60384400

CONTROL DATA CORPORATION

OPERATING SYSTEM REFERENCE MANUAL

CONTROL DATA®
STAR COMPUTER SYSTEM

REVISION RECORD

REVISION	DESCRIPTION
01	Preliminary printing
(1-31-73)	
A	Original edition
(10-1-73)	
B	Section 11 and appendix G are added with this revision. Other affected pages are: Front matter; 1-7,
(1-31-74)	1-8; 2-3 through 2-16; 3-1; 4-1 through 4-28, 4-31 through 4-34, 4-37 through 4-45; 5-4 through 5-32;
	6-1 through 6-7; 7-1 through 7-6; 8-1, 8-8 through 8-15; 9-1, 9-7 through 9-20; 10-1 through 10-27;
	C-1 through C-6; D-1 through D-3, D-9, D-13 through D-16; E-2, E-5 through E-7; Index-1 through
	Index-16; Comment Sheet.
C	Incorporates system improvements to reflect version 1.1 of STAR-OS. Information added includes STAR
(2-10-75)	Record Manager (new section 12); sequential record files added to appendix C; EDITT, a source line
	editor, added to section 9; and a new appendix H, containing system error messages and codes. The
	entire manual has been reprinted.
D	Incorporates enhancements to version 1.1 of STAR-OS. Appendix I is added with this revision.
(9-1-75)	
Publication No. 60384400	

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

Address comments concerning
this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the
back of this manual

LIST OF EFFECTIVE PAGES

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

Feature	Page	Revision
	Cover	–
	Title page	–
	ii thru ix	D
	1-1, 1-2	D
	1-3	A
	1-4	D
	1-5	A
	1-6	D
	1-7	B
	1-8	D
	2-1, 2-2	C
	2-3 thru 2-9	D
	2-10	C
	2-11 thru 2-13	D
	3-1	B
	3-2	C
	4-1 thru 4-3	D
	4-4	B
	4-5 thru 4-15	D
	4-16	C
	4-17 thru 4-19	D
	4-20	C
	4-21	D
	4-22	C
	4-23, 4-24	D
	4-25	C
	4-26	D
	4-27, 4-28	C
	4-29	D
	4-30, 4-31	C
	4-32	D
	4-33	C
	4-34 thru 4-36	D
	4-37	C
	4-38 thru 4-46	D
	5-1	C
	5-2, 5-3	A
	5-4	C
	5-5	B
	5-6	A
	5-7	B
	5-8	C
	5-9 thru 5-12	D
	5-13	D
	5-14	D
	5-15 thru 5-18	B
	5-19 thru 5-21	D
	5-22 thru 5-24	B
	5-25	C
	5-26	B
	5-27 thru 5-36	D
	6-1 thru 6-10	D
	7-1 thru 7-6	D
	8-1	D
	8-2, 8-3	C
	8-4 thru 8-11	D
	9-1 thru 9-8	D
	9-9 thru 9-14	C
	9-15	D
	9-16 thru 9-23	C
	10-1 thru 10-6	D
	10-7	C
	10-8	B
	10-9	C
	10-10, 10-11	B
	10-12, thru 10-14	C
	10-15, 10-16	B
	10-17 thru 10-20	D
	10-21	B
	10-22, 10-23	C
	10-24 thru 10-27	B
	11-1	D
	11-2, 11-3	C
	12-1 thru 12-13	D
	12-14	C
	12-15 thru 12-24	D

Feature	Page	Revision
	A-1	D
	A-2	C
	B-1	C
	B-2	A
	B-3	C
	B-4 thru B-10	A
	B-11	C
	B-12 thru B-19	A
	C-1	C
	C-2	D
	C-3	C
	C-4 thru C-9	D
	D-1	B
	D-2 thru D-7	D
	D-8	A
	D-9	B
	D-10 thru D-12	A
	D-13	B
	D-14 thru D-19	D
	E-1 thru E-6	D
	F-1	A
	F-2	C
	G-1	C
	G-2, G-3	B
	G-4	C
	G-5	B
	H-1 thru H-5	C
	H-6, H-7	D
	H-8 thru H-11	C
	H-12 thru H-22	D
	I-1 thru I-11	D
	Index-1 thru -9	D
	Cmt Sheet	D
	Return Env	–
	Back Cover	–

Feature	Page	Revision

PREFACE

This manual presents information of interest to the user of release version 1.1 of the STAR Operating System. Some of the significant additions to STAR-OS through this release are the Record Manager and the text editor, EDITT. Although documented in preceding editions of this reference manual, the library maintenance program UPDATE, and the checkpoint/restart feature were not usable with version 1.0 of STAR-OS.

