspaced.

Date Processing

When label-processing options are utilized, the user has the ability to initialize dates used in the generation or verification of header labels. The two date initialization functions included in the SUP package are:

- SDATE=yyddd—system date

  This is used to specify the system date, which is used to set creation dates on output volumes and verify expiration dates of output volumes. In this function, yy is the current year, and ddd is the current Julian calendar day.
- EDATE=yyddd—expiration date

  This is used to specify the expiration date, which is set on the output volumes. When the expiration date is reached, the tape may be used for other data. In this function, yy is the year, and ddd is the Julian calendar day.

NOTE

> The SDATE and EDATE instructions may be inserted at any point in the program. If the two instructions are not used, both dates are assumed to be 99999. Verification of the date is not attempted.

OPERATIONAL CONTROL STATEMENTS

Operational control statements provide instructions to execute any of the following functional modules:

- DUMP
- PRINT
- COPY
- VERIFY
- Initialize volume labels

Operational control statements usually follow OPEN control statements. Functions which require input or output files are not executed unless the required files have been opened. In some cases, files may be improperly defined such as improper record type or length specifications. Recovery under these circumstances is not possible. The file or files must be closed and reopened before any function can be executed.

Through proper definition of input and output file characteristics, files may be blocked, unblocked, converted, and/or labeled.

DUMP Statement

The DUMP statement prints the input tape file on the standard list device or any list device assigned in a preceding OPEN command. The dump list may be in either hexadecimal or character mode and may be formatted or unformatted.

The format for this statement is:

```
[DUMP,] [FCnn,] [RCnnnn,]

                                  [PNAM (='XXXXXX')]
[Hex   ]  [ForMat    ]            [TNAM (='XXXXXX')]
[Char ,]  [UnFormat ,]            [SNAM (='XXXXXX')]
                                  [NAM  (='XXXXXX')]
```

Where:

- FCnn—number of files

  This specifies the number of files to be dumped where nn is a one-to-two-digit number indicating the number of files to be dumped.

- RCnnnn—record count

  The number of records to be used, where nnnn is a one-to-four-digit decimal number. If RC is omitted all records are used until a tape mark or End of Tape marker is detected.

- H—hex dump

  H indicates a hexadecimal dump is to be printed.

- C—character dump

  C indicates a character dump is to be printed.

DUMP TYPE

See page F-29 for sample dumps.

- FM—formatted

  This initiates a formatted dump. In the character mode where 100 characters per line are printed, the record number is printed with each number, and the records are separated by triple spacing.

- UF—unformatted

  This is used to initiate an unformatted dump in character mode. Single spacing is assumed with 56 lines per page. If the record length is greater than the listing format width, the record is continued on the following line. If UF is omitted, a formatted dump listing is assumed. Hexadecimal unformatted dumps are not given.

SELECTIVE PARAMETERS

Selective parameters (as previously defined in the OPEN statement) may be used in the DUMP statement. It is possible, therefore, to dump selected records without closing and reopening files. Caution, however, must be exercised when using the selective parameters PNAM, TNAM, SNAM, and NAM. The conditions which cause termination are determined from the record following the last record dumped. It is impossible to obtain the record which caused termination for subsequent processing, without first issuing the backspace function.

The following parameters may be used whether or not they are specified in the OPEN statement:

- PNAM (='XXXXXX')—position to name

  Specifies a NAM block in which all records are dumped until either an end of file or the conditions specified by TNAM are detected.

- TNAM ('XXXXXX')—terminate name

  Indicates dump is terminated upon detection of the specified NAM block or statement.

- SNAM (='XXXXXX')—select name

  Indicates the selection of the set of records following the specified NAM block or statement. SNAM then dumps the records and terminates the dump at the next NAM block or statement.

- NAM (='XXXXXX')—name

  Indicates only the specified NAM block or blocks are dumped until an end of file or the conditions specified by TNAM are detected.

The DUMP statement requires an OPEN statement list file and input file. The list file may be a teletypewriter or line printer, but it may not be a tape file since forms control records may be less than the minimum tape record size.

The list file, as defined under the OPEN statement, specifies printer or teletypewriter forms information.

By specifying block or record sizes, the user may control the length of a single output line. The number of lines per page can be specified by setting the LC parameter of the OPEN statement to the required size. If LC is not specified, the number of lines per page will be 56 (decimal).

The following is an example of a DUMP command sequence:

```
***TAPE UTILITY***
*NEXT:  OPEN,PR9,LB134
*NEXT:  OPEN,I,LB100
*NEXT:  DUMP,RC50
```

This sample dump puts 50 records on the printer (unit 9) after opening the print and input files.

PRINT Statement

The PRINT statement may print an input tape on the standard list device or a device assigned in the OPEN statement. The print module provides for printer control characters which are in the first position of each record. The PRINT instruction is generally used for printing listable tapes.

The format for this statement is:

```
PRINT, [FCnn,] [RCnnnn,] [US,]

[PNAM (='XXXXXX')]
[TNAM (='XXXXXX')]
[SNAM (='XXXXXX')]
[NAM  (='XXXXXX')]
```

Where:

- FCnn—number of files

  The number of files to be printed is specified where nn is a one-to-two-digit number.

- RCnnnn—record count

  The number of records to be used is specified. In this instance, nnnn is a one-to-four-digit decimal number. If RC is omitted, all records are used until a tape mark or end-of-tape mark is detected.

- US—control character

  This specifies USASCI standard printer control characters are in the first character position on each record. COS listable output has USASCI-coded characters.

SELECTIVE PARAMETERS

Selective parameters (as previously specified in the OPEN statement) may be used in the PRINT statement. It is possible, therefore, to print selected records without closing and reopening the files. Caution must be exercised, however, when using the selective parameters PNAM, TNAM, SNAM, and NAM. The conditions which cause termination are determined from the record following the last record dumped. If is impossible to obtain the record which caused termination for subsequent processing without issuing the backspace function. The succeeding parameters may be used whether or not they are specified in the OPEN statement.

- PNAM (='XXXXXX')—position to name

  This parameter specifies a NAM block in which all records are printed until either an end of file or the conditions specified by TNAM are detected.

- TNAM (='XXXXXX')—terminate name

  This parameter indicates printing is terminated on detection of the specified NAM block or statement.

- SNAM (='XXXXXX')—select name

  This indicates the selection of the set of records following the specified NAM block or statement, which represents a specified name or NAM block. NAM prints the records and terminates the printing of the next NAM block or statement.

- NAM (='XXXXXX')—name

  This indicates only the specified NAM block or blocks are printed until an end of file or the conditions specified by TNAM are detected.

Printing continues until an end-of-file or end-of-volume mark is detected or until all tape marks counted equal the number of files specified for printing.

The PRINT statement requires an OPEN input and list file. The block-size parameter is required as described in the DUMP statement. Line counts are not used since the input records are assumed to have standard print control characters.

FORMS ALIGNMENT STATEMENT

Since the PRINT statement is used as a tape-to-print operation, a means to ensure proper forms alignment is included. After the PRINT instruction is issued, the forms in the line printer are ejected, and the first record is printed followed by a page eject. The following message is typed on the operator's console:

```
*FORMS ALIGNED?
```

The operator's response may be any character or the carriage return. The carriage return directs the PRINT module to continue after reprinting the first line. Any character instructs the printer that the forms are not aligned. The operator may realign the printer forms before responding. The forms alignment statement and the printing is repeated until the carriage return is entered. The following is a sample print command sequence:

```
*NEXT:   OPEN,PR9,LB134,LC=60   [CR]
*NEXT:   OPEN,I7,LB200  [CR]
*NEXT:   PRINT,US,RC500   [CR]
*FORMS ALIGNED:   [CR]
```

The preceding set of instructions prints up to 500 records on unit 9, page ejecting every 60 lines or every time a NAM statement is found in the input file.

COPY Statement

The COPY statement is designed to copy a tape file from the standard input device to the standard output device unless otherwise specified. By using a declarative control statement blocking, unblocking, and/or code conversion may be implemented.

The format for this statement is:

```
                                [PNAM (='XXXXXX')]
COPY,  [RCnnnn,] [FCnn,]        [TNAM (='XXXXXX')]
                                [SNAM (='XXXXXX')]
                                [NAM  (='XXXXXX')]
```

Where:

- RCnnnn—record count

  This parameter specifies the number of records to be copied where nnnn is a one-to-four-digit decimal number. If RC is omitted, all records are copied until a tape mark or end-of-volume mark is detected.

- FCnn—file count

  This indicates the number of files to be copied where nn is a one- or two-digit decimal number. If FC is omitted, one file is copied.

SELECTIVE PARAMETERS

Selective parameters (as previously specified in the OPEN statement) may be used in the COPY statement. It is possible, therefore, to copy selected records without closing and reopening the files. Caution, however, must be exercised when using the selective parameters PNAM, TNAM, SNAM, and NAM. The conditions that cause termination are determined from the record that follows the last record copied. It is impossible to obtain the record which caused termination for subsequent processing without issuing the backspace function. The following parameters may be used whether or not they are specified in the OPEN statement:

- PNAM (='XXXXXX')—position to name

  This parameter specifies a NAM block in which all records are copied until either an end of file or the conditions specified by TNAM are detected.

- TNAM (='XXXXXX')—terminate name

  This indicates copy is terminated on detection of the specified NAM or block statement.

- SNAM (='XXXXXX')—select name

  This parameter indicates the selection of the set of records following the specified NAM block or statement, which represents a specified NAM block and terminating copy when the next NAM block or statement is detected.

- NAM (='XXXXXX')—name

  This indicates only the specified NAM block or blocks are copied until an end of file or the conditions specified by TNAM are detected.

The COPY statement can be used to block, unblock, convert, etc. any data tape or 1700 system tape. It also can be used by proper specification of selection parameters to replace, delete, or select relocatable program modules, listable program modules, and source program modules on 1700 system tape. The following is an example of a copy command sequence.

```
*NEXT:   OPEN,I,F,LR80
*NEXT:   OPEN,O,VB,LB400,LR80,E
*NEXT:   COPY,RC403
*NEXT:   CLOSE,O
*NEXT:   CLOSE,I
```

This example copies 403 records (or one file) from the input file, converting the ASCII input data to EBCDIC data and reformats the input records to variable, blocked output records.

The following example copies an assembly listing tape. It copies the ASCII input tape starting from VLOSOP and continues up to but not including MTTYPE. The output tape is blocked EBCDIC. A warning error occurs because input and output record lengths are defined differently; this error is informative only.

```
*NEXT:   OP,I6,NL,U,A,PNAM='VLOSOP'
*NEXT:   OP,O7,NL,VB,E,LB500,LR100
*NEXT:   CO,TNAM='MTTYPE'
**** W003****
```

VERIFY Statement

The VERIFY statement is designed to verify two tape files on a specified unit, or tapes mounted on a standard input and output device. Tapes may be verified with or without the blocking or conversion options.

The format for this statement is:

```
VERIFY, [RCnnnn,] [FCnn,] [H ,]
                          [C  ]

        [PNAM (='XXXXXX')]
[UF  ]  [TNAM (='XXXXXX')]
[FM ,]  [SNAM (='XXXXXX')]
        [NAM  (='XXXXXX')]
```

Where:

- RCnnnn—record count

  This is used to specify the number of records to be verified, where nnnn is a one-to-four-digit decimal number. If RC is omitted, all records are verified until a tape mark or end-of-volume mark is detected on the input file.

- FCnn—file count

  This indicates the number of files to be verified, where nn is a one- or two-digit decimal number. If FC is omitted, one file is verified.

- H— hex dumps

  This specifies a hexadecimal record dump of each error printed. (The dump module must be present.)

- C—character dump

  The character dump specifies a character record dump of each error record printed.

- UF—unformatted dump

  If UF is specified, the error records are formatted to either hexadecimal or character formats. Each record is preceded by header lines which specify block size, record size, and the sequence of the record. If dump is not present, UF is not available.

SELECTIVE PARAMETERS

Selective parameters (as specified in the OPEN statement) may be used in the DUMP function. It is possible, therefore, to verify selected records without closing and reopening the files. Caution must be exercised, however, when using the selective parameters PNAM, TNAM, SNAM, and NAM. The conditions which cause termination are determined by the record following the last record dumped. It is impossible, therefore, to obtain the record which caused termination for subsequent processing without issuing the backspace function.

The following characters may be used whether or not they are specified in the OPEN statement.

- PNAM (='XXXXXX')—position to name

  This parameter specifies a NAM block in which all records are verified until either an end of file or the conditions specified by TNAM are detected.

- TNAM (='XXXXXX')—terminate name

  The TNAM parameter indicates that verification is terminated on detection of the specified NAM block or statement.

- SNAM (='XXXXXX')—select name

  This indicates the selection of the set of records following the specified NAM block or statement, which is terminated following verification of the next NAM block or statement.

- NAM (='XXXXXX')—name

  NAM specifies that only the special NAM block or blocks are verified until either an end of file or the conditions specified by TNAM are detected.

The VERIFY statement verifies data files on a record basis. Data files may be verified if the blocking factors are different; if the data is in different external codes (ASCII, EBCDIC, or BCD); or in some cases, if the data is in different record formats (see page F-23). If records fail to match, the VERIFY instruction dumps both the input and output records to the standard list device. The number of characters not matching and the first character of the record in error are also printed. The dump format is the same as that used in a character record dump. In the event that more than 10 consecutive records are in error, the system interrogates the operator as to whether to continue or to terminate. A carriage return response designates termination. Any other character response followed by a carriage return designates program continuation.

SAMPLE VERIFICATION OPERATION

The following sample instructions verify two files or 1,000 records, whichever occurs first. The input file contains undefined records, not greater than 100 characters; the verify file contains variable blocked records with each record not greater than 100 characters;

and the external codes are in EBCDIC. The differences in record format and content of the records require all conversions be completed before any comparison takes place.

```
**TAPE UTILITY**
*NEXT:  OPEN,PR,LB134,LC=56
*NEXT:  OPEN,I,LB=100
*NEXT:  OPEN, VF7,VB,LB=400,LR=100,E
*NEXT:  VERIFY,FC=2,RC=1000,H
*NEXT:
```

In the preceding sample, if 10 consecutive mismatching records were found the following sequence would occur:

10 ERRORS
*CONTINUE: Y $\boxed{\text{CR}}$

Verification continues.

Initialize Statement

The initialize statement is used to write volume 1 header records following the load point on a specified unit. The initialize function differs from other functions in that once the initialize process is implemented, it must be specially terminated. Once initialization is started, it requests a volume serial number and then writes the first header on the mounted tape. On completion, the tape is unloaded and a request for the next tape is made. The previous volume serial number is incremented (if numeric) and may be used as the next volume serial number.

The format for this statement is:

$$[\text{INIT,}]\ [\text{Onn,}]\begin{bmatrix}\underline{E}\\ \underline{A}\\ \underline{B}\end{bmatrix}$$

Where:

- Onn—output unit

  O specifies the output unit number where nn is a one- or two-digit decimal number. If nn is omitted, the standard tape output unit is assumed.

- E—EBCDIC labels

  E specifies EBCDIC IBM-standard volume 1 labels are to be generated. the VOL 1 label is compatible with System/360/370 OS/DOS/VS.

- B—BCD code labels

  This specifies BCD codes are to be used in the generation of VOL 1 labels.

- A—American National Standard Labels

  This is used to specify American National Standard VOL 1 labels are to be generated.

The label type may be specified by using the OPEN conversion parameters. If the OPEN parameters are omitted, A is assumed. The initialize statement communicates with the operator through a series of messages. Operator responses are used to terminate the initialize statement, to determine if a tape is mounted and ready, and to assign the volume serial number. The following mount message instructs the operator to mount a scratch tape on the specified logical unit:

*MOUNT,OUTPUT,SCRATCH:

Response of a carriage return designates the tape is mounted. Any other character followed by a carriage return terminates the initialize function.

The volume serial number message VOLSER=nnnnn: indicates the next volume serial number to be written on the mounted scratch tape. To change the serial number, the operator enters a one-to-six-character volume serial identifier. The identifier may be alpha, alphanumeric, or numeric. Only characters from A to Z and numerics 0 to 9 are permitted. A carriage return terminates the identifier. If the identifier is completely numeric, the next VOLSER message contains the last number plus one; otherwise, the next VOLSER message contains VOLSER=000001. To use the serial number contained in the message, the operator responds with a carriage return.

The following example illustrates the various features of the initialize statement.

- **TAPE UTILITY**

- *NEXT: INIT,06

  (Use logical unit 5.)

- *MOUNT,OUTPUT,SCRATCH: $\boxed{\text{CR}}$

  (When the tape is mounted, type a carriage return.)

- *VOLSER=000001: $\boxed{\text{CR}}$

  (The first volume serial number is assumed to be 1. A carriage return designates the user does not want to change what is typed.)

- *MOUNT,OUTPUT,SCRATCH: $\boxed{\text{CR}}$

- *VOLSER=000002: 670002

  (This instruction changes the serial number.;)

- *MOUNT,OUTPUT,SCRATCH: CR
- VOLSER=670003: CR

  (The last serial number is incremented.)

- *MOUNT,OUTPUT,SCRATCH: CR
- *VOLSER=670004: SYSAB1 CR

  (The volume serial number is changed to alphanumeric.)

- *MOUNT,OUTPUT,SCRATCH: CR
- *VOLSER=000001: SYS002 CR

  (Since alphanumeric data cannot be incremented, the VOLSER is reset.)

- *MOUNT,OUTPUT,SCRATCH: TERM CR

  (Terminate initialize.)

- *NEXT:

EXIT Statement

The EXIT statement returns control to the operating system after job termination. All files not already closed by a CLOSE statement are closed with the rewind and unload function. If SUP is used to position a file for processing, the operator must ensure the file is closed without the rewind function prior to issuing an EXIT request. The format for this statement is:

EXIT: CR

DIAGNOSTIC PROCESSING

When a diagnostic appears in any of the functions, SUP allows for the correction of the errors by use of the following procedure:

1. To delete a line of type, use a rubout, then a line feed, followed by a carriage return.

2. If there is a syntax error in the function statement, SUP allows a reentry of only the parameter or parameters in error. The unrecognized parameters are retyped by SUP for ease in discerning which parameter is in error.

The following functions illustrate sample parameter errors. The functions underlined are the only commands actually entered.

- *NEXT: OPEN,PR,LB134,LN=54 CR
- INVALID PARM='LN=54.'

  (LN is not a valid parameter.)

- RETYPE PARM: LC=54 CR

  (The proper parameter is entered.)

- *NEXT:

  (NEXT implies the function was acceptable.)

- OPEN,I,SNAM=CLRRGN,LB80

  (No quotes appear around CLRRGN.)

- INVALID PARM='SNAM=CLRRGN,'
  RETYPE PARM: SNAM='CLRRGN'
  *NEXT:

  (Reentered with quotes.)

SYSTEM MESSAGES

System messages are designed to inform the operator of the systems functions, cue the operator to take a certain action, output statistical data, and indicate that errors have occurred during execution. The messages are written either on the system console or the system list device depending on the purpose of the message.

Information Message

Statistical data and information messages such as those that indicate errors during the verify process should appear on the standard list device if a standard device has been activated by an OPEN statement. If not, the messages will appear on the console.

The format for the statistical data message is:

```
*NEXT:  COPY,RC100
00100 INPUT BLOCKS
00100 INPUT RECORDS
00020 OUTPUT BLOCKS
00100 OUTPUT RECORDS
```

NOTE

Output block and record totals may not reflect the same information as input totals. The I/O discrepancy is a result of blocking, unblocking, multiple files, and other functions.

All totals reflect the total number of records that have been read and written. For INPUT files, the total number of records are cumulative over the entire volume. For OUTPUT files, the totals reflect only the last file written minus the last applicable unwritten block. Labeled files may generate zero totals in the block field because the counts are cleared during EOF processing.

Action Messages

Whenever action messages occur, an operator response is required. The action message indicates that a decision as to whether to proceed is possible, an operation has been completed, or more information is necessary before processing can continue. Table F-1 lists all action messages.

Table F-1. Action Messages

Message	Response	Description
*NEXT:	Any utility operational or declarative statement	System has completed all prior requests and can accept next request
*INVALID PARM='XXX...' *RETYPE PARM:	Enter corrected parameter	The characters in quotes are invalid and may be corrected (refer to functional actions)
10 ERRORS *CONTINUE:	Type carriage return $\boxed{cr}$ to terminate or type one character followed by carriage return to continue	Verify function has located 10 consecutive records that contain errors
*MOUNT,OUTPUT,SCRATCH:	Type carriage return, which implies tape is ready; or type any other character followed by a carriage return to terminate the initialize function	See initialize
*VOLSER=nnnnnn:	See initialize function	See initialize function
VOL NOT EXPIRED USE:	Carriage return implies 'do not use', U implies use, ignoring expiration date	Label processing: Output volume header records are checked against the system date
*DATA SET NAME:	DSN='XXXXX'	Label processing: Output volumes require a data set name if not available from input (see page F-17)
VOLSER=nnnnnn	None	Informative tape file just opened with the specified volume serial number

System Error Messages

System error messages are always issued to a comment device, teletypewriter, etc. There are four types of system errors. They include:

- Descriptive
- Critical
- Serious
- Warning errors

DESCRIPTIVE ERROR MESSAGES

The descriptive error messages indicate what the error is and directs that the previous instruction could not be executed. The descriptive error messages are listed in table F-2.

CRITICAL, SERIOUS, AND WARNING ERROR MESSAGES

Critical error messages imply an error has occurred which prevents further processing by SUP. If the critical error occurs, control is returned to the operating system.

Table F-2. Descriptive Error Messages

Message	Description	Action
FILE(S) NOT OPEN	A required file is not open, and the specified function cannot be executed.	Open file and reenter function.
*INVALID OPEN OR CLOSE	The file being opened or closed is already in that state.	Open or close the proper file.
*FUNCTION NOT AVAILABLE	An attempted function is not available in the system. The function is not invalid; rather, the system was configured without the requested module.	Use another function, if possible.
*PARM NOT AVAILABLE	A parameter is not available in the system. The parameter is not invalid; rather, the system was configured without the requested module.	Use another parameter, if possible.
*INCORRECT VOL MOUNT:	The volume mounted does not contain a volume label, or the header label sequence is incorrect; i.e., the wrong volume of a multiple volume file is mounted.	Mount the correct volume, and type a carriage return.

Serious errors are a result of mistakes such as improper specification of valid parameters and tape errors. If a serious error occurs, control returns to SUP, and the user may continue programming.

Warning messages indicate parameters may be inconsistent or a possible processing error has occurred. Execution continues following a warning. Warning messages are used to allow processing flexibility since the inconsistent parameters may be required to execute a function; for example, when converting fixed-length records to variable-length records, the copy function checks for length mismatches. In this case, the warning may be ignored.

Tables F-3, F-4, and F-5 describe corrective action for all coded error messages.

Table F-3. Critical Errors

Error Code	Description	Action
****C000****	Data buffer linkage has been destroyed – cause: I/O malfunction, CPU malfunction	Reload utility

Table F-4. Serious Errors

Error Code	Description	Action
****S000****	Available memory has been filled	Free memory by closing a 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 error, lost data or alarm	Retry the 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 the erroneous record; file cannot be processed as variable length

Table F-4. Serious Errors (cont)

Message	Description	Action
****S004****	Blocking has been requested, and specified block size is smaller than specified record size	Reopen file with proper parameters
****S005****	Variable size error detected prior to write	Attempt to reexecute function after closing and reopening all files; possible hardware malfunction
****S006****	Fixed block error detected prior to write; record length is not specified	Close file, and reopen 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 reopen
****S008****	Labeled file EOF1 trailer label contains invalid information which 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 the file with EOV, and reopen after mounting new tape Reenter function to complete processing
****S011****	A double file mark has been sensed on an input file; processing is terminated	Close input file, and mount next volume; reenter function to complete processing
****S012****	Invalid date	Reenter date function with proper date
****S013****	Labeled volume sequence number incorrect (occurs after OPEN; file is not opened)	Mount proper volume and reopen file
****S014****	Zero-length block specified in OPEN; file is not opened	Reopen, specifying proper block length
****S015****	Block or record length specified is not a multiple of two; file is not opened	Reopen, specifying even block and record length; if either block or record length is odd, the data cannot be processed by the system

Table F-5. Warning Messages

Error 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 statement
xxxW001xxx	File count specified as zero	Reenter function with proper parameters or continue statement
xxxW002xxx	Record count specified as zero	Reenter function with proper parameters or continue statement
xxxW003xxx	Input and output record lengths have been specified differently for COPY	Reenter function with proper parameters or continue statement

LABEL PROCESSING

Label processing is discussed in the following paragraphs.

Labeling

During the OPEN, READ, WRITE, and CLOSE operations, labels may be processed. The three potential processing techniques available include: no labels, bypass labels, and standard labels. Each technique requires a specific file structure.

NO LABELS

When no labels are defined, the volume may contain one or more files. The file structure for a single volume, unlabeled file is defined in figure F-1.

Single File	File A						*	*
Multifile	File A	*	File B	*	...	File N	*	*

Figure F-1. Single-Volume, Unlabeled File

The volume is terminated by two tape marks. Specific provisions to process files extending to more than one volume have been excluded; however, a double tape mark on input terminates all processing, which allows mounting of the next volume. Output processing terminates at the end-of-tape mark. An error message is issued. A new tape is mounted, and processing is continued by reexecuting the function.

BYPASS LABELS

The bypass label processing ignores all labels; however, SUP expects the normal sequence of tape marks found in standard labels. The bypass label option is not available for output files.

STANDARD LABELS

Standard-labeled volumes may contain one or more files and may extend to more than one volume. Each volume must contain a VOL1 header label as the initial record. Each file must be preceded by a HDR1 label and terminated by an end-of-file label. The system utility processor processes VOL1, HDR1, HDR2, EOF1, EOF2, EOV1, and EOV2 labels.

Definitions

The following definitions apply to COS label processing:

FILE

A file is a collection of data consisting of records pertaining to a general subject. The delineation of a file may be arbitrary. The absence of information may result in the creation of a file, delineated by adjacent tape marks, without information.

VOLUME

A volume is a physical unit of storage media. In the discussion of SUP, volume is synonymous with a reel of magnetic tape.

FILE SET

A file set is a collection of one or more related files, recorded on one or more volumes. A file set consists of:

- One file recorded on a single volume
- More than one file recorded on a single volume
- One file recorded on more than one volume
- More than one file recorded on more than one volume

LABEL

A label is defined as a block at the beginning or end of a volume or a file, which serves to identify and/or delineate that volume or file.

LABEL GROUPS

Label groups are defined as a collection of contiguous labels related to a file. The label group either precedes or follows the related file on a single volume. The volume header labels and the following file header labels are the first label group on a volume.

TAPE MARK

A tape mark is a special configuration record on magnetic tape, synonymous with an end-of-file mark, indicating the boundary between files and labels and between certain label groups. The tape mark configuration for nine-track tape is defined in CDC-STD 1.10.005 and CDC-STD 1.10.006. The tape mark configuration for seven-track tape is an octal 17 written in even parity.

FIELD

A field is a specified area in a record which is used for a particular category of data.

Requirements

The requirements defined in the following paragraphs are necessary for SUP.

GRAPHIC REPRESENTATION

In this section, n means any numeric digit (0 through 9). An a means any of the characters occupying the center four columns of ASCII, except position 5/15, and those positions where there is a provision for alternative graphic representation.

The limitation of the a characters is intended as a guide to provide maximum interchangeability and consistent print especially when international interchange is a possibility. Checking for conformity to this limitation is not implied.

LABEL FORMAT

Each required label is an 80-character block recorded in even parity mode at the same density as the rest of the tape for seven-track tapes or in convert mode ASCII for nine-track tapes.

TAPE MARKS

Use of tape marks is not permitted in positions that are not specified in this manual. A file which contains nonstandard tape marks cannot be considered as a legitimate interchange tape.

LABELLED FILE STRUCTURE

Required labels and tape marks must be used to establish the file structure according to the formats displayed in table F-6. The table shows that the beginning of the tape is on the left and the end of the tape is on the right. Required labels are specified by their first four characters, and tape marks are specified by asterisks.

No Good For 370

VOLUME HEADER LABEL

Every volume shall have a volume header label (VOL1) as the first block in the volume. No volume header label shall be used at any other place in the volume.

END-OF-VOLUME LABEL

An end-of-volume label (EOV) should follow the last block of a file when a volume ends in the file. One tape mark should immediately precede, and two tape marks should immediately follow every end-of-volume label group. No file sets should be terminated by an end-of-volume label.

FILE HEADER LABEL

Every file must be preceded by a file header label (HDR1). Whenever a volume ends in a file, the continuation of the file in the next volume must also be preceded by a file header label. Every file header label group must immediately be followed by a tape mark.

END-OF-FILE LABEL

The last block of every file must be followed by an end-of-file label (EOF1). A tape mark must immediately precede, and another tape mark must immediately follow, every end-of-file label group. The end-of-file label that appears at the end of the last file in a set must be followed by two tape marks.

If end-of-file and end-of-volume labels coincide on a multifile, multivolume tape, the following configurations will be used. Letters in parentheses indicate to which file the labels belong. A dash indicates the label belongs to neither file.

```
VOL1  HDR1*(FILE A)------------------*EOF1*HDR1**EOV1**
(—)                                   (A)    (B)     (—)

VOL1  HDR1*-------------------------(FILE B)----
(—)    (B)
```

Table F-6. Structure of Magnetic Tape Files

Type of File	File Format
Single-volume file	VOL1 HDR1*-----------*EOF1**
Multivolume file	VOL1 HDR1*-----------*EOV1** VOL1,HDR1*-----------*EOF1**
Multifile volume file	VOL1 HDR1*-----------*EOF1*HDR1*---*EOF1**
Multifile, multivolume file	VOL1 HDR1*--(A)--*EOF1*HDR1*---(B)---*EOV1** VOL1 HDR1*--(B)----------*EOV1** VOL1 HDR1*--(B)---*EOF1*HDR1*---(C)--*EOF1**

Optional Label Format

Optional operating system labels and user labels must be fitted into the file structure according to the following rules so the relationship between required labels and file is not affected.

OPTIONAL FIELDS

When optional is used in defining a field, it means the field may contain information already described. If an optional field does not contain the designated information, the field contains blanks.

REQUIRED FIELDS

Fields which are not defined as optional are required and are written or read as specified in SUP.

VOLUME HEADER LABEL (VOL1)

The fields in the volume header label are arranged as shown in table F-7. Anyone who is using a magnetic tape that is not his must return the entire volume header label unless otherwise authorized by the user; however, a volume header label may be written if authorized.

Table F-7. Volume Header Label (VOL1)

Character Position	Field	Name	Length	Description
1 to 3	1	Label identifier	3	Must be VOL
4	2	Label number	1	Must be 1
5 to 10	3	Volume serial number	6	Six a characters permanently assigned by the user to identify this physical volume; i.e., reel of tape
11	4	Accessibility	1	As an a character which indicates any restrictions on who may have access to the information in the volume; a blank means unlimited access; any other character means special handling, in the manner agreed between the interchange parties; field is not edited
12 to 31	5	Reserved for future use	20	Must be blanks
32 to 37	6	Reserved for future use	6	Must be blanks
38 to 51	7	owner identification	14	Any a characters, identifying the owner of the physical volume; may be blank
52 to 79	8	Reserved for future use	28	Must be blanks
80	9	Label standard level 8	1	1 means the labels and data format on this volume conform to the requirements of this manual; blanks means the labels and data formats on this volume require the agreement of the interchange parties

EOL
16

EOD
19/15-E

FILE HEADER LABEL

The fields in the file header label are arranged as shown in table F-8.

Table F-8. File Header Label 1

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 21	3	File identifier	17	Any a characters agreed on between originator and recipient	
22 to 27	4	Set identification VOL ID	6	Any a characters to identify the set of files of which this is one; this identification must be the same for all files of a multifile set	EOD 29
28 to 31	5	File section number GENERATION NUMBER	4	The file section number of the first header label of each file is 0001; applies to the first or only file on a volume and to subsequent files on a multifile volume; field is incremented by one on each subsequent volume of the file	EOD 38
32 to 35	6	File sequence number	4	Four n characters denoting the sequence; i.e., 0001, 0002, etc. of files in the volume or set of volumes; in all the labels for a given file, this field contains the same number	EOD 4
36 to 39	7	Generation number (optional)	4	Four n 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	
40 and 41	8	Generation version	2	Two n characters distinguishing successive repetitions of the same generation; generation version number of the first attempt to produce a file is 00	
42 to 47	9	Creation date	6	A blank followed by two n characters for the year followed by three n characters for the day (001 to 366) in the year	EOD 32
48 to 53	10	Expiration date	6	Same format as field 9; this file is regarded as expired 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 overwritten; to be effective on multifile volumes, therefore, the expiration date of a file must be less than or equal to the expiration date of all previous files on the volume	
54	11	Accessibility	1	An a character which indicates restrictions on who may have access to the information in this file; a blank means unlimited access; any other character means special handling, in a manner agreed between the interchange parties	EOD 49/15-8
55 to 60	12	Block count	6	Must be zeros	
61 to 73	13	System code (optional)	13	Thirteen a characters identifying the operation system that recorded this file	
74 to 80	14	Reserved for future use	7	Must be blanks	

END-OF-FILE LABEL (EOF1)

The fields of the end-of-file label are arranged as shown in table F-9.

Table F-9. End-of-File Label (EOF1)

Character Position	Field	Name	Length	Description
1 to 3	1	Label identifier	3	Must be EOF
4	2	Label number	1	Must be 1
5 to 54	3 through 11	Same as corresponding fields in the first file header label (optional)	Total 50	Same as corresponding fields in the first file header label
55 to 60	12	Block count	6	Six n characters denoting the number of data blocks (exclusive of labels and tape marks) since the preceding HDR label group
61 to 80	13 and 14	Same as corresponding fields in the first file header label (optional)	Total 20	Same as corresponding fields in the first file header label

EOF
9

SECOND END-OF-FILE LABEL

This option constitutes a second end-of-file label specified by EOF2 as the label identifier and label number. The label contains the same information in fields 3 through 8 (all optional) as does HDR2.

END-OF-VOLUME LABEL (EOV1)

The fields of the end-of-volume label are arranged as shown in table F-10.

Table F-10. End-of-Volume Label (EOV1)

Character Position	Field	Name	Length	Description
1 to 3	1	Label identifier	3	Must be EOV
4	2	Label number	1	Must be 1
5 to 54	3 through 11	Same as corresponding fields in the first file header label (optional)	Total 50	Same as corresponding fields in the first file header label
55 to 60	12	Block count	6	Six n characters denoting the number of data blocks (exclusive labels and tape marks) since the preceding HDR label group
61 to 80	13 and 14	Same as corresponding fields in the first file header label (optional)	Total 20	Same as corresponding fields in the first file header label

EOV
9

SECOND END-OF-VOLUME LABEL

This option constitutes a second end-of-volume label specified by EOV2 as the label identifier and label number. The label contains the same information in fields 3 through 8 (all optional) as does HDR2. The fields of the second file header label are arranged as shown in table F-11.

Table F-11. Second File Header Label

Character Position	Field	Name	Length	Description
1 to 3	1	label identifier	3	Must be HDR
4	2	Label number	1	Must be 2
5	3	Record format	1	F–fixed length V–variable with the number of characters in the record specified in binary U–undefined
6 to 10	4	Block length	5	Five n characters specifying the maximum number of characters per block
11 to 15	5	Record length	5	Five n characters specifying: If record format is F, record length is V, maximum record length including any count fields; if U, then undefined
16 to 50	6	Reserved for operating systems	35	Reserved for operating systems; any a characters
51 and 52	7	Buffer offset (optional)	2	Input ignored; output zeros
53 to 80	8	Reserved for future use	28	Must be blanks

I/O Processing With Standard Labels

The following paragraphs discuss input and output processing using standard labels.

INPUT FILE PROCESSING WITH STANDARD LABELS

The OPEN statement initiates label processing by reading the VOL1 header record. Header 1 overlays the VOL1 header in a 40-word buffer label. Header 1 is retained. If header 2 is available, it is added to the 40-word buffer, and if necessary, the contents of the block length and record length fields are used to complete open processing. All other label records are placed in a chained buffer until a tape mark is encountered. The saved labels may be used by SUP for output labels with the copy function.

OUTPUT FILE PROCESSING WITH STANDARD LABELS

All output files must contain a VOL1 header record, however, output files may contain other header records. If the output file contains a HDR1 header, the expiration date is checked against the system date. The output volume may not be used if the expiration date is greater than the system date unless the operator responds to the following message:

$$\text{VOL NOT EXPIRED: } \begin{matrix} \underline{\text{U}} & \boxed{\text{CR}} \\ \text{X} & \boxed{\text{CR}} \end{matrix}$$

U—implies use of volume
X—implies the volume is not available

Once the initial verification is complete, new headers must be written. Since COPY is the only instruction that uses output files, the new headers may originate from either of two sources: the input volume or the operator. If the new headers are to be copied from the input volume, the input volume must contain standard labels. All labels except the VOL1 label are transferred from the input to the output file. To specify new output labels, the input file must be opened as BL (bypass labels) or NL (no labels), and the output must be opened as SL (standard labels). In this instance, the operator enters the following label information:

Data Set Name — DSN = 'a...a'

Where:

a is an alphanumeric character from 1 to 17

All other header fields are generated by SUP in accordance with the preceding standards.

END-OF-VOLUME AND END-OF-PROCESSING OF STANDARD LABELS

The system utility processor recognizes and writes end-of-volume (EOV) or end-of-file (EOF) labels. Automatic volume sequencing occurs on input and when the EOV label is sensed. On output, the sensing of the EOT marker causes an EOV label to be written and automatic sequencing to a new volume.

RECORD PROCESSING REQUIREMENTS

The following descriptions describe record formats and processing requirements for the various record types.

Record Formats

FIXED-LENGTH RECORDS

Each record must be exactly the same length (figure F-2).

Label (if any)	T M	REC 1	G A P	R REC 2	G A P	....REC 11	G A P	T M	Trailer (if any)	T M	T M

Figure F-2. Fixed-Length Records

FIXED-LENGTH, BLOCKED RECORDS

Each block must contain an integral number of fixed-length records. There is no requirement, however, that the block length must remain fixed. Only the last block may be shorter than any preceding blocks. A maximum length is associated with fixed blocking. The fixed -length, blocked record is shown in figure F-3.

Block 1 Block N

Label	T M	REC 1	REC 2	REC 3	G A P	....	G A P	REC N-1	REC N	G A P	T M	Trailer	T M	T M

Figure F-3. Fixed-Length, Blocked Records

VARIABLE-LENGTH RECORDS

A standard variable-length record consists of a record descriptor word (RDW) and data characters. The number of characters is contained in the RDW. Refer to figure F-4.

Label	T M	R D W	REC 1	G A P	R ....D W	REC N	G A P	R D W	REC	G A P	T M	Trailer	T M	T M

Figure F-4. Variable-Length Records

In figure F-4, RDW is 32 bits. The upper 16 bits is record size and the lower 16 bits is 0 (reserved).

VARIABLE-LENGTH, BLOCKED RECORDS

A variable-length, blocked record consists of a block descriptor word (BDW) and any number of variable-length records. The BDW holds the block length, i.e., the sum of the RDW's + 4 (figure F-5).

Label	T M	B D W	R D W	REC 1	R D W	REC 2	G A P	....	B D W	R D W	REC N	G A P	T M	Trailer	T M	T M

Figure F-5. Variable-Length, Blocked Records

In figure F-5, BDW is 32 bits. The upper 16 bits is the block size and the lower 16 bits is 0.

Undefined Records

An undefined record is either variable or fixed length. The user must provide record length by using the LR parameter. The record format is uncontrolled, and blocking is not possible.

Record Processing

Record processing of the acceptable format types is related to the OPEN statement. Formats may be converted only as indicated in table F-12.

Table F-12. Record Processing Format Conversion

Record	Convert To
Fixed Length	Fixed blocked Variable Variable blocked Undefined
Fixed Length Blocked	Fixed Variable Variable blocked Undefined
Variable Length Records	Variable blocked Undefined
Variable Length Blocked	Variable Undefined
Undefined Records	Fixed – if undefined records are fixed length Fixed blocked – if undefined records are fixed length Variable Variable blocked

The open parameters LB and LR must conform to the rules for fixed block and for variable and variable blocked records if blocking is used.

FIXED BLOCK

The rules are:

- LB must be equal to or greater than the longest record to be read or written.
- LR must be less than or equal to LB.

VARIABLE AND VARIABLE BLOCKED

The rules are:

- LB must be equal to or greater than the largest record.
- LR must be equal to or greater than the largest record.
- LR must be less than or equal to LB.

CONVERSION CODES

Table F-13 shows the relationships among codes in an ascending collating sequence. The chart explains the equivalence between characters in the various machine and teletype codes.