Much of the information on which this document was originally based was derived from the University of California document UCID 30021.

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

CONTENTS

APPENDIXES

FIGURES

TABLES

INTRODUCTION TO STAR-OS 1

STAR-OS consists of a group of programs that comprise the operating system for the STAR-100, a virtual memory computer. The functions of input, compilation or assembly, loading, execution, and output of all programs submitted to the computer, as well as allocation of main memory, are monitored and controlled by STAR-OS. The system is file-oriented, and uses specifically allocated mass storage space as a backup for information processing in virtual space.

STAR-100 ARCHITECTURE

The STAR-100 is a large scale, high-speed computer with a significant number of new developments. Most important are the architectural concepts: virtual memory operation, distributive processing, and string array processing capability. Further, the system employs many advanced design concepts, such as stream processing, large high-speed register file, high input/output channel capacity, virtual bit addressing, large storage bandwidth, and a powerful instruction repertoire.

VIRTUAL MEMORY

Through the virtual memory system of the STAR-100, an apparently unlimited memory structure may be viewed as if it were entirely main memory. The STAR-100 hardware mechanisms manage system information in 65,536-word blocks (large pages) or in 512-word blocks (small pages). System software determines the block size to use in allocating main memory. A software mechanism ensures that the most frequently accessed pages exist in main memory, while unused pages are sent to slower backup media as necessary.

Virtual addresses are contained in a 48-bit format. When 512-word pages are addressed, the virtual page identifier is contained in 33 of the 48 bits; for large pages, the virtual page identifier requires only 26 of the 48 bits. Because unused virtual space imposes no burden on the system, the user may organize program addresses in almost any convenient manner.

Virtual addresses comprise the set assumed to exist by the programmer. Virtual addresses are translated into physical memory addresses by system software as needed when code is brought into the central processing unit. The system keeps track of the relationship between physical memory addresses and virtual memory addresses through a translator, called a page table. Each entry in the page table contains the virtual page address and the corresponding physical memory address, together with an access mode lock and other control information. A successful association between a virtual address and an entry in the page table causes that entry to be moved to the head of the table; all entries in between are moved down by one place. In STAR-100 the first 16 entries in the table are kept in high speed registers; the registers are examined in parallel with a simultaneous associative compare. An unsuccessful compare results in a sequential search through the remainder of the table held in main memory. Addresses of infrequently used pages automatically float to the end of the table.

If an address has no entry in the page table, various hardware sequences are initiated, and the program requesting the address is interrupted. Normally, the program monitor provides the space addressed by requesting the desired block be moved to main memory either from a storage station or from the paging station. The program is restarted, to continue processing from the point of interruption.

DISTRIBUTED SYSTEM

The computation unit of the STAR-100 is an autonomous central processor with input/output channel connections. Stations, consisting of a minicomputer, display/keyboard unit, small drum and buffer memory, handle peripheral processing functions and are linked to the STAR central processor. Slow speed input/output devices, terminals, magnetic tapes, etc., are grouped and connected to stations.

A station consists primarily of a small processor designed for data handling, rather than data processing; each has its own storage system and channels for handling the particular set of devices attached to it. Figure 1-1 shows the layout of a STAR system, with the connections between the various functional units.

A STAR central processor, with its immediate storage, can be likened to a data processing station within the system but with no particular priority over any other station. Two other stations, however, are closely associated with the central processor. The paging station, under control of the hardware virtual page mechanism and the operating system, provides temporary storage for programs exceeding available core space. The maintenance station, in addition to its functions of off-line fault diagnosis/repair and preventive checking, is capable of collecting detailed information about the STAR's performance.