Table F-13. Collating Sequence

Collating Sequence	EBCDIC Character	ANSI Character	Internal and ASCII Tape Code (Hex)	Tape Code BCD (Octal)	Tape Code EBCDIC (Hex)
00	Others	Del	7F		3E
01	Reject	@	40	56	3F
02	Space	Space	20	20	40
03					
04					
05					
06					
07					
08					
09					
10					
11					

Table F-13. Collating Sequence (cont)

Collating Sequence	EBCDIC Character	ANSI Character	Internal and ASCII Tape Code (Hex)	Tape Code BCD (Octal)	Tape Code EBCDIC (Hex)
12	\	[Left Bracket	5B		4A
13	.	.	2E	73	4B
14	<				4C
15	(	(Left Paren	28	34	4D
16	+	+	2B	60	4E
17	|				4F
18	&	&	26	15	50
19					
20					
21					
22					
23					
24					
25					
26					
27					
28	!	!	21	52	5A
29	$	$	24	53	5B
30	*	*	2A	54	5C
31	)	) Right Paren	29	74	5D
32	;	;	3B	77	5E
33	¬	——	5F	36	5F
34	–	–	2D	40	60
35	/	/	2F	21	61
36					
37					
38					
39					
40					
41					
42					
43					
44			7C		6A
45	,	,	2C	33	6B
46	%	%	25	16	6C
47	--	∧	5E	37	6D
48	>				6E
49	?	?	3F	75	6F
50					
51					
52					
53					
54					
55					

Table F-13. Collating Sequence (cont)

Collating Sequence	EBCDIC Character	ANSI Character	Internal and ASCII Tape Code (Hex)	Tape Code BCD (Octal)	Tape Code EBCDIC (Hex)
56					
57					
58					
59	\	\	60		79
60	:	:	3A	00	7A
61	#	£	23	55	7B
62	@	£			7C
63	'	'	27	14	7D
64	=	=	3D	13	7E
65	"	"	22	76	7F
66					
67	a	a	61		81
68	b	b	62		82
69	c	c	63		83
70	d	d	64		84
71	e	e	65		85
72	f	f	66		86
73	g	g	67		87
74	h	h	68		88
75	i	i	69		89
76					
77					
78					
79					
80					
81					
82					
83	j	j	6A		91
84	k	k	6B		92
85	l	l	6C		93
86	m	m	6D		94
87	n	n	6E		95
88	o	o	6F		96
89	p	p	70		97
90	q	q	71		98
91	r	r	72		99
92					
93					
94					
95					
96					
97					
98					
99	~	~	7E		A1

Table F-13. Collating Sequence (cont)

Collating Sequence	EBCDIC Character	ANSI Character	Internal and ASCII Tape Code (Hex)	Tape Code BCD (Octal)	Tape Code EBCDIC (Hex)
100	s	s	73		A2
101	t	t	74		A3
102	u	u	75		A4
103	v	v	76		A5
104	w	w	77		A6
105	x	x	78		A7
106	y	y	79		A8
107	z	z	7A		A9
108					
109					
110					
111					
112					
113					
114					
115					
116					
117					
118					
119					
120					
121					
122					
123					
124					
125					
126					
127					
128					
129					
130	{	{ Left Brace	7B		C0
131	A	A	41	61	C1
132	B	B	42	62	C2
133	C	C	43	63	C3
134	D	D	44	64	C4
135	E	E	45	65	C5
136	F	F	46	66	C6
137	G	G	47	67	C7
138	H	H	48	70	C8
139	I	I	49	71	C9
140					
141					
142	ʃ	] Right Bracket	5D	32	CC
143					

Table F-13. Collating Sequence (cont)

Collating Sequence	EBCDIC Character	ANSI Character	Internal and ASCII Tape Code (Hex)	Tape Code BCD (Octal)	Tape Code EBCDIC (Hex)
144	⑂	<	3C	72	CE
145					
146	}	}Right Brace	7D		D0
147	J	J	4A	41	D1
148	K	K	4B	42	D2
149	L	L	4C	43	D3
150	M	M	4D	44	D4
151	N	N	4E	45	D5
152	O	O	4F	46	D6
153	P	P	50	47	D7
154	Q	Q	51	50	D8
155	R	R	52	51	D9
156					
157					
158					
159					
160					
161					
162					
163					
164	S	S	53	22	E2
165	T	T	54	23	E3
166	U	U	55	24	E4
167	V	V	56	25	E5
168	W	W	57	26	E6
169	X	X	58	27	E7
170	Y	Y	59	30	E8
171	Z	Z	5A	31	E9
172					
173					
174	⑀	>	3E	57	EC
175					
176					
177					
178	0	0	30	12	F0
179	1	1	31	01	F1
180	2	2	32	02	F2
181	3	3	33	03	F3
182	4	4	34	04	F4
183	5	5	35	05	F5
184	6	6	36	06	F6
185	7	7	37	07	F7
186	8	8	38	10	F8
187	9	9	39	11	F9
188	ǀ	\	5C	35	FA

SAMPLE DUMPS

The following are lists of dumps that include an unformatted dump, a formatted character dump, and a hexadecimal dump.

Unformatted Dump Example

Refer to figure F-6.

```
NAM        DELMOD     DELETE MODULES VER 1.0 OF SUP 06/15/73      DEL00001
NAM        TAPUTL     TAPE UTILITY VER 1.0 OF SUP 06/15/73        TAP00001
NAM        FNN        FIND FUNCTION VER 1.0 OF SUP 06/15/73       FNN00001
NAM        SCAN       SCAN COMMAND VER 1.0 OF SUP 06/15/73        SCA00001
NAM        OPENIO     OPEN FUNCTION VER 1.0 OF SUP 06/15/73       OPE00001
NAM        RDWTR      READ AND WRITE VER 1.0 OF SUP 06/15/73      RDW00001
NAM        LIOC       LOGICAL I/O VER 1.0 OF SUP 06/15/73         LI000001
NAM        COPY       COPY FUNCTION VER 1.0 OF SUP 06/15/73       COP00001
NAM        EXIT       EXIT SYSTEM VER 1.0 OF SUP 06/15/73         EXI00001
```

Figure F-6. Unformatted Dump

Formatted Character Dump Example

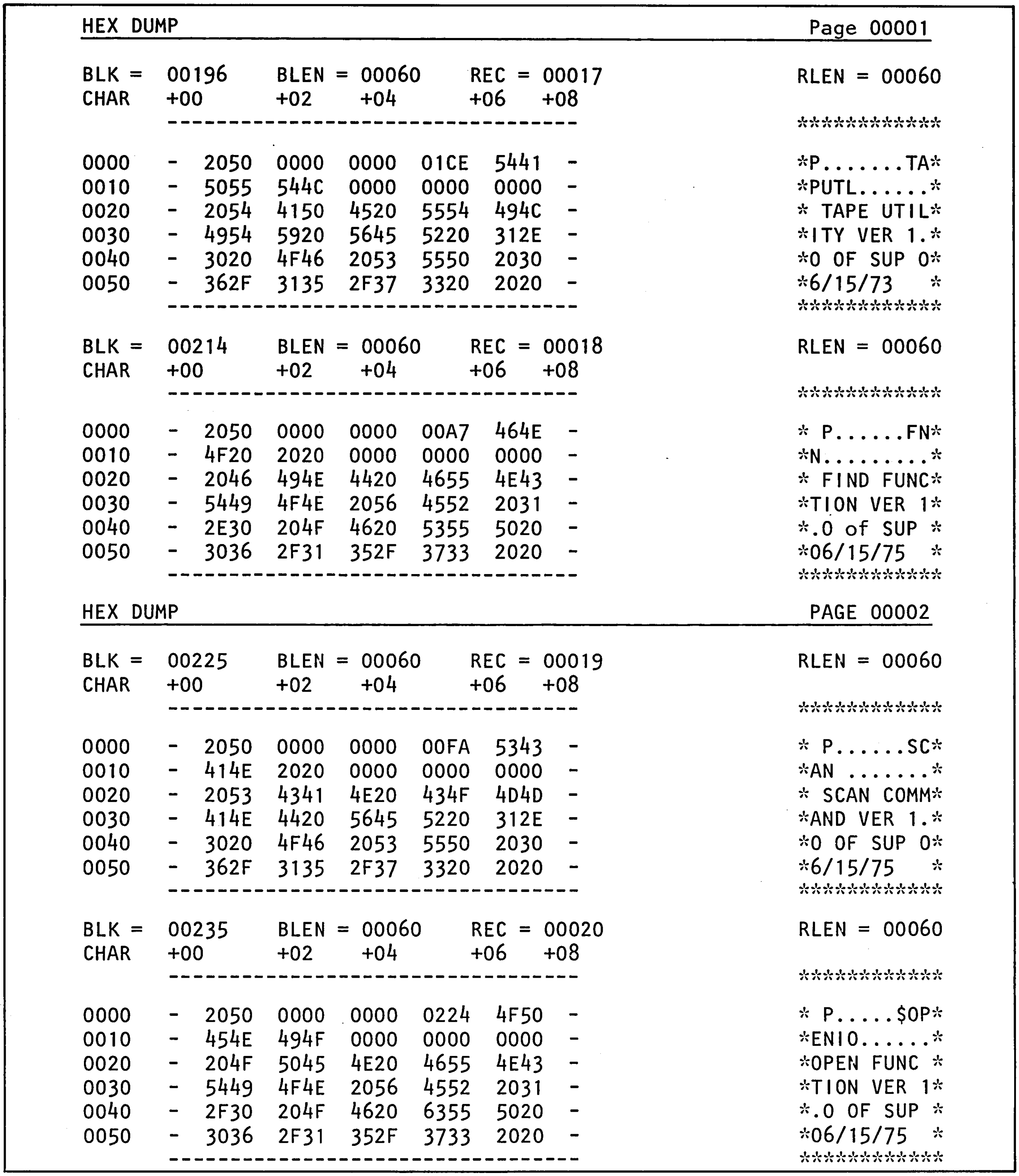
Refer to figure F-7.

```
ASCII     DUMP                                                            PAGE 00001

BLK =     00001    BLEN= 00080    REC = 00001    RLEN = 00080
CHAR POSITION
           00     05     10     15     20     25     30     35     40     45     49
          -------------------------------------------------------------------

0000      -          NAM    DELMOD         DELETE MODULES VER 1.0 OF
0050      -   SUP 06/15/73             DEL00001
          -------------------------------------------------------------------
BLK =     00071    BLEN = 0080    REC = 00002    RLEN = 00080
CHAR      POSITION
           00     05     10     15     20     25     30     35     40     45     49
          -------------------------------------------------------------------

0000 =               NAM    TAPUTL         TAPE UTILITY VER 1.0 OF SUP
0050          06/15/73           TAP00001
          -------------------------------------------------------------------

BLK =     00563    BLEN = 00080   REC = 00003    RLEN = 00080
CHAR      POSITION
           00     05     10     15     20     25     30     35     40     45     49
          -------------------------------------------------------------------

0000      -          NAM    FNN            FIND FUNCTION VER 1.0
0050          OF SUP 06/15/73         FNN00001
          -------------------------------------------------------------------
```

Figure F-7. Formatted Character Dump

Hexadecimal Dump Example

Refer to figure F-8.



```
HEX DUMP                                                Page 00001

BLK =   00196    BLEN = 00060     REC = 00017           RLEN = 00060
CHAR    +00      +02    +04       +06   +08
        ---------------------------------               ************

0000    -  2050  0000  0000  01CE  5441  -              *P.......TA*
0010    -  5055  544C  0000  0000  0000  -              *PUTL......*
0020    -  2054  4150  4520  5554  494C  -              * TAPE UTIL*
0030    -  4954  5920  5645  5220  312E  -              *ITY VER 1.*
0040    -  3020  4F46  2053  5550  2030  -              *0 OF SUP 0*
0050    -  362F  3135  2F37  3320  2020  -              *6/15/73   *
        ---------------------------------               ************

BLK =   00214    BLEN = 00060     REC = 00018           RLEN = 00060
CHAR    +00      +02    +04       +06   +08
        ---------------------------------               ************

0000    -  2050  0000  0000  00A7  464E  -              * P......FN*
0010    -  4F20  2020  0000  0000  0000  -              *N.........*
0020    -  2046  494E  4420  4655  4E43  -              * FIND FUNC*
0030    -  5449  4F4E  2056  4552  2031  -              *TION VER 1*
0040    -  2E30  204F  4620  5355  5020  -              *.0 of SUP *
0050    -  3036  2F31  352F  3733  2020  -              *06/15/75  *
        ---------------------------------               ************
HEX DUMP                                                PAGE 00002

BLK =   00225    BLEN = 00060     REC = 00019           RLEN = 00060
CHAR    +00      +02    +04       +06   +08
        ---------------------------------               ************

0000    -  2050  0000  0000  00FA  5343  -              * P......SC*
0010    -  414E  2020  0000  0000  0000  -              *AN .......*
0020    -  2053  4341  4E20  434F  4D4D  -              * SCAN COMM*
0030    -  414E  4420  5645  5220  312E  -              *AND VER 1.*
0040    -  3020  4F46  2053  5550  2030  -              *0 OF SUP 0*
0050    -  362F  3135  2F37  3320  2020  -              *6/15/75   *
        ---------------------------------               ************

BLK =   00235    BLEN = 00060     REC = 00020           RLEN = 00060
CHAR    +00      +02    +04       +06   +08
        ---------------------------------               ************

0000    -  2050  0000  0000  0224  4F50  -              * P.....$OP*
0010    -  454E  494F  0000  0000  0000  -              *ENIO......*
0020    -  204F  5045  4E20  4655  4E43  -              *OPEN FUNC *
0030    -  5449  4F4E  2056  4552  2031  -              *TION VER 1*
0040    -  2F30  204F  4620  6355  5020  -              *.0 OF SUP *
0050    -  3036  2F31  352F  3733  2020  -              *06/15/75  *
        ---------------------------------               ************
```

Figure F-8. Hexadecimal Dump

SUP Commands

The following two commands are used to enter and exit from SUP.

- *SUPLP

  This calls the SUP load program to load SUP from the program library.

- EXIT

  This returns to the operating system. Tape files which are not closed will rewind and unload.

Tapes and printer files must be opened before the units can be accessed. I, O, VF, or PR parameters must immediately follow OPEN or CLOSE. Only one logical unit of each type (I, O, VF, or PR) may be open at the same time. Other parameters can be in any order. Only the underlined portion of a parameter needs to be entered. In the following commands, nn is the logical unit number and nnnnn is the number of physical records.

Write header record on tape.

TERM—terminate initialize (INIT repeats until terminated)

$$\underline{\text{OP}}\text{EN,}\quad \left\{\begin{matrix}\underline{\text{I}}\text{nn}\\ \underline{\text{O}}\text{nn}\\ \underline{\text{V}}\text{Fnn}\\ \underline{\text{P}}\text{Rnn}\end{matrix}\right\}\text{,}\ \left[\begin{matrix}\underline{\text{SL}}\\ \underline{\text{BL}}\\ \underline{\text{NL}}\end{matrix}\right]\text{,}\ \left[\begin{matrix}\underline{\text{V}}\\ \underline{\text{VB}}\\ \underline{\text{F}}\\ \underline{\text{FB}}\\ \underline{\text{U}}\end{matrix}\right]\text{,}\ \left[\begin{matrix}\underline{\text{B}}\\ \underline{\text{E}}\\ \underline{\text{A}}\end{matrix}\right]\text{,}\ [\underline{\text{SF}}\text{nn ,}]\ [\underline{\text{BR}}\text{nnnnn,}]\ \left[\begin{matrix}\underline{\text{PN}}\text{AM(='XXXXXX')}\\ \underline{\text{TN}}\text{AM(='XXXXXX')}\\ \underline{\text{SN}}\text{AM(='XXXXXX')}\\ \underline{\text{NA}}\text{M (='XXXXXX')}\end{matrix}\right]\ [\underline{\text{LB}}\text{nnnnn,}]$$

$$[\underline{\text{LR}}\text{nnnnn,}]\ [\underline{\text{LC}}\text{nnn}]$$

$$\underline{\text{CL}}\text{OSE,}\quad \left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OUT}}\\ \underline{\text{VF}}\\ \underline{\text{PR}}\end{matrix}\right\}\text{,}\quad \left[\begin{matrix}\underline{\text{RW}}\\ \underline{\text{UN}}\text{LOAD}\\ \underline{\text{LE}}\text{AVE}\\ \underline{\text{EO}}\text{V}\end{matrix}\right]$$

$$\underline{\text{BS}}\text{PACE,}\quad \left\{\begin{matrix}\underline{\text{I}}\\ \underline{\text{O}}\\ \underline{\text{VF}}\end{matrix}\right\}\text{,}\quad [\text{nnnnn}]$$

$$\underline{\text{DU}}\text{MP,}\quad [\underline{\text{FC}}\text{nn,}]\quad [\underline{\text{RC}}\text{nnnn,}]\quad \left[\begin{matrix}\underline{\text{H}}\text{ex}\\ \underline{\text{C}}\text{har}\end{matrix}\right]\text{,}\quad \left[\begin{matrix}\underline{\text{F}}\text{or}\underline{\text{M}}\text{at}\\ \underline{\text{U}}\text{n}\underline{\text{F}}\text{ormat}\end{matrix}\right]\text{,}\quad \left[\begin{matrix}\underline{\text{PN}}\text{AM (='XXXXXX')}\\ \underline{\text{TN}}\text{AM (='XXXXXX')}\\ \underline{\text{SN}}\text{AM (='XXXXXX')}\\ \underline{\text{NA}}\text{M (='XXXXXX')}\end{matrix}\right]$$

$$\underline{\text{PR}}\text{INT,}\quad [\underline{\text{FC}}\text{nn,}]\quad [\underline{\text{RC}}\text{nnnn,}]\quad \left[\begin{matrix}\underline{\text{US}}\\ \underline{\text{TS}}\end{matrix}\right]\text{,}\quad \left[\begin{matrix}\underline{\text{PN}}\text{AM (='XXXXXX')}\\ \underline{\text{TN}}\text{AM (='XXXXXX')}\\ \underline{\text{SN}}\text{AM (='XXXXXX')}\\ \underline{\text{NA}}\text{M (='XXXXXX')}\end{matrix}\right]$$

$$\underline{\text{CO}}\text{PY,}\quad [\underline{\text{RC}}\text{nnnn,}]\quad [\underline{\text{FC}}\text{nn,}]\quad \left[\begin{matrix}\underline{\text{PN}}\text{AM (='XXXXXX')}\\ \underline{\text{TN}}\text{AM (='XXXXXX')}\\ \underline{\text{SN}}\text{AM (='XXXXXX')}\\ \underline{\text{NA}}\text{M (='XXXXXX')}\end{matrix}\right]$$

$$\underline{\text{VE}}\text{RIFY,}\quad [\underline{\text{RC}}\text{nnnn,}]\quad [\underline{\text{FC}}\text{nn,}]\quad \left[\begin{matrix}\underline{\text{H}}\\ \underline{\text{C}}\end{matrix}\right]\text{,}\quad \left[\begin{matrix}\underline{\text{UF}}\\ \underline{\text{FM}}\end{matrix}\right]\text{,}\quad \left[\begin{matrix}\underline{\text{PN}}\text{AM (='XXXXXX')}\\ \underline{\text{TN}}\text{AM (='XXXXXX')}\\ \underline{\text{SN}}\text{AM (='XXXXXX')}\\ \underline{\text{NA}}\text{M (='XXXXXX')}\end{matrix}\right]\quad [\text{INIT,}]\quad [\text{Onn,}]\quad \left[\begin{matrix}\underline{\text{E}}\\ \underline{\text{A}}\\ \underline{\text{B}}\end{matrix}\right]$$

SUP COMMAND PARAMETERS

The following parameters are used in the SUP commands:

A

ASCII mode or ANS VOL 1 labels

B

BCD mode for seven-track only

BL

bypass labels

BR

bypass records before processing

C

character dump

E

EBCDIC mode or EBCDIC standard volume labels

EOV

end of volume writes trailer labels, any end of file labels, double tape mark, and rewinds and unloads tape

F

fixed-length records

FB

fixed-length, blocked records

FC

file count; number of files to be dumped

FM

formatted dump 100 characters per line

H

hexadecimal dump

I and In

input logical unit

LB

length of block (decimal); maximum is 32767

LC

line count per page; default is 56

LEAVE

leave tape positioned; labels and tape marks are written

LR

length of records (decimal); maximum is 32767

NAM

select NAM statements

NL

no labels

O and OUT

output logical unit

PNAM

position to name

PR

print logical unit

RC

record count

RW

rewind to load point; writes tape mark and required end-of-file labels before rewinding

SF

skip files before processing

SL

standard tape labels

SNAM

select and process by specific name

TNAM

terminate processing at name

U

undefined record format

UF

unformatted dump in character mode

UN

rewind and unload; writes tape mark and required end-of-file labels

US

USASCI printer control character in first character position

V

variable-length record

VB

variable-length, blocked records

VF

verify file logical unit

APPENDIX G

ASSEMBLER COMMANDS

Assembler Commands

INTRODUCTION

The macro assembler for the 1700 Computer System is a three-pass assembler which can convert source language input, including macro instructions, to relocatable output and can generate list output. The source programs are written with symbolic machine, pseudo, and macro instructions.

Macro definitions may be defined by the user in the source program, or they may be placed on a separate macro library.

Input is from the standard input device, binary output is to the standard output device, and list output is to the standard list device.

The following describe functions occurring in each pass of the assembler:

- Pass 1

  Programmer-defined macros are processed, and appropriate tables are built. Whenever a macro instruction is encountered, the macro skeleton with actual parameters substituted is inserted into the source input on the mass storage device.

  The source input is copied onto the mass storage device.

  Sequence numbers of the input source images are checked.

- Pass 2

  Each source image on the mass storage device is read, and pass 2 errors are listed as they occur.

  Conditional assembly pseudo instructions are processed.

  Symbol and external tables are built.

- Pass 3

  Each image is read, and pass 3 errors are listed.

  List and relocatable binary output are generated according to the input options.

- TABLST

  TABLST prints and punches the entry points and external images. The transfer image is punched.

  An EOF image is output to the next load-and-go sector on mass storage.

  A symbol table listing is given.

- XREF

  XREF creates and prints the cross-reference lists.

INSTRUCTION FORMAT

The following paragraphs discuss the ASSEM instruction format.

Source Program

The number of independent subprograms which comprise a source program is limited only by available space. Each subprogram may be assembled independently, or several may be assembled as a group. The main subprogram of a group is the one to which initial control is given; it need not be the first subprogram. The last subprogram of a group must be followed by the MON pseudo instruction which indicates the end of assembly and return to the operating system.

Communication between subprograms is accomplished by the subprogram linkage pseudo instructions and by the use of common and data storage.

At execution time, the entry point named in the END pseudo instruction specifies the entry point to which initial control passes. A jump to the dispatcher or an exit request signals return of control to the operating system on job completion. EXIT or a jump to the dispatcher must be the last statement to be executed.

Source Statement

A source statement consists of location, instructions, address, remarks, and sequence fields. The first four fields may not exceed 72 characters; in that limitation they are free field. The sequence field is used when the source image is 80 characters; it is restricted to columns 73 through 80.

Each field is terminated by a tab ($B; paper tape only), carriage return (end-of-statement mark), or blanks. Any number of blanks may separate fields. A carriage return is always the end-of-statement mark on paper tape.

LOCATION FIELD

The location field of a source statement must begin in column 1.

This field is used to specify a labeled (label starting in column 1) or an unlabeled (blank or tab in column 1) statement.

The statement label is a symbolic name which consists of from one to six alphanumeric characters; the first must be alphabetic. Characters in excess of six are ignored. A two-character name makes the most efficient use of storage and assembly time.

Examples of statement label are:

- LOOP1—legal
- 123456—illegal;first character is numeric
- P1—legal
- A123456—legal; only A12345 is processed

REMARKS

An asterisk in column 1 of the location field specifies that the source statement is a remark. Comments, written in columns 2 through 72, are printed with the assembly list output but have no effect on the object program. An asterisk elsewhere in the location field is illegal. Remarks may also follow the address field of any instruction. There must be at least one blank separating the address field from the remarks.

INSTRUCTION

The instruction field begins to the right of the location field and must be separated from it by at least 1 blank character or a tab. If the location field contains no label (blank or tab on column 1), the operation code may begin in column 2.

The operation code field contains the three-letter instruction codes for machine and pseudo instructions; or it contains macro instructions, which may be up to six characters. Certain instructions may be followed by a one-character terminator.

ADDRESS FIELD

The address field, which begins to the right of the operation code field, is separated from it by at least one blank character or a tab. It is terminated by a blank or tab or by the 72nd character of the source statement. Exceptions are the macro instructions which may have a continuation line and the pseudo instruction ALF (page G-23).

This field contains an address expression which consists of an operand or string of operands joined by arithmetic operators. It may also contain a series of operands separated by commas. An operand may be any of the following:

- Symbolic name
- Numeric constant
- One of the special characters: * A Q M 0 I B

Symbolic Operand

A symbolic name used as an operand in the address field must be defined in one of the following ways.

- Label in the location field of any machine instruction
- Label in the location field of any macro instruction
- Label in the location field of constant declaration pseudo instructions (ADC, ALF, NUM, DEC, VFD)
- Symbolic name in the address field of the pseudo instructions (EXT, COM, DAT, BSS, BZS, EQU)

A defined symbolic name references a specific location in memory. It may be relocatable or absolute. A relocatable symbol refers to a location that may be relocated during loading.

Storage is divided into three areas: program, data, and common. These areas are defined at assembly time, and the initial location of each is set to a relocation address of zero. The object code produced by the assembler contains addresses which are modified by a relocation factor to produce the actual address in memory.

A symbol is program relocatable if it references a location in the subprogram, data relocatable if it references a location in data storage, and common relocatable if it references a location in common storage. All other symbols are absolute. A symbol is made absolute by equating it to a number, an arithmetic expression, or another absolute symbol.

In all cases, a symbolic label and a symbol defined by BSS or BZS take the relocation and value of the current location counter. The location counter of a program is originally program relocatable; however, its relocation may be changed by the ORG instruction.

An address expression which includes more than one operand must reference only one relocatable area. Terms of different relocation types must reduce to one relocatable area or to an absolute address. When the address mode of an instruction is made one-word relative by an asterisk terminator, the relocation type of the address expression must agree with the type of the current location counter.

A symbolic operand may be preceded by a plus or a minus sign. If preceded by a plus or no sign, the symbol refers to its associated value; if preceded by a minus, the symbol refers to the ones complement of its associated value. When an expression contains more than one symbol, the final sign of the expression is the algebraic sum of the operands. The following paragraphs give examples of symbolic operands. In the examples, RT relocation type of current location counter are P (program relocatable), C (common relocatable), D (data relocatable), and A (absolute address).

```
RT   LABEL     OPERATION     ADDRESS                                          COMMENTS


                 COM         COM1, COM2
                 DAT         DAT1, DAT2
                 EQU         D(1),E(3),G(E-D),H($1000),I(DAT1)
P                BZS         A,B,C
P                BZS         J,K(10)
```

Figure G-1. Assembler Coding, Example 1

In figure G-1, the symbols D, E, G, and H are absolute. DAT1, DAT2, and I are data relocatable. COM1 and COM2 are common relocatable. A, B, C, J, and K are program relocatable.

```
RT   Label     Operation     Address                                          Comments


P    START       ADC         0
P                LDA*        START
P                STA*        DAT1                                             (Error)
P                STA*        COM1                                             (Error)
```

Figure G-2. Assembler Coding, Example 2

The errors in figure G-2 resulted because the relocation types of the symbols in the address field do not match that of the location counter, and the one-word relative address mode was requested by an asterisk terminator.

```
RT   Label     Operation     Address                            Comments


P                LDA+                                           (Not an error)
```

Figure G-3. Assembler Coding, Example 3

Relocations need not match as in figure G-3 when the mode is two-word absolute.

```
RT    LABEL     OPERATION    ADDRESS                   COMMENTS


P                LDA         START                     (Okay, relocations match)
P                LDA         COM1                      (Not an error)
```

Figure G-4. Assembler Coding, Example 4

The assembler changes the instruction in figure G-4 to a two-word absolute because relocations do not match, but no error is indicated.

```
RT    LABEL     OPERATION    ADDRESS                                COMMENTS


P                LDA         COM2-DAT1+COM1-D+E-COM2+START-K+DAT2
```

Figure G-5. Assembler Coding, Example 5

The address expression in figure G-5 results in a common relocation type; all other relocations cancel out (refer to address expressions).

```
RT    LABEL     OPERATION    ADDRESS                                COMMENTS


                 ORG         DAT1
```

Figure G-6. Assembler Coding, Example 6

In figure G-6, ORG changes the relocation of the location counter to data.

```
RT    LABEL     OPERATION    ADDRESS                                COMMENTS


D                LDA*        START                                  (Error)
D                STA*        DAT2+9
                 ORG*
```

Figure G-7. Assembler Coding, Example 7

ORG* returns the location counter to the original relocation (figures G-7 and G-8).

```
RT    LABEL    OPERATION    ADDRESS                        COMMENTS

P                LDA*       START                          (Not an error)
                 ORG        H
A                LDA*       START                                  (Error)
A                STA*       DAT1                                   (Error)
A                LDA*       $1001
A                STA-       B
                 ORG*
                 END
```

Figure G-8. Assembler Coding, Example 8

Numeric Operand

A numeric operand in the address field may be decimal or hexadecimal. A decimal number is represented by up to five decimal digits and must be in the range ±32767. A hexadecimal number is represented by a dollar sign and not more than four hexadecimal digits in the range of ± 7FFF (hexadecimal operands in the NUM pseudo instruction may be in the range of ± FFFF).

Numeric operands in the address field may be preceded by a plus or a minus sign. If a plus or no sign is specified, the binary equivalent of the number is the value used; a minus means the one's complement of the binary equivalent is the value.

A numeric operand has no relocation type; it is always absolute.

Address Expression

An address expression may be a single operand or a string of operands joined by the following arithmetic operators.

- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)

Arithmetic operators may not follow each other without an intervening operand. Parentheses are not permitted for grouping terms.

The asterisk has an additional meaning as an operand. When it is used as the multiplication operator (refer to special characters), it must be immediately preceded by an operand which may be another asterisk. When the asterisk is used as an operator, only one of its associated operands may be relocatable.

The slash, used as the division operator, must be between two operands. The operand which follows may not be zero or relocatable.

An external name may be used in an address expression only as a single operand. Arithmetic operators preceding or following an external operand are illegal (figure G-9).

```
        NAM   EXAMPL
        COM   A,B
        EQU   C(1),D(5)
        EXT   G
        BZS   E(10),F
START   LDA   D-C/5+**2
        ADD   A-B/2
        ADD   E+5
        STA   G
        END
```

Figure G-9. Assembler Coding, Example 9

The first asterisk in the LDA instruction refers to the value of the current location counter.

The instructions in figure G-10 are illegal assuming the same pseudo instructions precede the START.

```
START   LDA   D-C**5+2        *5 has no intervening operator
        ADD   A-2/B           Division by relocatable operand
        ADD   E*F             Both operands are relocatable
        STA   G+5             An external must stand alone
```

Figure G-10. Assembler Coding, Example 10

The hierarchy for the evaluation of arithmetic expressions is:

- / or * (Evaluated first)
- + or - (Evaluated next)

Expressions which contain operators at the same level are evaluated from left to right. The expression A/B+C*D is evaluated algebraically as follows:

A/B+(C)(D)

The expression should not be evaluated as any of the following:

- $\frac{(A)(D)}{B+C}$
- $\frac{A}{(B+C)(D)}$
- $\frac{A}{B+(C)(D)}$

Parentheses may not be used for grouping operands. The algebraic expression (A-D)(B+C/E) must be specified as follows:

A*B+A*C/E-D*B-D*C/E

The following expression is illegal.

(A-D)*(B+C/E)

Division in an address expression always yields a truncated result; therefore, 11/3=3. The expression A*B/C may result in a value different from B/C*A. For example, if A=4, B=3, and C=2 then:

A*B/C=4*3/2=6

However:

B/C*A=3/2*4=4

All expressions are evaluated modulo 2^{15}-1: An address expression which consists solely of numeric operands is absolute. If an expression contains symbolic operands, the final relocation for the expression is determined by the relocations of the symbolic operands. If the relocation of the operands is expressed by the following terms, the final relocation is the algebraic sum of the relocation terms:

- ±P—Positive or negative program relocation
- ±C—Positive or negative common relocation
- ±D—Positive or negative data relocation

The relocation must reduce to one of the relocation terms or to zero. If it is zero, the location is absolute (table G-1).

Table G-1. Relocation Examples

Source Statements	Relocation Formula
COM A,B	
DAT C,D	
EQU E(1),F(D)	
STRT LDA B+C-E*2-A-D	+C+D-C-D=0 (absolute)
LDA B+D-F+STRT-A-C	+C+D-D+P-C-D=P-D(illegal)
LDA B+D-E+STRT-A-C	+C+D+P-C-D=P (program)
LDA B-D-A	+C-D-C=-D (negative data)

Special Characters

Special characters may be used as operands in the address field of a source statement. Their definition may not be changed by the user. The three classes of special characters are storage, register, and index. (table G-2).

Storage class characters (*, I) reference storage locations. The asterisk refers to the location of the current instruction. For a word instruction, an asterisk references the location of the first word. Special character I refers to location FF_{16}. I is the only indexing character that may stand alone as an operand with storage reference instructions. It may not be defined as a location symbol in a program.

Table G-2. Special-Character Operands

Class	Character	Referenced Location
Storage	* I	Current location counter Location FF_{16}
Register	A Q M 0	A register Q register Mask register Destination registers
Index	Q I B	Index 1; Q register Index 2; location FF_{16} Index 1 plus index 2

The register class characters (A, Q, M, and 0) are used only with interregister transfer instructions. They refer to the A, Q, and M (mask) registers. Character 0 sets the destination registers to zero (page G-17). Refer to table G-3.

Table G-3. Register Class Characters

Instruction	Function
SET A,Q,M	Set A, Q, and mask registers to ones
TRA Q	Transfer contents of A register to Q register
LAM M	Transfer logical product of A and mask register to mask register

Index class characters (Q, I, and B) are used in conjunction with an address expression to refer to the index registers. Any one character may follow an address expression. It is separated from the expression by a comma with no intervening blank. Indexing may be used only with storage reference instructions.

- Q—contents of Q register are added to contents of the expression to form the actual address
- I—contents of location FF_{16} are added to contents of address expression to form the actual address
- B—contents of Q register are added to address expression, and this sum is added to contents of FF_{16} to produce the actual address

For examples of the index class characters, refer to table G-4.

Table G-4. Sample Index Class Characters

Address Field	Legality	Function
LOC1,B	Legal	Contents of registers Q and FF_{16} and the contents of LOC1 are added to produce the actual address
,,I	Illegal	Character following first comma is assumed to be index character
TAG2,Q,I	Illegal	Only one index notation allowed
Q	Illegal	Unless Q has been previously defined as a location symbol or is being used with the interregister transfer instruction, it must follow a location symbol
TAG3,I	Legal	Contents of FF_{16} and TAG3 are added to produce the actual address

COMMENT FIELD

The address field is followed by the comment field, which is used for remarks. Remarks do not affect the object code, but are printed as part of the list output. The comment field terminates at column 72 or with a carriage return (paper tape). Blanks are permitted in the comment field.

SEQUENCE FIELD

When the input image is 80 characters, columns 73 through 80 are available for sequencing; 73 through 75 may be used for program identification, and 76 through 80 for a sequence number.

Sequence numbers are checked for errors only if the input image is 80 characters. Each sequence number must be greater than or equal to the previous sequence number. The value of a character in the sequence number is in ASCII code except that a blank is treated as zero.

MACHINE INSTRUCTIONS

Machine instructions represented by a three-letter mnemonic code are divided into six classes.

- Group A storage reference — shift
- Group B storage reference — skip

- Register reference — interregister transfer

Storage reference instructions result in one or two machine words, depending on modification. Other machine instructions result in one machine word.

The function of each machine instruction is discussed in detail in the COS manual sections.Starting on page G-42, a list of the machine instructions is given in the order in which they are discussed in the following paragraphs.

Storage Reference Instructions

Group A and B storage reference instructions use storage addresses as operands or as operand addresses. Group B instructions include jump instructions and may not use the constant mode of addressing.

ADDRESS MODES

Group A storage reference instructions allow three modes of addressing: absolute, relative, and constant. Group B does not allow the use of the constant mode, but is otherwise the same as group A.

Special characters designate the mode of addressing, the number of words for the instruction, and indirect addressing (table G-5).

Table G-5. Special Characters for Addressing Mode

Character	Description
*	Asterisk, as the last character of operation code, specifies relative addressing in a one-word instruction
-	Minus, as the last character of operation code, specifies absolute addressing in a one-word instruction
+	Plus, as the last character of operation code, specifies absolute addressing in a two-word instruction
=	Equal sign, as the first character in address field preceding a constant, indicates constant addressing; instruction is always two words
()	Parentheses enclosing the address expression indicate indirect addressing

If no character is specified as a terminator to the operation code, two-word relative addressing is assumed with the following exceptions.

- If a constant is specified, the constant mode is assumed.

- If the relocation type of the address expression differs from the relocation type of the location counter, two-word absolute addressing is assumed.

- If a nonrelative external is referenced, absolute addressing is assumed.

The machine language format which results from a storage reference instruction is illustrated as follows.

First Word

The first-word machine language format is:

15	11	10	9	8	7 0
f	r	d	q	i	

Parameters for the first word are:

f—4-bit operation code is described previously

r—specifies relative addressing

d—specifies indirect addressing

q—index register 1 flag; specifies adding contents of Q register to address

i—index register 2 flag; specifies adding contents of storage register FF_{16} to address

—8-bit field; may be relative or absolute address for one-word instructions; when zero, indicates two-word instruction

Second Word

When the second word is used, it appears as follows:

15 0
c

The parameter for the second word is:

c—16-bit field for constant addressing or relative address; when it contains relative address, bit 15 is the sign

Second Word — Alternate

15	14 0
b	m

The parameters are:

b—indirect address bit

m—memory address

Address expressions are evaluated modulo 2^{15}-1.

ABSOLUTE ADDRESSING

The value of the address expression of a one-word absolute instruction must be nonrelocatable. The evaluated result is stored in 8 bits of the machine word. If this value is greater than 256, it is flagged as an error. If the 8-bit △ field is zero, two machine words are assumed regardless of the operation code terminator; no error message is printed. If the address expression is enclosed in parentheses for indirect addressing, bit 10 of the first word is set to 1. Refer to the following examples.

One-Word, Direct Addressing

The instruction is:

LDA- e

The following machine word shows the instruction format:

15	11	10	9	8	7 0
LDA	0	0	0	0	e

One-Word, Indirect Addressing

The instruction is:

ADQ- (e)

The machine word format is:

15	11	10	9	8	7 0
ADQ	0	1	0	0	e

The value of the address expression of a two-word absolute instruction is stored in the least-significant bits of the second word. If the expression is enclosed in parentheses for indirect addressing, bit 15 of the second word is set to 1. The indirect address bit 10 in the first word is always set to 1 when two-word absolute addressing is specified whether the address expression is specified as indirect or direct. This indicates that the address expression is in the second word. The 8-bit △ field of the first word is set to zero for two-word instructions. Refer to the following examples.

Two-Word, Direct Addressing

The following instruction is used:

EOR+ e

The machine words for the two-word, direct addressing are:

15	11	10	9	8	7 0
EOR	0	1	0	0	00

15	14 0
0	e

Two-Word, Indirect Addressing

The instruction is:

AND+ (e)

The following machine words show the format for the two-word, indirect addressing:

15	11	10	9	8	7 0
AND	0	1	0	0	00

15	14 0
1	e

RELATIVE ADDRESSING

When one-word relative addressing is specified, the value of the current location counter is subtracted (16-bit ones complement arithmetic) from the evaluated address expression. The result is placed in the 8-bit field. If the value of the result is outside the range $\pm 7F_{16}$, an error condition is flagged. An error condition is also flagged if the relocation type of the address expression differs from that of the location counter. If the 8-bit △ field is zero, two words are assumed, regardless of the operation code terminator. No error message is printed for this condition. Refer to the following examples.

One-Word, Direct Addressing

The instruction is:

AND* e

The machine word format is:

15	11	10	9	8	7 0
AND	1	0	0	0	e-*

One-Word, Indirect Addressing

The instruction is:

MUI* (e)

The machine word format for one-word, indirect addressing is:

15	11	10	9	8	7 0
MUI	1	1	0	0	e-*

In the expression e-*, the asterisk indicates the value of the current location counter.

When a two-word instruction is specified, the value of the current location counter plus one is subtracted (using 16-bit ones complement arithmetic) from the value of the address expression to obtain the 16-bit second word. If the relocation type of the address expression differs from that of the location counter, and the address does not reference an external, the assembler forces a two-word absolute instruction. If the address expression is an external reference, the instruction is absolute or relative depending on the definition of the external. Refer to the following examples.

Two-Word, Direct Addressing

The instruction is:

LDQ e

The following format applies to the machine words:

15	11	10	9	8	7 0
LDQ	1	0	0	0	00

15 0
e-*-1

Two-Word, Indirect Addressing

The instruction is:

LDA (e)

The machine word formats are:

15	11	10	9	8	7 0
LDA	1	1	0	0	00

15 0
e-*-1

In the expression, e-*-1, the asterisk indicates the value of the current location counter.

CONSTANT ADDRESSING

Constant addressing may be used only for group A storage reference instructions. Constants in the address field are preceded by an equal sign and a one-letter code. A constant may be one as listed in table G-6. Refer to table G-7 for examples.

Table G-6. Constants Used in Addressing

Code	Type	Meaning
A	aa	Two alphanumeric characters
N	+ddddd	Five-digit decimal number with or without a leading sign
N	+$hhhh	Four-digit hexadecimal number preceded by $, with or without a sign
X	e	Address expression evaluated modulo 2^{15}-1
X	(e)	Address expression evaluated modulo 2^{15}-1, with bit 15 set

Table G-7. Constant Addressing Examples

Operation	Description
DVI =N$1000	Hexadecimal constant
ADD =N-12345	Decimal constant
LDA =AXY	ASCII constant
AND =XTAG1+5	Address expression constant

An instruction which contains a constant in the address field results in two machine words. Refer to the following example.

The instruction is:

DVI =nc

Where:

n—the code

c—the constant

The machine word formats are:

15	11	10	9	8	7 0
DVI	0	0	0	0	0

15 0
c

DATA TRANSMISSION INSTRUCTIONS

Refer to table G-8.

Table G-8. Data Transmission Instructions

Operation	Description
STQ (F = 4)	Store Q; store the contents of the Q register in the storage location specified by the effective address; contents of Q are not altered
STA (F = 6)	Store A; store the contents of the A register in the storage location specified by the effective address; contents of A are not altered
SPA (F = 7)	Store A, parity to A; store the contents of the A register in the storage location specified by the effective address; clear A if the number of "1" bits in A is odd; set A equal to 0001_{16} if the number of "1" bits in A is even; contents of A are not altered if the write into storage is aborted because of parity error or protect fault
LDA (F = C)	Load A; load the A register with the contents of the storage location specified by the effective address; contents of the storage location are not altered
LDQ (F = E)	Load Q; load the Q register with the contents of the storage location specified by the effective address; contents of the storage location are not altered

ARITHMETIC INSTRUCTIONS

All the arithmetic operations in table G-9 use one's complement arithmetic.