STRING ARRAY PROCESSING

The STAR central processor includes several classes of instructions that can be used for conventional computing or string array processing. Conventional scientific and business data processing is performed by major high performance facilities that operate on floating point operands (64 or 32 bits) and on single bytes and bits. Some floating point instructions operate register-to-register; other operations on single bits or bytes are storage-to-storage.

STAR is also a vector processor. In STAR, a vector is defined as a contiguous set of bits, bytes, half words, or full words in virtual memory. The definition depends on how the contents of the virtual space may be treated by the vector processing instructions, rather than the nature of the contents.

Pipeline units operate on strings of operands, 64- or 32-bit arrays, byte strings, and bit strings. Information to specify the addresses of source and destination streams usually is held in the register file. Core storage system design accommodates two input operand streams and one output stream simultaneously, handled at logic speeds.

Many user functions provided by the string and array mechanism perform more complex operations on streamed data. Such functions amount to hardware macros and include, for example, polynomial evaluation, byte editing, scalar product of two vectors, sequencing by merging byte string records, and vector arithmetic on sparse vectors.

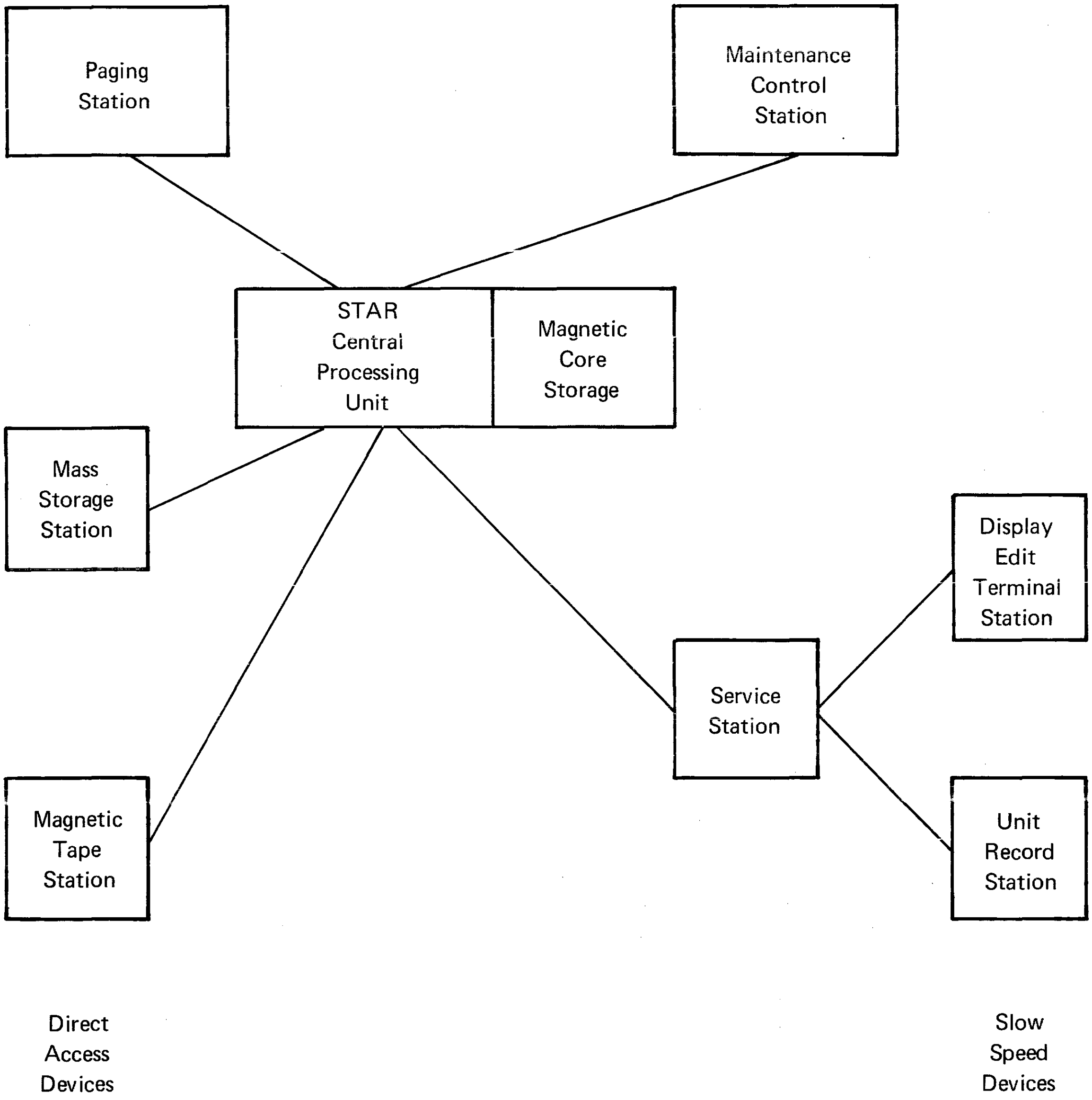


Figure 1-1. STAR System Showing Component Connections

The string instructions can operate on a maximum of 65,536 operands in one pass. Although the function is executed serially in a pipeline, it may be considered as being carried out in parallel on the data. Thus, the user has from one to 65,536 parallel processors at his disposal, depending upon how he achieves processing through code selection. Processing facilities allow for efficient computing in the conventional sense and also provide a fundamentally different approach to programming through string and array processing.