Table G-9. Arithmetic Instructions

Operation	Description
MUI (F = 2)	Multiply Integer; multiply the contents of the storage location, specified by the effective address, by the contents of the A register; 32-bit product replaces the contents of Q and A, the most-significant bits of the product in the Q register
DVI (F = 3)	Divide integer; divide the combined contents of the Q and A registers by the contents of the effective address; Q register contains the most-significant bits before dividing. If a 16-bit dividend is loaded into A, the sign bit of A must be extended throughout Q. The quotient is in the A register, and the remainder is in the Q register at the end of the divide operation.
	The OVERFLOW indicator is set if the magnitude of the quotient is greater than the capacity of the A register. Once set, the OVERFLOW indicator remains set until a skip on overflow (SOV) or skip on no overflow (SNO) instruction is executed.
ADD (F = 8)	Add to A; add the contents of the storage location, specified by the effective address, to the contents of the A register
	The OVERFLOW indicator is set if the magnitude of the sum is greater than the capacity of the A register. Once set, the OVERFLOW indicator remains set until a skip on overflow (SOV) or skip on no overflow (SNO) instruction is executed.
SUB (F = 9)	Subtract from A; subtract the contents of the storage location, specified by the effective address, from the contents of the A register. Operation on overflow is the same as for an add-to-A instruction.
RAO (F = D)	Replace add one in storage; add one to the contents of the storage location specified by the effective address; contents of A are not altered. Operation on overflow is the same as for an add-to-A instruction
ADQ (F = F)	Add to Q; add the contents of the storage location, specified by the effective address, to the contents of the Q register. Operation on overflow is the same as for an add-to-A instruction.

LOGICAL INSTRUCTIONS

The AND (AND with A) instruction achieves its results by forming a logical product. A logical product is a bit-by-bit multiplication of two binary numbers according to the following rules:

0 x 0 = 0   1 x 0 = 0
0 x 1 = 0   1 x 1 = 1

An example of the AND process is:

```
  0011 (Operand A)
x 0101 (Operand B)
  0001 (Logical product)
```

A logical product is used, in many cases, to select only specific portions of an operand for use in some operation; for example, if only a specific portion of an operand in storage is to be entered into the A register, the operand is subjected to a mask in A. This mask is composed of a predetermined pattern of 0's and 1's. Executing the AND instruction causes the operand to retain its original contents only in those bits which have 1's in the mask in A.

The EOR (exclusive OR with A) instruction achieves its result by forming an exclusing OR. Executing the EOR instruction causes the operand to complement its original contents only in those bits which have 1's in the mask in A. An exclusive OR is a bit-by-bit logical subtraction of two binary numbers according to the following rules:

Exclusive OR

A	B	A⊻B
1	1	0
1	0	1
0	1	1
0	0	0

An example of the EOR process is:

```
  0011 (Operand A)
x 0101 (Operand B)
  0110 (Exclusive OR)
```

Refer to table G-10.

Table G-10. Logical Instructions

Operation	Description
AND (F = A)	AND withA; form the logical product, bit-by-bit of the contents of the storage location specified by the effective address and the contents of the A register. The result replaces the contents of A. The contents of storage are not altered.
EOR (F = B)	Exclusive OR with A; form the logical difference (exclusive OR), bit-by-bit, of the contents of the storage location specified by the effective address and the contents of the A register. The result replaces the contents of A. The contents of storage are not altered.

JUMP INSTRUCTIONS

A jump (JMP) instruction causes a current program sequence to terminate and initiates a new sequence at a different location in storage. The program address register, P, provides continuity between program instructions and always contains the storage location of the current instruction in the program.

When a jump instruction occurs, P is cleared and a new address is entered.† In the jump instruction, the effective address specifies the beginning address of the new program sequence. The word at the effective address is read from storage and is interpreted as the first instruction of the new sequence.

A return jump (RTJ) instruction enables the computer to leave the main program, jump to some subprogram, execute the subprogram, and return to the main program via another instruction. The return jump provides the computer with the necessary information to enable returning to the main program. Figure G-11 shows how a return jump instruction can be used.

A return jump instruction is executed at main program address P. The computer jumps to effective address 0025_{16} and stores P ± 1 or P ± 2 (depending on the address mode of RTJ) at this location. Then program address counter P is set to 0026_{16}, and the computer starts executing the subprogram. At the end of the subprogram, the computer executes a jump instruction (JMP) with indirect addressing. This causes the computer to jump to the address specified by the subprogram address 0025_{16} (P ± 1 or P ± 2 of the main program). Now main program execution continues at P + 1 or P + 2. Refer to table G-11.

†Jumps or return jumps from unprotected to protected storage cause a fault, but the address that is saved in the trap location is the destination address; i.e., the address of the next sequential main program instruction.

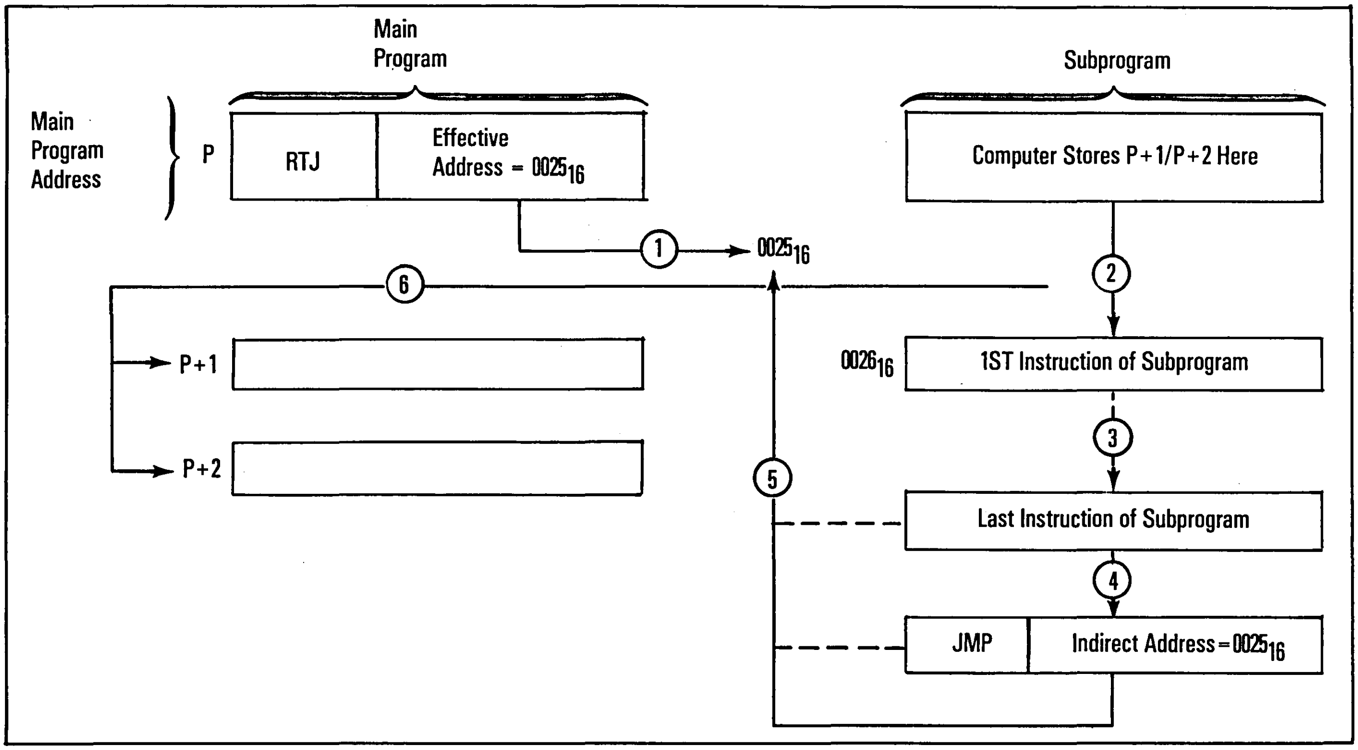

Main
Program
Subprogram
Main
Program
Address
P
RTJ
Effective
Address = 0025_{16}
Computer Stores P+1/P+2 Here
1
0025_{16}
6
2
P+1
0026_{16}
1ST Instruction of Subprogram
3
P+2
5
Last Instruction of Subprogram
4
JMP
Indirect Address=0025_{16}

Figure G-11. Program Using Return Jump Instruction

Table G-11. Jump Instructions

Operation	Description
JMP (F = 1)	Jump; jump to the address specified by the effective address; this effectively replaces the contents of program address counter P with the effective address specified in the jump instruction.
RTJ (F = 5)	Return jump; replace the contents of the storage location specified by the effective address with the address of the next consecutive instruction. The address stored in the effective address is P + 1 or P + 2, depending on the addressing mode of RTJ. The contents of P are then replaced with the effective address plus one.

Register Reference Instructions

Register reference instructions use the address mode field for the operation code. Register reference instructions are identified when the upper 4 bits (15 through 12) of an instruction are 0's.

The format is given in figure G-12.

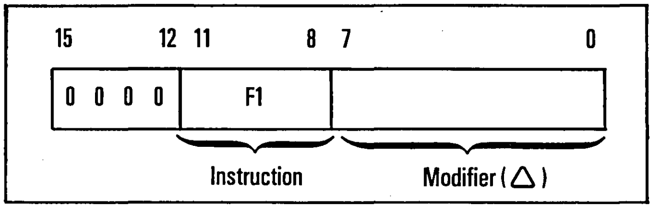


Figure G-12. Register Reference Instruction Format

Refer to table G-12 for instructions.

Table G-12. Register Reference instructions

Operation	Description
SLS (F1 = 0)	Selective stop; stops the computer if this instruction is executed when the SELECTIVE STOP switch is on. On restart, the computer executes the instruction at P + 1. This becomes a pass instruction when the SELECTIVE STOP switch is off.
INP (F1 = 2)	Input to A; reads one word from an external device into the A register. The word in the Q register selects the sending device. If the device sends a reply, the next instruction comes from P + 1. If the device sends a reject, the next instruction comes from P + 1 + △ , where delta is an 8-digit signed

Table G-12. Register Reference Instructions (cont)

Operation	Description
	number. If an internal reject occurs, the next instruction comes from P + △ .
OUT (F1 = 3)	Output from A; outputs one word from the A register to an external device. The word in the Q register selects the receiving device. If the device sends a reply, the next instruction comes from P + 1. If the device sends a reject, the next instruction comes from P + 1 + △ , where delta is an 8-bit signed number. If an internal reject occurs, the next instruction comes from P + △ .
INA (F1 = 9)	Increase A; replaces the contents of A with the sum of the initial contents of A and delta, where delta is treated as a signed number with the sign extended into the upper 8 bits. Operation on overflow is the same as for an add-to-A instruction.
ENA (F1 = A)	Enter A; replaces the contents of the A register with the 8-bit delta, sign extended.
NOP (F1 = B)	No operation; this is a pass instruction (no operation is performed); compares to selective stop instruction with the STOP switch off.
ENQ (F1 = C)	Enter Q; replaces the contents of the Q register with the 8-bit delta, sign extended.
INQ (F1 = D)	Increase Q; replaces the contents of Q with the sum of the initial contents of Q and delta, where delta is treated as a signed number with the sign extended into the upper 8 bits. Operation on overflow is the same as for an add-to-A instruction.

The following instructions (F1 equals 4, 5, 6, 7, or E) are legal only if the program protect switch is off or if the instructions themselves are protected. If an instruction is illegal, it becomes a selective stop and an interrupt on program protect fault is possible (if selected). Refer to table G-13.

- PROTECT switch on

  Selective stop unless instruction is protected

- PROTECT switch off

  Normal instruction execution (no program protection)

Table G-13. Restricted Register Reference Instructions

Operation	Description
EIN (F1 = 4)	Enable interrupt; activates the interrupt system after one instruction following EIN has been executed. The interrupt system must be active, and the appropriate mask bit must be set for an interrupt to be recognized.
IIN (F1 = 5)	Inhibit interrupt; deactivates the interrupt system. If the interrupt occurs during execution of this instruction, the interrupt is not recognized until one instruction after the next EIN instruction is executed.
SPB (F1 = 6)	Set program protect bit; sets the program protect bit in the address specified by Q.
CPB (F1 = 7)	Clear program protect bit; clears the program protect bit in the address specified by Q.
EXI (F1 = E)	Exit interrupt state; this instruction must be used to exit from any interrupt state. Delta defines the interrupt state from which the exit is taken. At the time an interrupt occurs, the value of P is stored in the interrupt trap location assigned to that particular interrupt state. This value is called the return address as it enables return to the next unexecuted instruction after interrupt processing. The EXI instruction automatically reads the address containing the return address, then jumps to the return address. In addition, if the computer is in 32K mode, this instruction also sets the OVERFLOW indicator to the state of bit 15 in the return address. This bit records in state of the OVERFLOW in the return address. This bit records the state of the OVERFLOW indicator when the interrupt occurred. In 65K models, this instruction does not reset the OVERFLOW indicator.

Interregister Instructions

These instructions cause data from certain combinations of two origin registers to be sent through the adder to any combination of destination registers. Various operations, selected by the adder control lines, are performed on the data as it passes through the adder.

The format is shown in figure G-13.

If bit 0 of an interregister instruction is set (M is the destination register) and the instruction is not protected, it is a program protect violation and becomes a nonprotected selective stop instruction. The program protect fault bit is set, and interrupt occurs.

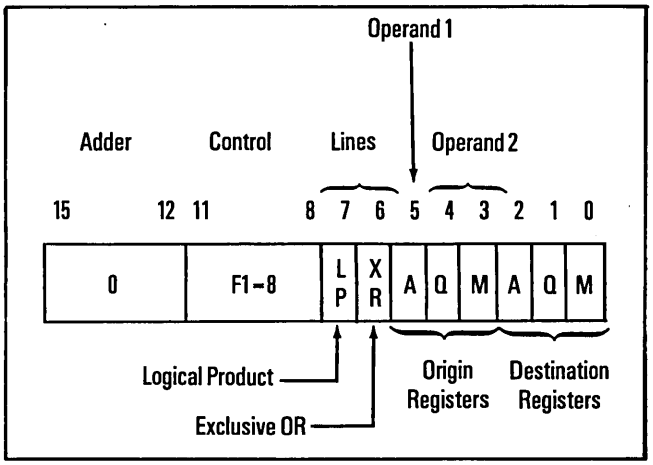


Figure G-13. Interregister Instruction Format

The origin registers are considered as operands. There are two kinds:

- Operand 1 may be FFFF (bit 5 is 0) or the contents of A (bit 5 is 1).

- Operand 2 may be FFFF (bit 4 is 0 and bit 3 is 0); the contents of M (bit 4 is 0 and bit 3 is 1); the contents of Q (bit 4 is 1 and bit 3 is 0); or the inclusive OR, bit-by-bit, of the contents of Q and M (bit 4 is 1 and bit 3 is 1).

The following operations are possible:

- Exclusive OR (LP = 0 and XR = 1)

  The data placed in the destination register(s) is the exclusive OR, bit-by-bit, of operand 1 and operand 2.

- Logical product (LP = 1 and XR = 0)

  The data placed in the destination register(s) is the logical product, bit-by-bit, of operand 1 and operand 2.

- Complement logical product (LP = 1 and XR = 1)

  The data placed in the destination register(s) is the complement of the logical product, bit-by-bit, of operand 1 and operand 2.

- Arithmetic sum (LP = 0 and XR = 0)

  The data placed in the destination register(s) is the arithmetic sum of operand 1 and operand 2. The OVERFLOW indicator operates the same for an add-to-A instruction.

INTERREGISTER MNEMONICS

Refer to table G-14.

Table G-14. Interregister Mnemonics

Operation	Description
SET (F1 = 8, bits 7 through 3 = 10000)	Set to ones
CLR (F1 = 8, bits 7 through 3 = 01000)	Clear to zero
TRA (F1 = 8, bits 7 through 3 = 10100)	Transfer A†
TRM (F1 = 8, bits 7 through 3 = 10001)	Transfer M†
TRQ (F1 = 8, bits 7 through 3 = 10010)	Transfer Q†
TRB (F1 = 8, bits 7 through 3 = 10011)	Transfer Q + M‡
TCA (F1 = 8, bits 7 through 3 = 01100)	Transfer complement A†
TCM (F1 = 8, bits 7 through 3 = 01001)	Transfer complement M†
TCQ (F1 = 8, bits 7 through 3 = 01010)	Transfer complement Q†
TCB (F1 = 8, bits 7 through 3 = 01011)	Transfer complement Q + M†
AAM (F1 = 8, bits 7 through 3 = 00101)	Transfer arithmetic sum A, M
AAQ (F1 = 8, bits 7 through 3 = 00110)	Transfer arithmetic sum A, Q

Table G-14. Interregister Mnemonics (cont)

Operation	Description
AAB (F1 = 8, bits 7 through 3 = 00111)	Transfer arithmetic sum A, Q + M
EAM (F1 = 8, bits 7 through 3 = 01101)	Transfer exclusive OR A, M
EAQ (F1 = 8, bits 7 through 3 = 01110)	Transfer exclusive OR A, Q
EAB (F1 = 8, bits 7 through 3 = 01111)	Transfer exclusive OR A, Q + M
LAM (F1 = 8, bits 7 through 3 = 10101)	Transfer logical product A, M
LAQ (F1 = 8, bits 7 through 3 = 10110)	Transfer logical product A, Q
LAB (F1 = 8, bits 7 through 3 = 10111)	Transfer logical product A, Q + M
CAM (F1 = 8, bits 7 through 3 = 11101)	Transfer complement logical product A, M
CAQ (F1 = 8, bits 7 through 3 = 11110)	Transfer complement logical product A, Q
CAB (F1 = 8, bits 7 through 3 = 11111)	Transfer complement logical product A, Q + M

†The use of bit 7 is optional; it may be a 1 or a 0. The assembler uses bit 7 = 0.

‡Note that the + implies an inclusive OR.

Shift Instructions

The shift instructions shift A, Q, or QA left or right the number of places specified by the 5-bit shift count. Right shifts are end-off with sign extension in the upper bits. Left shifts are end-around. The maximum long-right or long-left shift is 1F places.

The format for shift instructions is given in figure G-14.

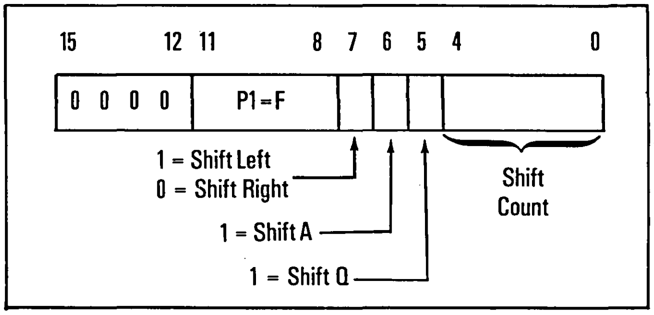


Figure G-14. Shift Instruction Format

An example of the shift instruction is shift A right two places — 0F42. The example is as in figure G-15.

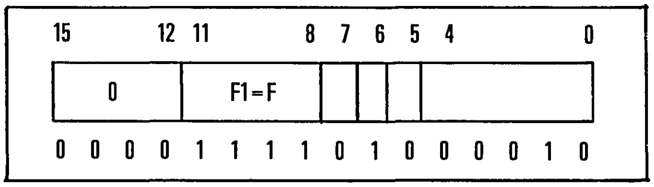


Figure G-15. Sample Shift Instruction

SHIFT MNEMONICS

The shift mnemonics are listed in table G-15.

Table G-15. Shift Mnemonics

Operation	Description
ARS (F1 = F, bits 7 through 5 = 010)	A right shift
QRS (F1 = F, bits 7 through 5 = 001)	Q right shift
LRS (F1 = F, bits 7 through 5 = 011)	Long right shift (QA)
ALS (F1 = F, bits 7 through 5 = 110)	A left shift
QLS (F1 = F, bits 7 through 5 = 101)	Q left shift
LLS (F1 = F, bits 7 through 5 = 111)	Long left shift (QA)

Skip Instructions

Skip instructions result in one machine word: a 12-bit operation code and a 4-bit unsigned skip count. The first 4 bits of the operation code field are set to zero, the next 4 bits contain the skip instruction code 0001, and the last 4 bits contain a unique identifier, F2, for each skip instruction. The expression in the address field of the instruction is evaluated modulo 2^{15}-1.

This expression may be absolute or relocatable. If the expression is absolute, the value of the expression is the skip count. If it is relocatable, the value of the skip count is obtained by subtracting (16-bit one's complement arithmetic) the value of the current location counter plus one from the expression. The skip count is then placed in the last 4 bits of the machine word. The final value of the skip count must not exceed 4 bits, or an error message is printed. If the expression is relocatable, the relocation type of the expression must match the relocation type of the location counter or an error results.

The skip instruction format is shown in figure G-16.

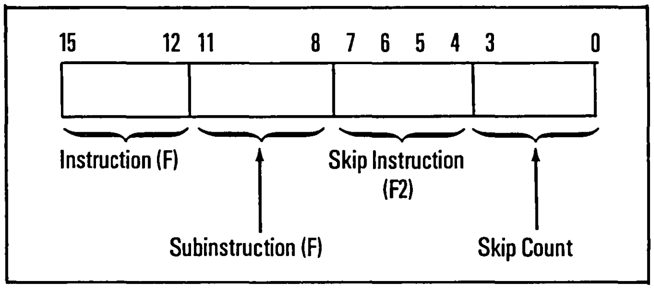


Figure G-16. Skip Instruction Format

When the skip condition is met, the skip count plus one is added to P to obtain the address of the next instruction e.g., when the skip count is zero, go to P + 1). When the skip condition is not met, the address of the next instruction is P + 1 (skip count ignored). The skip count does not have a sign bit. See table G-16.

Table G-16. Skip Mnemonics

Operation	Description
SAZ (F2 = 0)	Skip if A is positive zero (all bits are 0)
SAN (F2 = 1)	Skip if A is not positive zero (not all bits are 0)
SAP (F2 = 2)	Skip if A is positive (bit 15 is 0)
SAM (F2 = 3)	Skip if A is negative (bit 15 is 1)
SQZ (F2 = 4)	Skip if Q is positive zero (all bits are 0)
SQN (F2 = 5)	Skip if Q is not positive zero (not all bits are 0)
SQP (F2 = 6)	Skip if Q is positive (bit 15 is 0)
SQM (F2 = 7)	Skip if Q is negative (bit 15 is 1)
SWS (F2 = 8)	Skip if SELECTIVE SKIP switch is set
SWN (F2 = 9)	Skip if SELECTIVE SKIP switch is not set
SOV (F2 = A)	Skip on overflow; this instruction skips if an overflow condition is sensed. This instruction clears the OVERFLOW indicator.
SNO (F2 = B)	Skip on no overflow; this instruction skips if an overflow condition is not present. This instruction clears the OVERFLOW indicator.
SPE (F2 = C)	Skip on storage parity error; this instruction skips if a storage parity error occurred; it clears the storage parity error interrupt signal and the PARITY FAULT indicator.

Table G-16. Skip Mnemonics (cont)

Operation	Description
SNP (F2 - D)	Skip on no storage parity error
SPF (F2 - E)	Skip on program protect fault; the program protect fault is set by: • A nonprotected instruction attempting to write into an address that is protected. • An attempt to execute a protected instruction immediately following a nonprotected instruction, unless an interrupt caused the instruction sequence. • Execution of any nonprotected instruction affecting interrupt mask or enables. The program protect fault is cleared when it is sensed by the SPF instruction. The program protect fault cannot be set it the program protect system is disabled.
SNF (Fs = F)	Skip on no program protect fault

Negative Zero/Overflow Set

Negative zero and/or overflow set can be caused by two characteristics of the computer:

- The computer has a one's complement subtractive adder.
- Multiplication and division are done with positive numbers only; therefore, a sign correction occurs, if required, before and after the multiplication or division symbols.

Arithmetic operations that produce a negative zero result and/or set overflow in the computer are given in table G-17.

Table G-17. Operations Producing Negative Zero or Overflow

Function	Operation
Addition	(-0) + (-0) = (-0)
Subtraction	(-0) - (+0) = (-0)
Multiplication	(+0) x (-N) = (-0) (-N) x (+0) = (-0) (-0) x (+N) = (-0) (+N) x (-0) = (-0)
Division	$\frac{(+0)}{(-N)}$ = (-0, R = (+0)

Table G-17. Operations Producing Negative Zero or Overflow (cont)

Function	Operation

Where:
N = 0
R = Remainder
	$\frac{(-0)}{(+N)}$ = (-0), R = (-0)
	$\frac{(-0)}{(-N)}$ = (+0), R = (-0)
	$\frac{(+N)}{(+0)}$ = (-0), R = (+N) overflow set
	$\frac{(-N)}{(-0)}$ = (-0), R = (-N) overflow set
	$\frac{(-2N)}{(+N)}$ = (-2), R = (-0)
	$\frac{(-2N)}{(-N)}$ = (+2), R = (-0)
	$\frac{(+0)}{(+0)}$ = (-0), R = (+0) overflow set
	$\frac{(+0)}{(-0)}$ = (+0), R = (+0) overflow set
	$\frac{(-0)}{(+0)}$ = (+0), R = (-0) overflow set
	$\frac{(-0)}{(-0)}$ = (-0), R = (-0) overflow set

PSEUDO INSTRUCTIONS

Pseudo instructions control the assembler, provide subprogram linkage, control output listing, reserve storage, convert data, and so on.

Pseudo instructions may be placed anywhere in a source language subprogram. However, OPT or NAM must be the first statement of a subprogram and MON or END must be the last statement.

Subprogram Linkage

These instructions identify and link subprograms. A symbolic name in the location field is ignored.

NAM STATEMENT

NAM identifies a source language subprogram and must be the first statement of the subprogram. Only the assembler control pseudo instruction OPT (page G-29) may precede it.

The format is:

NAM s

Where:

s—optional symbolic name of the subprogram which is printed as part of the assembly list output

END STATEMENT

END must be the last statement of a source language subprogram. If END terminates a subprogram assembled separately or the last subprogram of a group, the MON instruction follows END. Otherwise END is followed by NAM or OPT.

The format is:

END s

Where:

s—optional symbolic name of an entry point to the first subprogram to be executed

If specified, s must be defined as an entry point in the subprogram to which control passes. This entry point may be in the same subprogram as the END statement or in a subprogram loaded at the same time.

An example of the END statement is as follows:

```
END    START
```

START is the location of the first statement to be executed.

ENT STATEMENT

The ENT instruction lists the symbolic names of entry points which may be referenced from other programs.

The format is:

ENT $s_1,s_2,\ldots,s_n$

Where:

s_i—entry points listed in the address field of ENT which must be defined in the subprogram containing the ENT instruction. s_i must not refer to a location outside the subprogram, common storage, or data storage.

An example of the use of the ENT statement is shown in figure G-17.

```
              NAM   PROG1
              ENT   ENT1,ENT2          (Legal)
       ENT1   LDA   XYZ1
       ENT2   STA   XYZ2
               .     .
               .     .
               .     .
              ENT   ENTX               (Illegal; ENTX not defined)
               .     .
               .     .
               .     .
              END   ENT1
```

Figure G-17. ENT Statement

EXT STATEMENT

The EXT instruction lists the symbolic names of entry points in external subprograms which may be referenced from this subprogram.

The format is:

EXT $s_1,s_2,\ldots,s_n$

Where:

s_i—entry points in the address field of EXT, which must be symbols defined in the subprograms they reference.

s_i must not refer to symbols in the same subprogram.

Examples of the use of the EXT statement are shown in figures G-18 and G-19.

```
          NAM
          EXT    ENT1,ENT2         (Legal)
ENT3      LDA    XYZ
          COM    ENT5
          EXT    ENT3              (Illegal; ENT3 is same subprogram)
          EXT    ENT4              (Legal)
          EXT    ENT5              (Illegal; ENT5 in common storage)
          EXT    ENT1              (Legal; defined in same way as above)
           .      .
           .      .
           .      .
          END
```

Figure G-18. EXT Statement, Example 1

```
          EXT    ENT1,ENT2
           .
           .
           .
          LDA    ENT1
```

Figure G-19. EXT Statement, Example 2

This reference to ENT1 results in the following two machine words:

```
15         11 10  9  8  7               0
| LDA       | 0 | 1 | 0 | 0 |     00       |

15                                      0
|            external link               |
```

External link is a pointer to the location of ENT1 used by the loader at load time.

EXT* STATEMENT

The EXT* instruction is the same as EXT except that s_i represent absolute locations in EXT and references to s_i are made relative in EXT*.

The format is:

EXT* $s_1,s_2,\ldots,s_n$

The plus terminator cannot be used with an operation code when the address references a relative external entry point. It is also illegal to enclose an external in parentheses in the address field of an ADC instruction.

An example of the EXT* statement is shown in figure G-20.

The reference to NAME1 in figure G-20 results in the following two machine words.

```
15         11 10  9  8  7               0
| LDA       | 1 | 0 | 0 | 0 |     00       |

15                                      0
|            external link               |
```

External link is a pointer to the location of NAME1 used by the loader at load time.

```
 .
 .
 .
EXT*   NAME1,NAME2,NAME3
LDA    NAME1
LDA+   NAME1                    (Illegal)
 .
 .
LDA    (NAME2)
ADC    (NAME3)                  (Illegal)

EXT*   NAME1,NAME2
 .
 .
LDA    NAME1
```

Figure G-20. EXT* Statement

Data Storage

The following instructions allocate data storage. BSS and BZS assign storage local to the subprogram in which they appear. COM and DAT assign data common to any number of subprograms. Symbolic names in the location fields of data storage instructions are ignored.

BSS STATEMENT

The BSS instruction assigns symbolic names to segments of storage in the instruction sequence of the subprogram.

The format is:

BSS $s_1(e_1),s_2(e_2),...,s_n(e_n)$

Where:

s_i—name

This symbolic name defines the first location of the named segment.

omitted

When omitted from a subfield, a segment is assigned with the length e, but no name is assigned to the segment.

e_i—expression

These corresponding expressions of the symbolic name define the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo 2^{15}-1 and must be absolute.

0

The associated symbolic name is assigned to the next segment which in effect assigns two names to that segment.

omitted

The length is assumed to be one computer word.

symbolic name

This must be previously defined. It can be assigned by an EQU instruction.

BZS STATEMENT

This statement functions in the same way as the BSS, except that the specified storage locations are set to zero.

The format is:

BZS $s_1(e_1),s_2(e_2),...,s_n(e_n)$

An example of using the BZS statement is shown in figure G-21.

COM STATEMENT

The COM instruction names and defines segments in a block of storage common to more than one subprogram.

The format is:

COM $s_1(e_1),s_2(e_2),...,s_n(e_n)$

Where:

s_i—name

This is the symbolic name which defines the first location of the named segment.

omitted

When omitted from a subfield, a segment is assigned with the length e, but no name is assigned to the segment.

e_i—expression

Corresponding expressions of the symbolic name which defines the length of the segment in

```
         NAM
NAM3     LDA   XYZ1
         BSS   NAM4(3)                   (Assign three words to NAM4)
         BZS   NAM5(5)                   (Assign five words, set to zero,
                                          to NAM5)
         BSS   NAM1,NAM2(9)              (Assign one word to NAM1; assign
                                          nine words to NAM2)
         BSS   NAM3                      (Illegal; NAM3 already assigned)
         BSS   NAM6,(4)                  (Assign one word to NAM6, assign
                                          four words to unnamed segment)
         BSS   NAM7                      (Assign one word to NAM7)
         EQU   NAM8(4),NAM9(2)

         BZS   NAM10(NAM8-NAM9)          (Assign two words, set to zero, to
                                          NAM10)
         BSS   NAM8(NAM10-1)             (Illegal; NAM8 already assigned)
         BSS   LOC1(0),LOC2              (Assign the same word to LOC1 and
          .                               LOC2)
          .
          .
         END
```

Figure G-21. BZS Statement

words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo 2^{15}-1, and must be absolute.

0

The associated symbolic name is assigned to the next segment which, in effect, assigns two names to that segment.

omitted

The length is assumed to be one computer word.

symbolic name

This must be previously defined; can be assigned by an EQU instruction.

If a program includes more than one COM statement, they define consecutive segments of common storage in the order of their appearance. The area used by common storage is assigned by the loader at load time to locations outside the program area. Data in common storage cannot be preset by the ORG pseudo instruction.

An example of the COM statement is figure G-22.

DAT STATEMENT

The DAT instruction reserves area for common storage which is assigned within the program area and may be

```
         NAM
         COM   NAM4
NAM3     STA   XYZ1
         COM   NAM7($1EF),NAM8
         EQU   NAM1(6),NAM2(2)
         COM   NAM5(NAM1-MAN2)
         COM   NAM6(NAM3)                (Illegal)
          .
          .
          .
         END
```

Figure G-22. COM Statement

preset with data or instructions by using the ORG pseudo instruction.

The format is:

DAT $s_1(e_1),s_2(e_2),...,s_n(e_n)$

Where:

s_i—name

This is the symbolic name which defines the first location of the named segment.

omitted

When omitted from a subfield, a segment is assigned with the length e, but no name is assigned to the segment.

e_i—expression

The corresponding expressions of the symbolic name define the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo 2^{15}-1 and must be absolute.

0

The associated symbolic name is assigned to the next segment which in effect assigns two names to that segment.

omitted

The length is assumed to be one computer word.

symbolic name

The symbolic name must be previously defined; can be assigned by an EQU instruction.

Constant Declarations

These pseudo instructions introduce constant values into the instruction sequence.

ADC/ADC* STATEMENTS

The ADC/ADC* instruction evaluates numerical constants or address expressions and inserts the results in line. When ADC is followed by an asterisk, the evaluated address expressions are made relative to the current location counter. The relocation type of the expression must be the same as that of the location counter. The value of the locations counter is subtracted from the value of the evaluated expression (16-bit one's complement arithmetic) and the result is the 16-bit address constant.

The format is:

s ADC $e_1,e_2,(e_3),...,e_n$

Where:

s—this is the symbolic name in the location field which is assigned to the first constant in the address field.

e_i—This represents the numerical constant or address expression to be evaluated. The result is evaluated modulo 2^{15}-1. Bit 15 is set if the expression is enclosed in parentheses (indicating an indirect reference). The results corresponding to $e_1,e_2,...,e_n$ are stored in consecutive storage locations.

NOTE

Indirect addressing cannot be specified in the ADC* statement.

ALF STATEMENT

The ALF instruction translates a message into ASCII format.

The format is:

s ALF n,message

Where:

s—This is the symbolic name in the location field which is assigned to the first constant in the address field.

n—This represents the unsigned integer, specifying the number of words to be stored; 2n equals the number of characters.

If n is an integer, 2n characters of the message are stored. Excess characters are treated as a remark. (The ALF statement, including the message, will not be processed beyond the 72nd character of the source image.) If the message is less than 2n characters, the unused portion of the specified area is blank filled.

It can be a noninteger character which signals the end of the message. When n is a special terminating character, the storage of the message terminates the first time this character is encountered in the message if it occurs before the 72nd character. If the character just prior to n is the first character of a word, a blank is placed in the second character to complete the word.

A character message is stored into consecutive locations in the instruction sequence. The message is converted to ASCII characters and is stored two 8-bit characters per word.

The typewriter control characters in table G-18 may be input with the ALF statement.

Table G-18. Typewriter Control Characters

Code	Meaning	Hexadecimal Value
:R	Carriage return	D
:T	Horizontal tab	9
:L	Line feed	A
:B	Bell	7
:F	Top of form	C
:V	Vertical tab	B

These codes are converted to a single output character with the corresponding hexadecimal value and are counted as one character in determining the value of n, when n is an integer character count. A colon is an 8-to-5 keypunch code with the ASCII value of $3A_{16}$.

A symbolic name in the location field is assigned to the first word of the message.

The following source language statements, for example, are translated into machine words as shown in table G-19.

```
       ALF    4,EXAMPLE1
NAM1   ALF    .,EXAMPLE2
       ALF    6,EXMP3:TEXMP4:R
NAM2   ALF    4,EXMP5
```

Table G-19. ALF Statements Translated to Machine Words

Location	Character	
	Left	Right
	E	X
	A	M
	P	L
	E	1
NAM1	E	X
	A	M
	P	L
	E	2
	△	△
	.	.
	.	.
	.	.
	△	△
	E	X
	M	P
	3	Tab
	E	X
	M	P
	4	Carriage return
NAM2	E	X
	M	P
	5	

In this example, △ is a blank. Three dots indicate blanks fill in the words between EXAMPLE2 and EXMP3. This is because the special terminating character, ., does not occur in the message before the 72nd character. If, in the example, n is in column 13, then 25 words of blanks are used to fill the words between EXAMPLE2 and EXMP3.

NUM STATEMENT

The NUM instruction defines numeric constants.

The format is:

s NUM $k_1,k_2,...k_n$

Where:

s—This represents the symbolic name in the location which is assigned to the first constant in the address field.

k_i—These are the specified integer constants stored into consecutive locations in the instruction sequence. Each constant may be a decimal integer in the range ±32767 or a hexadecimal integer preceded by a $ in the range ±7FFF. The constant may be signed. If it is not signed, the constant is assumed to be positive. When the sign is minus, the one's complement of the number is used.

The following source language statements are translated into machine words as shown in table G-20.

```
        NUM    1,2,3,$A
NAM1    NUM    +14,-10,-$13B,$7FF
```

Table G-20. NUM Statements Translated to Machine Words

Location	Contents	Location	Contents
	0001 0002 0003 000A	NAM1	000E FFF5 FEC4 07FF

DEC STATEMENT

The DEC instruction converts decimal constants into fixed-point binary.

The format is:

s DEC $k_1,k_2,...,k_n$

Where:

s—The symbolic name in the location is assigned to the first constant in the address field.

k_i—The specified integer constants are stored into consecutive locations in the instruction sequence. It is a signed decimal integer followed by a decimal and/or binary scaling factor. The decimal scaling factor consists of the letter D followed by a signed or unsigned decimal integer. The binary scaling factor is the letter B followed by one or two signed or unsigned decimal digits. The form of a constant in the address field may be:

fDdBb

which is equivalent to the algebraic expression:

$f \cdot 10^d \cdot 2^b$

The fixed-point binary number resulting from the conversion must have a magnitude less than 2^{15}. If the result of scaling is greater than $2^{15}-1$, an error diagnostic is printed.

A symbolic name in the location field is assigned to the location of the first constant. The source language statements are converted to machine words as indicated in table G-21.

```
        DEC    35D-1B6
NAM1    DEC    -35B6
        DEC    32760B-4
NAM2    DEC    32761D-5B15,+625D-2B3
NAM3    DEC    10D3
```

Table G-21. DEC Statement Translated to Machine Words

Location	Contents of Bits 15 Through 0
NAM1	0000000011100000 1111011100111111
NAM2	0000011111111111 0010100111101111
NAM3	0000000000110010 0010011100010000

VFD STATEMENT

The VFD (variable field definition) instruction assigns data to consecutive locations in the instruction sequence without regard for computer words. Data is stored in bit strings rather than word units. Data may be numeric constants, ASCII characters, or expressions. A symbolic name in the location field is assigned to the first word of data.

The format is:

s VFD $m_1n_1/v_1,m_2n_2/v_2,...,m_nn_n/v_n$

Where:

s—name

This is the symbolic name which defines the first location of the named segment.

m_i—mode of the data

N

When the value of the data is a numeric constant, the mode is specified as N, and the number of bits must not be greater than 16. If n is larger than necessary, the value is right-justified in the field and the sign is extended in the remaining high-order bits. If n is less than is required, the value is truncated and the least significant bits are stored. The value, v, is a decimal integer or a hexadecimal integer preceded by a dollar sign. Integers may be signed or unsigned; if the sign is omitted, the number is assumed to be positive. A decimal number must be in the range ±32767 and a hexadecimal integer in the range ±7FFF.

A

When v is a string of characters, m must be A, and n must be a multiple of 8. The number of characters in the string should be equal to n/8 including embedded blanks. The last character must be followed by a blank or a comma. The characters are converted to ASCII code and stored as in the ALF instruction.

X

When v is an expression, m must be X, and n must be less than or equal to 16. If n is less than 16, the final value of the expression may be relocatable or absolute. It is evaluated modulo 2^{15}-1=$7FFF_{16}$. If the final value is absolute and n exceeds the size required, the value is right-justified in the field. If absolute and n is less than the required size, the value is truncated and the least-significant bits are stored in the field. If the final value is relocatable, n must equal 15, and the expression must be positioned so that it will be stored right-justified at bit position 0 of the computer word.

If n equals 16, the expression must be absolute; it is evaluated, using 16-bit one's complement arithmetic. If a symbol is used in a 16-bit expression, bit 14 of the value of the symbol is extended to bit 15, and therefore, the calculation of the value of the symbol is accurate only to 2^{14}-1. If the symbol A is equated to the value -1, for example, the value of A in the symbol table is $7FFE_{16}$, but the value used in the 16-bit calculation of this symbol is $FFFE_{16}$. Numeric operands used in a 16-bit expression may be 16 bits in magnitude.

n_i—number of bits to be allocated

v_i—value of the data

The following source language statements:

```
NAM
VFD    N3/1,X5/6-4,A16/XY,X4/NAM1-NAM2
BSS    NAM2(3),NAM1
 .
 .
 .
END
```

Result in the following machine words:

Word 1

15	12	7 0
0 0 1	0 0 0 1 0	0 1 0 1 1 0 0 0

Word 2

15	7	3 0
0 1 0 1 1 0 0 1	0 0 1 1	0 0 0 0

The following source language statements:

```
NAM
VFD    N8/-1,A8/L,N1/0,X15/NAM1
BSS    NAM1
 .
 .
 .
END
```

Result in machine words as follows:

Word 1

15	7 0
1 1 1 1 1 1 1 0	0 1 0 0 1 1 0 0

Word 2

15	14 0
0	loc of NAM1

The following source language statements:

```
NAM
EQU    A(-1),B(2)
VFD    X16/A,X16/B,X16/$7FFF*2
  .
  .
END
```

Result in the following machine words:

Word 1

```
15                              0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
```

Word 2

```
15                              0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Word 3

```
15                              0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
```

Assembler Control

The assembly process is controlled or modified by the pseudo instructions defined in the following paragraphs. A symbolic name in the location field is ignored except where specifically noted.

EQU STATEMENT

The EQU instruction equates each symbolic name to the expression value.

The format is:

EQU $s_1(e_1),s_2(e_2),\ldots,s_n(e_n)$

Where:

s_i—name

Symbolic name s_i is equated to the value e_i.

e_i—expression

Any symbolic operand used in the expression must be previously defined and not external to the subprogram in which the EQU statement appears. e_i are evaluated modulo 2^{15}-1 and must be absolute.

omitted

The expression is assumed to be zero.

An example of the EQU statement is given in figure G-23.

ORG/ORG* STATEMENT

The ORG statement specifies an address expression to which the current location counter is set.

The format is:

ORG e

Where:

e—expression

The expression, e, is evaluated modulo 2^{15}-1, and the location counter is set to the resultant value. The value of the expression may be program or data relocatable. Or it can be abso-

```
        NAM   EXAMPL
PICKUP  LDA   XYZ1
NAM6    ADD   XYZ2
        EQU   NAM3($4F),NAM4(-39)
        EQU   NAM7(NAM6-1)
        EQU   NAM8(STORE)              (Illegal)
STORE   STA   XYZ3
        EQU   NAM9(STORE)              (Legal)
         .
         .
         .
        END
```

Figure G-23. EQU Statement

lute. If it is relocatable, it must be positive. Any symbolic operands in the expression must have been previously defined.

The instructions following an ORG statement are assembled into consecutive locations beginning at the location of the evaluated address expression, e. This sequence may be changed by another ORG or terminated by an ORG* statement. In the range of a data relocatable ORG, any reference to an external symbol is illegal.

The format is:

ORG*

This instruction is used to return to the normal instruction sequence previously interrupted by an ORG. More than one ORG may be specified without an intervening ORG*; however, when an ORG* does occur, the location counter is reset to the value it had prior to the first ORG.

An example of ORG and ORG* statements is given in figure G-24.