SYSTEM CONFIGURATION

The minimum STAR-100 configuration consists of a central processor interconnected to the standard magnetic core storage (MCS) unit, and four input-output channels.

The central processor contains all streaming and instruction control, arithmetic units, storage access control, and input-output communication control. The standard MCS contains 524,288 64-bit words of storage. The MCS has eight sections; each connects to 132-bit read and write data buses (128 data and 4 parity bits). Each MCS section contains four banks, giving a total of 32 multi-phased banks. An optional MCS, identical to the standard, provides a total system capability of 1,048,576 64-bit words.

A Maintenance Station, connected to any input-output channel, consists of a station processor having maintenance control and monitoring capabilities.

Each of three additional standard input-output channels provides 16-bit data communication and control to a station processor. Each station processor has a buffer controller and control circuitry connected to the corresponding peripheral equipment. Flexibility in the selection of peripheral equipment connected to the buffer controller is possible because the operating system software is modular, and only relevant portions are loaded into a particular station processor. (A typical station processor might be connected to a line printer, a card reader, and some magnetic tape units.) As shown in Figure 1-2, a total of 12 input-output channels (in groups of four) may be added to the system.

Further information about the STAR hardware may be found in the CONTROL DATA® STAR-100 Computer System Hardware Reference Manual (publication number 60256000).