```
           .
           .
      BSS     ORG1(10),ORG2,ORG3(5)
NAM1  ENA     0
           .
           .
           .
NAM2  JMP*    NAM3
      ORG     NAM1
      (sequence of code beginning at NAM1)
      ORG*
      (resume sequence of code at NAM2+1)
NAM3  JMP*    NAM4
      ORG     ORG1
      (sequence of code beginning at ORG1)
      ORG     ORG3
      (sequence of code beginning at ORG3)
      ORG*
      (resume sequence of code at NAM3+1)
           .
           .
      END
```

Figure G-24. ORG/ORG* Statements

IFA STATEMENT

The IFA instruction assembles a set of coding lines only if a specified conditions is true.

The format is:

s IFA e_1,c,e_2

Where:

s—The symbolic name in the location field is used as an identifying tag only; it is not defined as a location symbol in the program. If it is specified, the first two characters of the identifier, s, must match the first two characters of the symbolic name in the address field of the corresponding EIF. If s is blank in an IFA statement, it must also be blank in the corresponding EIF.

e_i—The expressions e_1 and e_2 are evaluated modulo 2^{15}-1 and must result in an absolute value. Any symbolic name in either expression must have been previously defined.

c—If the conditions specified by c exist between e_1 and e_2, the code is assembled; if the condition does not exist, the code following the IFA statement is skipped until a corresponding EIF statement is encountered.

The following conditions may be specified by c.

Condition	Meaning
EQ	$e_1 = e_2$
NE	$e_1 = e_2$

GT	e_1 e_2
LT	e_1 e_2

EIF STATEMENT

The EIF instruction signals the termination of an IFA or IFC instruction when coding lines are skipped as a result of an untrue condition. When the condition in the IFA or IFC is true, EIF is ignored.

The format is:

EIF s

Where:

s—The symbolic name, s, in the address field establishes the correspondence between an IFA or IFC and an EIF instruction. The first two characters of s must be the same as the first two characters in the location field of the corresponding IFA or IFC. An EIF with a blank address field terminates an unlabeled IFA or IFC.

An example of the EIF statement is given in figure G-25.

```
        NAM
         .
         .
LOC1    BSS    A(20),B(10),C(2)
        EQU    NAM1(10),NAM4(B),NAM2(2)
NAM3    IFA    NAM1,EQ,NAM2+8
OP1     SAZ    1
        EIF    NAM3
        IFA    NAM1,GT,NAM2+8
OP2     SAZ    2
        EIF
         .
         .
         .
        END
```

Figure G-25. EIF Statement

In figure G-25, OP1 is assembled and OP2 is skipped if the value of NAM1 equals the value of NAM2+8; OP1 is skipped and OP2 is assembled if the value of NAM1 is greater than the value of NAM2+8; both OP1 and OP2 are skipped if the value of NAM1 is less than the value of NAM2+8.

OPT STATEMENT

The OPT pseudo instruction signals the input of control options to the assembler.

The format is:

OPT

When OPT appears, the assembler requests input of control options by typing:

OPTIONS

The control options in table G-22 are entered in any order on the teletypewriter. Imbedded spaces and illegal characters are ignored. A carriage return signals the end of control options input.

Table G-22. OPT Statement Control Options

Option	Meaning
L	List output on standard list device
P	Punch output on standard punch device
X	Execute output on mass storage device
M	List called macro skeletons
A	Abandon all remaining assemblies and return control to operating system
Ilu	Input from unit lu; reads instructions until the END statement is encountered, then returns to the standard input device; lu may be any ASCII or BCD input device
C	List cross references at end of assembly listing

OPT is not a part of the source language program. It is used strictly for control of the assembler and has no code associated with it.

OPT may precede any NAM instruction in any subprogram. If the first statement encountered is not OPT, standard options are assumed until END is encountered. If OPT is encountered between the first statement of a program and the END statement, a diagnostic is issued. The standard options are L, P, X, and C.

MON STATEMENT

The MON instruction returns control to the operating system after the last subprogram has been assembled.

The format is:

MON

MON may be used only after the END statement. The location and address fields are ignored. This statement is part of the source language program and is used strictly for control of the assembler; no code is associated with it.

Listing Control

The following pseudo instructions control the printing of assembly output. The location and address fields are ignored unless they are specified.

NLS STATEMENT

The NLS instruction inhibits list output.

The format is:

NLS

Normally list output is enabled initially until an NLS occurs and then remains inhibited until an LST instruction or the end of the program occurs.

LST STATEMENT

The LST instruction initiates list output after an NLS has inhibited it.

The format is:

LST

SPC STATEMENT

The SPC instruction control line spacing on the list output unit.

The format is:

SPC e

Where:

e—The number of lines to be skipped; the expression is evaluated modulo 2^{15}-1 and must be absolute.

EJT STATEMENT

The EJT instruction causes page ejection during printing of the list output.

The format is:

EJT

MACROS

An often used set of instructions may be grouped together to form a macro. Once a macro is defined, it may be used as a pseudo instruction. The 1700 macro assembler includes two types of macros.

- Programmer defined

  These are macros which must be declared by MAC pseudo instructions immediately following the NAM image. Comment cards may, however, be placed anywhere in the macro definition.

- Library

  These are definitions contained on the system library and may be called from any subprogram.

Macro Pseudo Instructions

These pseudo instructions are used only in a macro definition.

MAC STATEMENT

The MAC instruction is required and names a macro and lists its formal parameters. The location field contains the name used to call the defined macro. It may be any name which is not a machine or pseudo instruction.

The format is:

s MAC $P_1,P_2,...,P_n$

Where:

s—This is a symbolic name in the location field which is assigned to the first word of the generated code.

p_i—These symbolic names are local to the macro definition and may be used anywhere else in the program without ambiguity. The formal parameters must conform to the following rules:

- They must be symbolic names of 1 or 2 characters.

- The parameter list must not extend beyond the 72nd character of the line containing MAC.

- The parameter list must terminate with a blank or the 72nd character of the line.

- Each parameter in the list is separated from the next by a comma.

EMC STATEMENT

The EMC instruction is required and signals the end of a macro definition. A symbolic name in the location or address field is ignored. EMC is always the last instruction in a macro definition.

The format is:

```
    EMC
```

LOC STATEMENT

The LOC instruction is optional and allows the use of the same symbols in macros and programs to avoid doubly defined symbols. Symbols, other than formal parameters, that are local to the macro being defined are listed in this instruction. Local symbols have meaning only in the macro in which they are listed by LOC, thereby allowing the same symbols to be used elsewhere in the program without ambiguity.

The LOC instruction must immediately follow the MAC instruction. A symbol in the location field of the LOC instruction is ignored.

The format is:

LOC $s_1,s_2,...,s_n$

Where:

s_i—These local symbols in the address field must conform to the following rules:

- They must be symbolic names of one or two characters.

- The list cannot extend beyond the 72nd character of the line containing the LOC instruction.

- The list terminates with a blank or the 72nd character of the line.

- Each symbol in the list is separated from the next by a comma.

- No local symbol in the list may be the same as a formal parameter specified for the macro.

- No more than 256 local symbols can be used in one program.

IFC STATEMENT

The IFC instruction is optional and allows a set of instructions in a macro definition to be assembled only if a specified condition is true. This instruction is meaningful only in the range of a MAC pseudo instruction.

The format is:

s IFC a_1,c,a_2

Where:

s—The symbol in the location field is an identifying tag used to establish correspondence with the terminating EIF. An EIF terminates an IFC when the first two characters of the symbol in the address field of EIF are the same as the location symbol of the IFC or when both symbols are blank and it is the first EIF encountered.

a_i—This must be a string of from one to six characters or a formal parameter specified in the MAC statement. The character string should not contain commas, blanks, or apostrophes. Two character strings are equal when they contain the same characters in the same position and are of the same length. Characters in excess of six are ignored.

c—The specified condition meets the following rules:

Condition	Meaning
EQ	$a_1 = a_2$
NE	$a_1 \neq a_2$

If the condition specified exists between a_1 and a_2, the code is assembled; if not, the code following the IFC is skipped until a corresponding EIF pseudo instruction is encountered.

Source language examples of macro definitions and instructions are given on page G-34.

Macro Skeleton

A macro skeleton is the set of instructions in a macro definition that is the prototype of the operations to be performed when the macro is called.

The instructions may be any machine or pseudo instruction except MAC, LOC, EMC, NAM, END, or MON. A macro skeleton may also contain macro instructions calling other macros. A macro skeleton may not contain a macro instruction which calls itself. Formal parameters, enclosed in apostrophes, may appear anywhere in the instruction format of a prototype instruction. Local symbols defined by a LOC statement may be used anywhere in the macro skeleton; they also must be enclosed in apostrophes. The only legal use of the apostrophe in a macro definition is to enclose formal parameters or local symbols. Formal parameters that extend past the 72nd character into the sequence field are ignored. Formal parameters in a remark statement signaled by an * in column 1 are also ignored.

In addition to the formal parameters specified in the MAC pseudo instruction, a special formal parameter (a period enclosed in apostrophes) may be used in the macro skeleton. It is replaced by the instruction terminator of the calling macro instruction when a terminator is specified.

Let A, B, C, ... be distinct arbitrary macro skeletons. A may contain a macro instruction calling B, B a macro calling C, etc. Up to 10 such successive macro calls are allowed by the assembler. Further successive calls are ignored.

A sample macro skeleton is illustrated by figure G-26.

```
XYZ   MAC     P1,P2,P3,P4,P5
      LOC     A

      LDA     'P1'      }
      'P2'    'P3'      }
      S'P4'Z  'A'-*-1   }     Macro skeleton
      JMP'.'  'P5'      }
'A'   ENA     1         }

      EMC
```

Figure G-26. Macro Skeleton

Macro Instruction

With a macro instruction, the code generated from the named macro is inserted in the instruction sequence beginning at the location of the macro instruction.

The format is:

s N $P_1, P_2, \ldots, P_n$

Where:

s—A symbolic name in the location field is assigned to the first word of the generated code.

N—This is the symbolic name of the macro in the operation code field. It is the name specified in the location field of the MAC statement of the macro definition it calls. The macro name may be followed by one of the special terminators +, -, or *.

p_i—These symbolic names are local to the macro definition and may be used anywhere else in the program without ambiguity.

PARAMETERS

Parameters are defined in the following paragraphs.

Actual

The actual parameters must be listed in the same order as the formal parameters in the MAC statement. The list of actual parameters must conform to the following rules.

- Each parameter in the list is separated from the next by a comma.
- The list is terminated with a blank or the 72nd character unless the 72nd character is a comma.
- The list may be continued onto the next line; if so, the last parameter on the list is terminated by a comma and a blank or the 72nd character.
- The continuation line must contain the macro name in the operation code field. A symbolic name in the location field is ignored.

- An actual parameter containing embedded blanks or commas must be enclosed by apostrophes.

The internal buffer for storage of actual parameters is 96 words long; this allows approximately three continuation lines. If the buffer overflows, an error message is given.

An example of actual parameters is where the macro defined in the previous example as XYZ could be called by the following macro instruction:

```
TAG1  XYZ*  SYMB1,STA,'SYMB2,I',
      XYZ*  Q,LABEL1      (Continuation line)
```

This macro instruction would generate the following code starting at location TAG1:

```
TAG1  LDA    SYMB1
      STA    SYMB2,I
      SQZ    [nn-*-1
      JMP*   LABEL1
[nn   ENA    1
```

NOTE

[nn is a unique identifier assigned at assembly time.

Null

Actual parameters may be omitted from a macro instruction. An omitted (null) parameter in the middle of the list is indicated by its terminating comma only. Parameters at the end of the list may be omitted with no indication.

An example of the use of null parameters would be:

```
XYZ  MAC     P1,P2,P3,P4,P5,P6
```

The macro instruction, with P2, P4, and P6 omitted in the actual parameter list, would be:

```
XYZ     MUI,,SYMB5,,3
```

Empty fields are allowed in all machine and pseudo instructions with the following exceptions:

```
ALF  n,message  (n must be specified)

EQU  s(e)   )
COM  s(e)   }  (If e is specified, s
DAT  s(e)   )   must be specified)

IFA  e1,c,e2 }
IFC  a1,c,a2 }  (c must be specified)
```

Actual parameters to be inserted into the value of a VFD instruction using mode A must agree with the number of characters that are specified. A null actual parameter can cause an error in the generated code unless the VFD allows for null parameters, for example:

```
X     MAC     P,Q,R
      VFD     A8/'P',A8/'Q',A8/'R'
```

For the macro defined, the calling macro instruction must specify each actual parameter as one character long. If an actual parameter is more than one character, an error message is given. If an actual parameter is omitted, however, a code is generated, and an error results, for example:

```
X       A,,B                 (Q is omitted)
VFD     A8/A,A8/,A8/B        (Code generated)
```

If actual parameters might be omitted, the VFD instruction in the macro skeleton should include empty subfields for each character, for example, the macro definition should be written:

```
X     MAC     P,Q,R
      VFD     A8/'P',,A8/'Q',,A8/'R',
```

A calling sequence with no actual parameters generates the following code and no error results.

```
VFD     A8/,,A8/,A8/,
```

EXAMPLES OF MACRO INSTRUCTIONS

The examples in figures G-27 to G-32 show macro definitions and the code generated by macro instructions calling the defined macros.

Macro Definition

```
XYZ        MAC        P1,P2,P3,P4,P5,P6
           LDQ        =N'P5','P6'
'P4'       LDA        'P3'
           'P1'       'P2'
           ADD        SYMB1
           IFA        'P5',NE,0
I1         IFC        'P1',EQ,MUI
           STA        SYMB3
```

```
              LDA            SYMB2
              EIF
              EIF            I1
              EMC


Macro Instruction
CALL1         XYZ            MUI,'SYMB4,I',SYMB5,HERE,3,I


Generated Code
CALL1         LDQ            =N3,I
HERE          LDA            SYMB5
              MUI            SYMB4,I
              ADD            SYMB1
              IFA            3,NE,0            (Condition satisfied)
I1            IFC            MUI,EQ,MUI        (Condition satisfied)
              STA            SYMB3             (Assembled)
              LDA            SYMB2             (Assembled)
              EIF
              EIF            I1


Macro Instruction
CALL2         XYZ            DVI,SYMB7,'SYMB8,I',THERE,2


Generated Code
CALL2         LDQ            =N2
THERE         LDA            SYMB8,I
              DVI            SYMB7
              ADD            SYMB1
              IFA            2,NE,0            (Condition satisfied)
I1            IFC            DVI,EQ,MUI        (Condition not satisfied)
              STA            SYMB3             (Not assembled)
              LDA            SYMB2             (Not assembled)
              EIF
              EIF            I1
```

Figure G-27. Macro Instruction, Example 1

```
Macro Definition
A     MAC     P1,P2,P3,P4
I1    IFC     *,EQ,'P1'
      LDA     'P2'
      EIF     I1
I2    IFC     *,NE,'P1'
      LDA     'P3'
      EIF     I2
      STA     'P4'
      EMC
```

```
Macro Instruction
                 A            *,NAM1,NAM2,NAM3

Generated Code
I1               IFC          *,EQ,*                 (Condition satisfied)
                 LDA          NAM1                   (Assembled)
                 EIF          I1
I2               IFC          *,NE,*                 (Condition not satisfied)
                 LDA          NAM2
                 EIF          I2
                 STA          NAM3
```

Figure G-28. Macro Instruction, Example 2

```
Macro Definition
JAN   MCA        SY
      IFC        I,EQ,'.'
      SAZ        1
      EIF
      IFC        I,NE,'.'
      SAZ        2
      EIF
      JMP'.'     'SY'
      EMC

Macro Instruction
                 JAN*         SYMB1

Generated Code
                 IFC          *,EQ,*                 (Condition satisfied)
                 SAZ          1                      (Assembled)
                 EIF                                 (Ignored)
                 IFC          *,NE,*                 (Condition not satisfied)
                 SAZ          2                      (Not assembled)
                 EIF                                 (Skip terminated)
                 JMP*         SYMB1

Macro Instruction
                 NAM          SYMB2

Generated Code
                 IFC          *,EQ,                  (Condition not satisfied)
                 SAZ          1                      (Not assembled)
                 EIF                                 (Skip terminated)
                 IFC          *,NE,                  (Condition satisfied)
                 SAZ          2                      (Assembled)
                 EIF                                 (Ignored)
                 JMP          SYMB2
```

Figure G-29. Macro Instruction, Example 3

Macro Definition

```
IFEXMP MAC    P1
Z      IFC    *,EQ,'P1'
       NUM    2
       EIF    Z
Y      IFC    *,NE,'P1'
X      IFC    0,EQ,'P1'
       NUM    1
       EIF    X
Y      IFC    0,NE,'P1'
       NUM    0
       EIF    Y
       EMC
```

Macro Instruction

```
              IFEXMP     *
```

Generated Code

```
Z      IF     *,EQ,*                                (Condition satisfied)
       NUM    2                                     (Assembled)
       EIF    Z
Y      IFC    *,NE,*                                (Condition not satisfied)
X      IFC    0,EQ,*                                (Not assembled)
       NUM    1                                     (Not assembled)
       EIF    X                                     (Not assembled)
Y      IFC    0,NE,*                                (Not assembled)
       NUM    0                                     (Not assembled)
       EIF    Y                                     (Skip terminated)
```

Macro Instruction

```
              IFEXMP     0
```

Generated Code

```
Z      IFC    *,EQ,0                                (Condition not satisfied)
       NUM    2                                     (Not assembled)
       EIF    Z                                     (Skip terminated
Y      IFC    *,NE,0                                (Condition satisfied)
X      IFC    0,EQ,0                                (Condition satisfied)
       NUM    1                                     (Assembled)
       EIF    X
Y      IFC    0,NE,0                                (Condition not satisfied)
       NUM    0                                     (Not assembled)
       EIF    Y                                     (Skip terminated)
```

Macro Instruction

```
              IFEXMP
```

Generated Code

```
Z      IFC    *,EQ,                                 (Condition not satisfied)
       NUM    2                                     (Not assembled)
       EIF    Z                                     (Skip terminated)
Y      IFC    *,NE,                                 (Condition satisfied)
X      IFC    0,EQ,                                 (Condition not satisfied)
       NUM    1                                     (Not assembled)
       EIF    X                                     (Skip terminated)
Y      IFC    0,NE,                                 (Condition satisfied)
       NUM    0                                     (Assembled)
       EIF    Y
```

Figure G-30. Macro Instruction, Example 4

Macro Definitions

```
DEPTH1   MAC      A
         DEPTH2   'A',PARAM1
         EMC

DEPTH2   MAC      A,B
         DEPTH3   'A',PARAM2
         EMC

DEPTH3   MAC      C,D
         LDA      'C'
         STA      'D'
         EMC
```

Macro Instruction

```
         DEPTH1   SYMB1
```

Generated Code

```
         DEPTH2   SYMB1,PARAM1
         DEPTH3   SYMB1,PARAM2
         LDA      SYMB1
         STA      PARAM2
```

Figure G-31. Macro Instruction, Example 5

Macro Definition

```
B        MAC      A,B,C,D,E,F,G,H,I,J,K
         LOC      LO
         ALF      'A,'B'△ ERROR
         VFD      'C'/'D',A16/'E',,,A32/TEST
         IFC      'G',EQ,SKIP
         LDA      'H'
         EIF
'I'      INA      'J'
         'K'      1
         SAN      'LO'
         ENA      -1
'LO'     STA      'F'
         EMC
```

Macro Instruction

```
         B        4,1,N4,-1,XY,'TEMP,I',SKIP,'TEMP,I',
         B        NAM2,10,NOP
```

Generated Code