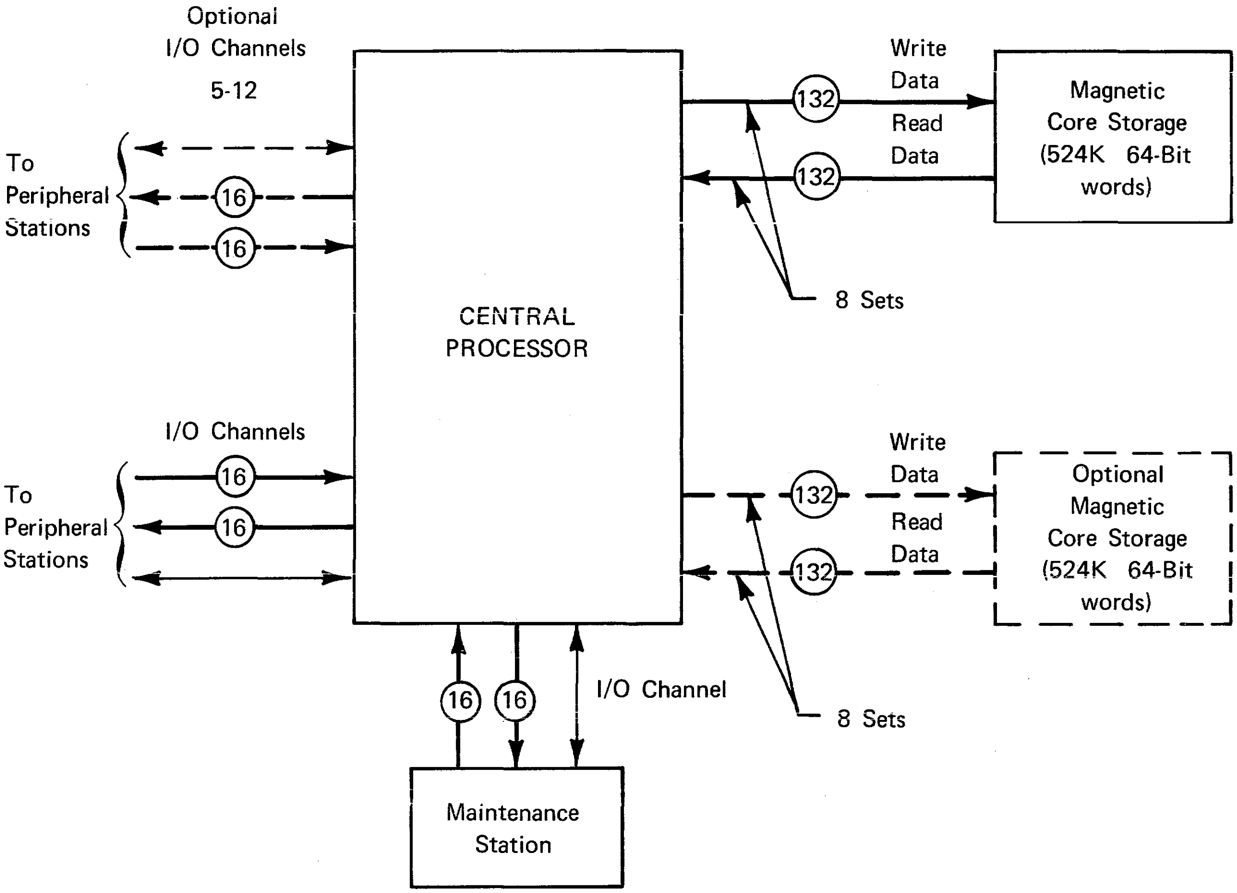
OPERATING SYSTEM PRECEPTS

STAR-OS is basically a timesharing system. Its programs provide the fundamental elements of an operating system, so that development of user-specific features will not impact the efficiency of the operating system. This basic approach should encourage users to fully explore the possibilities offered by the STAR-100 architecture.

Several precepts were followed in designing STAR-OS to allow a multitude of control languages, language processors, and utilities to be supported with a minimum of system overhead:

1. The operating system is file oriented. All user information in memory is allocated corresponding storage space on a mass storage device. Tasks can be entered automatically into and removed from main memory by the system as task management requirements dictate.

2. Communication between user and operating system is independent of the software conventions for any language processor or user process.



NOTES:

◄━► Data interconnections

←→ Control interconnections

—⑯— Number of bits in data transfers

All options are shown in dashed lines

I/O channel 1 is a special channel connected to the maintenance station

Optional I/O channels may be added to the system in increments of four channels. For example, I/O channels 5-8 would be added as a group

Figure 1-2. Central STAR – 100 Configuration

3. The operating system provides rudimentary, function-oriented system messages. A user process can issue messages to produce system functions, such as input and output, file manipulation, resource allocation, and message handling.

4. No limitations are placed on input/output device utilization. Language processors can implement their access methods and data structures without incurring additional system overhead.

5. Basically, the operating system provides the means for a user to identify himself, cause execution of a set of code on an existing mass storage file, and signify the end of the control sequence. A set of executable files with global access (public files) will replace the conventional control language.

6. One program can initiate and run another program – an important adjunct to developing message interface routines and batch processors.

OPERATING SYSTEM UTILIZATION OF STAR ARCHITECTURE

VIRTUAL MEMORY CONCEPTS

Generally, programmers must be aware of the total main memory required for holding and executing any given program. Programs that exceed the amount of main memory must be redesigned into separate sections, and execution planned so that individual sections can be called in as needed from auxiliary storage to overlay other sections whose functions are complete or currently not needed.