```
         ALF      4.1△ ERROR
         VFD      N4/-1,A16/XY,,,A32/TEST
         IFC      SKIP,EQ,SKIP
```

```
            LDA     TEMP,I
            EIF
    NAM2    INA     10
            NOP     1
            SAN      nn
            ENA     -1
     nn     STA     TEMP,I
```

Figure G-32. Macro Instruction, Example 6

Macro Library

LIBMAC is released as a separate library macro preparation routine. Input to this routine is in the form of a set of macro definitions, each starting with a MAC statement and ending with an EMC statement. The complete set of macro definitions to be input to the library is terminated by the characters ENDMAC starting in column 1 of the source image.

The library macro preparation routine outputs two files on the standard I/O device for binary output. One contains a macro directory; the other contains the macro skeletons. The routine checks for errors and prints an error message along with the line containing the error.

Binary output is in two sections; the macro skeleton file and the macro directory file. After the skeleton file has been output, the message MACSKL END is output on the typewriter and a carriage return must be typed to start output of the macro directory.

The output files are placed on the program library in two permanent files using the system initializer or library editor of COS.

The library editor is used to put the macros on the program library.

The following control statement places the macro directory file on the program library:

*N,MACROS,,,B

The following control statement places the macro skeletons on the program library:

*N,MACSKL,,,B

ASSEMBLER OUTPUT

The following paragraphs discuss the output of the assembler.

Control Options

Four standard options (table G-23) determine the type of output from the assembler. All four are automatically selected if no OPT statement is encountered before the first NAM.

Table G-23. Control Options

Option	Meaning
P	Relocatable binary output on standard output unit
X	Load and go; execute output loaded on a mass storage device
L	List output on standard list unit
C	List cross-references at end of assembly listing

P OPTION

Relocatable binary output is selected by the P option.

The standard output binary device is used for relocatable binary information. If the binary output device is magnetic tape, the final relocatable program terminates with the EOL record *T. If the binary output device is paper tape, a blank trailer terminates each assembly.

X OPTION

If the X option is selected, relocatable binary output is placed on the mass storage unit for subsequent loading and execution.

L OPTION

The L option results in an assembly listing described as follows. With the OPT pseudo instruction, any or all of the preceding options may be omitted. OPT also provides options for listing macro skeletons and abandoning assembly.

C OPTION

The C option produces a cross-reference list which is printed at the end of the assembly list.

Assembly Listing

The assembly list as output to standard list output device, consists of 18 columns of information related to the source statement, followed by a maximum of 80 columns listing the source statement (table G-24).

Each page has a header which contains the program name, page number, and date.

Following the assembly list, the lengths of the program, common, and data are given in hexadecimal and decimal values.

- PGM = 0155(341)
- COM = 2BE(702)
- DAT = 0000(0)

Table G-24. Assembly List

Column	Contents
1 through 4	Card number; truncated from five to four decimal digits
5	Space
6	Relocation designator for location P–program relocation D–data relocation
7 through 10	Location in hexadecimal
11	Space
12 through 15	Machine word in hexadecimal
16 through 17	Relocation designator for word P–program relocation -P–negative program relocation C–common relocation -C–negative common relocation D–data relocation -D–negative data relocation X–external blank–absolute
18	Space
19 through 98	Input source statement

The data length includes those areas reserved by DAT pseudo instructions.

ERROR LISTING

A list of errors which occur in passes 1 and 2 precedes the program listing on the standard list I/O unit. If the L option is selected, errors in pass 3 precede the source line on the list output. A decimal error count is printed at the end of each subprogram. If L is not selected, error messages are output on the standard comment unit.

The format for pass 1 and 2 error messages is given in table G-25.

Table G-25. Pass 1 and 2 Error Messages

Column	Contents
1 and 2	**
3 through 6	4-digit line number
6 and 7	**
8 and 9	Two-character error code
10 through 19	**********

The format for pass 3 error messages is given in table G-26.

Table G-26. Pass 3 Error Messages

Column	Contents
1 through 6	******
7 and 8	Two-character error code
9 through 18	**********

The error codes and their meanings are given on page G-50.

CROSS-REFERENCE LISTING

Cross-references are listed at the end of an assembly listing if the option C was specified by the user. Cross-references are also listed if no OPT statement was found since the C option is a default option.

The cross-references are divided into four functional parts:

- Equivalences
- Symbols
- Externals
- Symbols in alphabetical order

if cross-references are to be listed and there is not enough core to process all four parts of the cross-references listing, the assembler attempts to sort the symbol table alphabetically. If there is not enough core to sort the symbol table alphabetically, the symbol table is dumped.

The equivalences, symbols, and externals are listed according to the line number at which they are defined. In addition to the definition line number, the value or address and the line numbers of all references to that symbol are given. The list of symbols in alphabetical order includes all the symbols in the program. The number following each symbol is the corresponding definition line.

SAMPLE PROGRAM

The source program in figure G-33 results in the assembly listing in figure G-34.

```
         NAM      TEST2 ERS MACRO EXAMPLEX
XYZ      MAC      P1,P2,P3,P4,P5,P6
         LDQ      =N'P5','P6'
'P4'     LDA      'P3'
         'P1'     'P2'
         ADD      SYMB1
         IFA      'P5',NE,0
I1       IFC      'P1',EQ,MUI
         STA      SYMB3
         LDA      SYMB2
         EIF
         EIF      I1
         EMC
MACRO    MAC      P1,P2,P3,P4,P5,P6
         LOC      A
         LDA      'P1'
         'P2'     'P3'
         S'P4'Z   'A'-*-1
         JMP'P5'  'P6'
'A'      ENA      1
```

```
                EMC
                MACRO   SYMB1,STA,'SYMB2,I',
                MACRO   Q,*,LABEL1
        SYMB1   ADC     0
        SYMB2   ADC     0
                XYZ     MUI,,SYMB5,,3
        SYMB5   ADC     0
        CALL1   XYZ     MUI,'SYMB4,I',SYMB5,HERE,3,I
        SYMB4   ADC     0
        CALL2   XYZ     DVI,SYMB7,"SYMB8,I',THERE,2
        SYMB8   ADC     0
        SYMB7   ADC     0
                END
```

Figure G-33. Sample Source Program

SAMPLE LISTING

The assembly listing in figure G-34 is output from the assembly of the source program in figure G-33.

```
0001                       NAM      TEST2 ERS MACRO EXAMPLEX
0002               XYZ     MAC      P1,P2,P3,P4,P5,P6
0003                       LDQ      =N'P5','P6'
0004               'P4'    LDA      'P3'
0005                       'P1'     'P2'
0006                       ADD      SYMB1
0007                       IFA      'P5',NE,0
0008               I1      IFC      'P1',EQ,MUI
0009                       STA      SYMB3
0010                       LDA      SYMB2
0011                       EIF
0012                       EIF      I1
0013                       EMC
0014               MACRO   MAC      P1,P2,P3,P4,P5,P6
0015                       LOC      A
0016                       LDA      'P1'
0017                       'P2'     'P3'
0018                       S'P4'Z   'A'-*-1
0019                       JMP'P5'  'P6'
0020               'A'     ENA      1
0021                       EMC
0022                       MACRO SYMB1,STA,'SYMB2,I',
0023                       MACRO Q,*,LABEL1
0023 P0000 C800
     P0001 0006
0023 P0002 6900
     P0003 0005
0023 P0004 0141
******UD**********
******RL**********
0023 P0005 1000
0023 P0006 0A01
```

```
0024 P0007 0000 SYMB1  ADC      0
0025 P0008 0000 SYMB2  ADC      0
0026                   XYZ      MUI,SYMB5,,3
0026 P0009 E000
     P000A 0003
0026 P000B C800
     P000C 0009
0026 P000D 2400
     P000E 0000
0026 P000E 8800
     P0010 FFE6
******J********
0026 P0011 6400
     P0012 0000
0026 P0013 C800
     P0014 FFE3
0027 P0015 0000 SYMB5  ADC      0
0028            CALL1  XYZ      MUI,'SYMB4,I',SYMB5,HERE,3,I
0028 P0016 E100
     P0017 0003
0028 P0018 C800
     P0019 FFE8
0028 P001A 2900
     P001B 0007
0028 P001C 8800
     P001D FFE9
******J********
0028 P001E 6400
     P001F 0000
0028 P0020 C800
     P0021 FFE6
0029 P0022 0000 SYMB4  ADC      0
0030            CALL2  XYZ      DVI,SYMB7,'SYMB8,I',THERE,2
0030 P0023 F000
     P0024 0002
0030 P0025 C900
     P0026 0005
0030 P0027 3800
     P0028 0004
0030 P0029 8800
     P002A FFDC
0031 P002B 0000 SYMB8  ADC      0
0032 P002C 0000 SYMB7  ADC      0
0033                   END
```

Figure G-34. Sample Listing for Source Program

MNEMONIC INSTRUCTION CODES

Machine Instructions

There are six classes of machine instruction codes.

- Storage reference, group A
- Storage reference, group B
- Register
- Shift
- Skip
- Interregister transfer

STORAGE REFERENCE INSTRUCTIONS

These instruction codes and corresponding definitions are listed in table G-27.

Table G-27. Storage Reference Instructions

Operation Code		Definition
Group A	LDA	Load A register
	LDQ	Load Q register
	ADD	Add to the A register
	SUB	Subtract from A register
	ADQ	Add to Q register
	AND	Perform logical AND with A register
	EOR	Perform logical exclusive OR with A register
	MUI	Multiply integer with A register
	DVI	Divide integer into a Register
Group B	STA	Store A register
	STQ	Store Q register
	JMP	Unconditional jump
	RTJ	Return jump
	RAO	Replace add one in storage
	SPA	Store A register, return parity to A register

REGISTER INSTRUCTIONS

The register instructions are defined in table G-28.

Table G-28. Register Instructions

Operation Code	Definition
SLS	Selective stop
INP	Input to A register
OUT	Output from A register
ENA	Enter A register
ENQ	Enter Q register
INA	Increase A register
INQ	Increase Q register
NOP	No operation
EIN	Enable interrupt
IIN	Inhibit interrupt
EXI	Exit interrupt state
SPB	Set program protect bit
CPB	Clear program protect bit

SHIFT INSTRUCTIONS

Table G-29 lists the shift instructions and their corresponding definitions.

Table G-29. Shift Instructions

Operation Code	Definition
ARS	A right shift
QRS	Q right shift
LRS	Long right shift (Q and A combined)
ALS	A left shift
QLS	Q left shift
LLS	Long left shift (Q and A combined)

SKIP INSTRUCTIONS

Refer to table G-30 for the skip instructions.

Table G-30. Skip Instructions

Operation Code	Definition
SAZ	Skip if A=0
SAN	Skip if A≠0

Table G-30. Skip Instructions (cont)

Operation Code	Definition
SAP	Skip if A is positive
SAM	Skip if A is negative
SQZ	Skip if Q=0
SQN	Skip if Q≠0
SQP	Skip if Q is positive
SQM	Skip if Q is negative
SWS	Skip if switch is set
SWN	Skip if switch is not set
SOV	Skip on overflow
SNO	Skip on no overflow
SPE	Skip on storage parity error
SNP	Skip on no storage parity error
SPF	Skip on program protect fault
SNF	Skip on no program protect fault

INTERREGISTER TRANSFER INSTRUCTIONS

The interregister transfer instructions are listed and defined in table G-31.

Table G-31. Interregister Transfer Instructions

Operation Code	Definition
SET	Set specified register to ones
CLR	Clear specified register to zeros
TRA	Transfer A to specified register
TRM	Transfer M to specified register
TRQ	Transfer Q to specified register
TRB	Transfer both (Q+M) to specified register
TCA	Transfer complement of A to specified register
TCM	Transfer complement of M to specified register
TCQ	Transfer complement of Q to specified register
TCB	Transfer complement of both (Q+M) to specified register
AAM	Transfer arithmetic sum of A and M to specified register
AAQ	Transfer arithmetic sum of A and Q to specified register
AAB	Transfer arithmetic sum of A and both (Q+M) to specified register
EAM	Transfer exclusive or of A and M to specified register
EAQ	Transfer exclusive or of A and Q to specified register
EAB	Transfer exclusive or of A and both (Q+M) to specified register
LAM	Transfer logical product of A and M to specified register
LAQ	Transfer logical product of A and Q to specified register
LAB	Transfer logical product of A and both (Q+M) to specified register
CAM	Transfer complement of logical product of A and M to specified register
CAQ	Transfer complement of logical product of A and Q to specified register
CAB	Transfer complement of logical product of A and both (Q+M) to specified register

Note: + indicates an inclusive OR.

Pseudo Instructions

There are six classes of pseudo instructions.

- Subprogram linkage
- Data storage
- Constant declaration
- Assembler control
- Listing control
- Macro definition

SUBPROGRAM LINKAGE

For the subprogram linkage pseudo instructions refer to table G-32.

Table G-32. Subprogram Linkage Pseudo Instructions

Operation Code	Definition
NAM	Identify source language subprogram
END	End source language subprogram
ENT	Designate internal entry point names
EXT	Designate external entry point names
EXT*	Designate relative external entry point names

DATA STORAGE

Table G-33 defines the data storage pseudo instructions

Table G-33. Data Storage Pseudo Instructions

Operation Code	Definition
BSS	Define a block of storage starting at symbol
BZS	Define a block of zero storage
COM	Define a block of common storage
DAT	Define a block of data storage

CONSTANT DECLARATIONS

Refer to table G-34 for the definitions and codes for the constant declaration pseudo instructions.

Table G-34. Constant Declaration Pseudo Instruction

Operation Code	Definition
ADC	Store address constants
ADC*	Store relative address constants
ALF	Store an alphanumeric message
NUM	Store numeric constants
DEC	Convert and store decimal constants in fixed-point format
VFD	Variable field definition and storage

ASSEMBLER CONTROL

The assembler control pseudo instruction is listed in table G-35.

Table G-35. Assembler Control Pseudo Instructions

Operation Code	Definition
EQU	Equate symbols to addresses
ORG	Defines origin for assembly of instructions following ORG
ORG*	Terminate ORG
IFA	If condition is true, assemble following instructions
EIF	Terminate IFA (or IFC macro pseudo instruction)
OPT	Signal input of control options
MON	Return control to operating system

LISTING CONTROL

For the listing control pseudo instructions, refer to table G-36.

Table G-36. Listing Control Pseudo Instructions

Operation Code	Definition
NLS	Inhibit list output
LST	Resume list output after NLS
SPC	Space lines on list output
EJT	Eject page on list output

MACRO DEFINITION

Table G-37 defines the codes used in the macro definition pseudo instruction.

Table G-37. Macro Definition Pseudo Instruction

Operating Code	Definition
MAC	Specify name of macro
EMC	End macro definition

Table G-37. Macro Definition Pseudo Instruction (cont)

Operating Code	Definition
LOC	Define local symbolic labels
IFC	If condition is true, assemble following instructions in macro

PROGRAMMING CONSIDERATIONS

Programming considerations for COS are discussed in the following paragraphs.

Coding Techniques

The following limitations should be observed when coding program to run in 65K mode.

All 16 bits of an address word are needed to address all the available core. This means that bit 15 can no longer be used to indicate the conditions it can be used for in a 32K-mode system.

Multilevel indirect addressing cannot be used in 65K mode which signifies that instructions of the following form can no longer be used:

```
ADC     (TAG)
LDA+    (TAG)
```

If relative addresses are generated, the following instruction is allowed:

```
LDA     (TAG)
```

The following instruction is allowed in 65K mode if there are no storage instructions that make indirect reference to this location and the program containing this expression will never be loaded into part 1 of a 65K system.

```
ADC     (TAG)
```

ASCII CODES

The 1963 Control Data Subset of ASCII (CDC-STD 1.10.003, revision C) is used by the macro assembler. ASCII code uses 8 bits. The eighth bit, which is always zero, is omitted in table G-38.

Table G-38. ASCII Codes

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

Table G-38. ASCII Codes (cont)

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
CR	000 1101	D	Carriage return
SO	000 1110	E	Shift out
SI	000 1111	F	Shift in
DC$_0$	001 0000	10	Device control/data link escape
DC$_1$	001 0001	11	Device controls
DC$_2$	001 0010	12	
DC$_3$	001 0011	13	
DC$_4$ (STOP)	001 0100	14	Device control/stop
ERR	001 0101	15	Error
SYNC	001 0110	16	Synchronous idle
LEM	001 0111	17	Logical end of media
S$_0$	001 1000	18	Information separators
S$_1$	001 1001	19	
S$_2$	001 1010	1A	
S$_3$	001 1011	1B	
S$_4$	001 1100	1C	
S$_5$	001 1101	1D	
S$_6$	001 1110	1E	
S$_7$	001 1111	1F	
△	010 0000	20	Word separator (space)
!	010 0001	21	Exclamation point
"	010 0010	22	Quotation mark
#	010 0011	23	Number
$	010 0100	24	Dollar sign (hexadecimal)
%	010 0101	25	Percent
&	010 0110	26	Ampersand

Table G-38. ASCII Codes (cont)

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
' (APOS)	010 0111	27	Apostrophe
(	010 1000	28	Left parenthesis
)	010 1001	29	Right parenthesis
*	010 1010	2A	Asterisk
+	010 1011	2B	Plus
,(Comma)	010 1100	2C	Comma
-	010 1101	2D	Minus
.	010 1110	2E	Decimal point or period
/	010 1111	2F	Slash
0	011 0000	30	Numbers
1	011 0001	31	
2	011 0010	32	
3	011 0011	33	
4	011 0100	34	
5	011 0101	35	
6	011 0110	36	
7	011 0111	37	
8	011 1000	38	
9	011 1001	39	
:	011 1010	3A	Colon
;	011 1011	3B	Semi-colon
<	011 1100	3C	Less than
=	011 1101	3D	Equals
>	011 1110	3E	Greater than
?	011 1111	3F	Question mark
@	100 0000	40	Each
A	100 0001	41	
B	100 0010	42	

Table G-38. ASCII Codes (cont)

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
C	100 0011	43	Letters
D	100 0100	44	
E	100 0101	45	
F	100 0110	46	
G	100 0111	47	
H	100 1000	48	
I	100 1001	49	
J	100 1010	4A	
K	100 1011	4B	
L	100 1100	4C	
M	100 1101	4D	
N	100 1110	4E	
O	100 1111	4F	
P	101 0000	50	
Q	101 0001	51	
R	101 0010	52	
S	101 0011	53	
T	101 0100	54	
U	101 0101	55	
V	101 0110	56	
W	101 0111	57	
X	101 1000	58	
Y	101 1001	59	
Z	101 1010	5A	
[	101 1011	5B	Left bracket
\	101 1100	5C	Reverse slant
]	101 1101	5D	Right bracket
↑	101 1110	5E	Up arrow (exponentiation)

Table G-38. ASCII Codes (cont)

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
←	101 1111	5F	Left arrow (replaced by)
ACK	111 1100	7C	Acknowledge
†	111 1101	7D	Unassigned control
ESC	111 1110	7E	Escape
DEL	111 1111	7F	Delete/idle

†The numbers between 5F and 7C have no ASCII code assigned to them.

MACRO ASSEMBLER ERRORS

The macro assembler errors are defined in table G-39.

Table G-39. Macro Assembler Errors

Message	Significance
xxxxyy***********	Format for pass 1 and 2 error messages Where: xxxx–is a four-digit line number yy–is a two-character error code
******yy***********	Format for pass 3 error messages. If the L option is selected, errors in pass 3 precede the source line on the list output. If L is not selected, error messages are output on the standard comment unit
ABS BASE ERR	Assembler was loaded at a different location from where it was absolutized
**DS	Double defined symbol; a name in either: • The location field of a machine instruction or an ALF, NUM, or ADC pseudo instruction • The address field of an EQU, COM, DATA, EXT, BSS, or a BZS pseudo instruction
**EX	Illegal expression, either: • No forward referencing of some symbolic operands • No relocation of certain expression values • A violation of relocation • Illegal register reference

Table G-39. Macro Assembler Errors (cont)

Message	Significance
	• A symbol other than Q, 1, or B is specified
INPUT ERROR	An error was returned by driver when doing a read
**LB	Numeric or symbolic label contains illegal character; label is ignored
MASS STORAGE OVERFLOW	Not enough room for input image on mass storage
**MC	Macro call error: • Illegal parameter list • No continuation card where one was indicated
**MD	Macro definition error
**MO	Overflow of load-and-go area; affects only X option
**NN	Missing or misplaced NAM statement
**OP	Illegal operation code, either: • Illegal symbol in operation code field • Illegal operation code terminator
**OV	Numeric constant or operand value is greater than allowed
**PP	Error in previous pass of compilation assembly. See output page immediately preceding first page of listing for pass 1 or pass 2 error message
**RL	Illegal relocation, either: • Violation of relocation • Violation of a rule for instructions that requires the expression value to either be absolute or have no forward referencing of symbolic operands
**SQ	Sequence error – tags instructions with sequence numbers that are out of order; this is not fatal and is not counted in the number of errors reported at the bottom of the symbol table
**UD	An undefined symbol in an address expression

APPENDIX H

INDEX

Index

equipment

COMMENT SHEET

MANUAL TITLE CDC® CYBERDATA Operating System

Reference Manual

PUBLICATION NO. 22263100 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 references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

CUT ALONG LINE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications Department
Peripheral Subsystems Division
2200 Berkshire Lane
Plymouth, Minnesota 55441

CUT ALONG LINE

FOLD FOLD