The virtual memory and associated paging scheme of the STAR-OS frees the programmer from this concern; the operating system software assumes this responsibility. It manages allocation of storage between main memory and auxiliary memory, moves information from auxiliary to main memory as needed, and translates virtual memory addresses to physical addresses in main memory.

STAR-OS considers every program to be executable only in virtual memory. Data files may be defined either by a set of virtual addresses or by physical mass storage addresses, and will be translated to appropriate virtual addresses.

DISTRIBUTED SYSTEM CONCEPTS

The central operating system and the peripheral operating system each consists of a resident system and a non-resident callable set of tasks. Various portions of the system communicate through messages. Figures 1-1 and 1-2 show the organization, distribution, and communication paths of the operating system.

The resident central operating system has two parts: the KERNEL, responsible for time slicing and message handling and the PAGER, responsible for memory management and page swapping. The non-resident set of central tasks comprises the virtual system; it controls user and job entries and the allocation of resources to jobs. In addition, the virtual system contains such functions as file management, explicit input and output, and terminal message input and output.

A special class of virtual system routines – privileged user tasks – include periodic system accounting and file routing and disposition. Typically, the duration of these tasks is much longer than for virtual system tasks, and they require explicit input and output.

The resident peripheral operating system is called the NUCLEUS; common to all station processors, each NUCLEUS uses its set of non-resident tasks to control peripheral equipment. Operating system tasks for each station processor are stored on its own microdrum.

The NUCLEUS consists of simple diagnostic routines, a system deadstart program, driver programs for the microdrum and keyboard/display, organizational program, programs to manage the system overlay mechanism, and the SCANNER which is the main control and organizational program.

Non-resident tasks are concentrated into larger processing routines to facilitate on-line error handling and maintenance procedures common to all stations. Further, station functions are grouped into different systems to minimize system tables. Any one system contains only those routines necessary to its job.

STRING ARRAY PROCESSING CONCEPTS

The internal structure and organization of the operating system fosters efficient processing of string array and vector instructions. For example, many tables can be searched by viewing the table as a full word array that can be streamed through a pipeline processor. In floating point operations, vector concepts and the streaming of vector information through STAR's two parallel pipeline processors reduces individual program computation time and allows several programs to be in various stages of execution at the same time.

FILE ORIENTATION

STAR-OS allows two modes of input-output: explicit and implicit. Explicit input/output is accomplished with functions such as READ and WRITE which provide a conventional manner of data transfer between user-defined buffers and tape or mass storage.

Implicit input/output is accomplished by the operating system when the user causes an access interrupt by referencing a page of data or code not in central memory. If the virtual page has been previously associated with physical space, the system will transfer the data between central memory and the physical device. If a virtual-to-physical relationship has not occurred previously, the system defines the virtual page in free space so that it becomes an extension of program space. When doing implicit input/output, the virtual address can be considered a symbolic reference to the mass storage auxiliary memory.

STAR-OS also recognizes two types of files; virtual, containing data or code; and physical, containing data only. A virtual file is prefaced by a 512-word block containing control information to be used by the operating system if the file is to be accessed in the implicit mode. This preface is known as the minus page. The bound implicit map is part of the minus page and relates a set of virtual addresses to a set of mass storage addresses allocated for the file. The bound implicit map has an entry for every discontinuity in virtual address space and physical space.

The system allows physical and virtual files to be opened in either the implicit or explicit mode. When opening a virtual file in the implicit mode, the user has the option to use the mapping information from the file minus page, or to map the file into a new set of virtual addresses. Since a minus page does not exist on a physical file, the user must supply the mapping information when opening a physical file in the implicit mode.

When virtual files are opened, the user must be aware that the first block on the file is the minus page. When either physical or virtual files are opened in the explicit mode, the system makes entries in the bound explicit map. These entries are used by the input/output system when the user makes explicit input/output requests for the file.

Only virtual code files that contain programs may be submitted to the system for execution, in which case the loader is responsible for generating the minus page for a virtual code file. The user must generate the minus page for a virtual data file.

A program can be swapped out of main memory to facilitate memory management. Corresponding mass storage space is provided for each virtual address space. As a program is put into execution, the operating system automatically creates a file to contain any modified pages of the program file, any free space attached, and any read-only data space defined to have temporary write access. This file, called a drop file, has a minus page with an area reserved for the drop file map.

PERMANENT FILES

Within the operating system, processing proceeds on a task basis. A task may create permanent, as well as transitory files. Permanent files allow information to be retained from task to task. Transistory files are retained until the task which created them is no longer active. When a task terminates without saving its drop file, it no longer has the option of restarting the task. Refer to the TERMINATE system message described in section 5. Tasks which are still active are exemplified by interrupted tasks and system-terminated tasks. After one or more related tasks are terminated, files may remain which are no longer required by the user. These files can be destroyed by using the DESTROY utility described in section 9.

SYSTEM STRUCTURE 2

STAR-OS is divided into four parts:

Resident System The resident system runs in a privileged mode, called monitor mode; it is always resident in core and references memory by absolute addresses, rather than through the virtual paging mechanism. When the processor is in monitor mode, interrupts are inhibited, and some extra instructions are enabled.

Virtual System Tasks The virtual system tasks run in user mode, and reference memory by virtual address. They communicate with the resident system by using reserved messages, and they may modify system tables.

Privileged User Tasks These tasks have the same characteristics as virtual system tasks, but they may not modify system tables directly. They perform tasks that require more time than virtual system tasks.

Peripheral System The peripheral operating system runs in the station processors.

RESIDENT SYSTEM

The resident portion of the operating system is comprised of the KERNEL, which handles alternator management (time slicing) and message communication, and the PAGER, responsible for main memory allocation and page swapping.

The time slicing portion of the KERNEL is controlled by an alternator loop that acts as a circular table with various indicators in each table entry. Indicators include a pointer to a minus page table entry, a descriptor block entry, and three sets of flag bits: one set is used by KERNEL, one by virtual system usage, and one set is shared. These bits define the status of each entry in the alternator loop.

A unique entry in the alternator loop is shared by all virtual system tasks. To prevent two routines from modifying the same system table simultaneously, only one system task is allowed to run at a time. This system alternator slot has highest priority; it always is run unless the slot is blocked for input/output or PAGER action. If this slot is empty, it is filled with the task at the head of the system demand task queue. When the queue is empty, the slot is filled with the task at the head of the system task queue; and when that queue is empty also, a user task can run.

(System demand tasks are critical to the efficient operation of the system for such operations as paging or job disconnection; system tasks are not considered critical.)

User jobs, privileged user tasks, and virtual system tasks communicate messages to the KERNEL through the Exit Force instruction (a machine language instruction). PAGER communicates messages by direct calls. The peripheral system communicates with the KERNEL by setting pointers in the station queuing structure and without external interrupts. The KERNEL communicates with the peripheral system by setting pointers and setting station channel flags.

All communications between the various portions of the system are by messages. Messages either pass through the KERNEL, in which case it acts as a message switcher, or are processed directly by the KERNEL.

All access interrupts, as well as certain messages dealing with core allocation, are passed to the PAGER by the KERNEL. The PAGER dynamically allocates both large and small pages, and performs all required implicit input/output necessary to free memory pages and obtain the pages causing access interrupts. PAGER operates in a demand mode; if an overload in page swapping occurs, PAGER disconnects one or more jobs from the alternator.

VIRTUAL SYSTEM TASKS

The virtual portion of the system controls entering users and jobs into the system, ordering jobs by priority, and entering and removing jobs from the alternator loop. In addition, it contains routines for system file management, explicit input/output, and terminal message handling. Virtual system demand tasks are queued by one of two occurrences:

Bits are set in one or more alternator slots to indicate virtual system action is required.

PAGER requests the KERNEL to queue a virtual system demand task.

Virtual system tasks are queued by one of three occurrences:

Communication from the service station requires processing.

User job requests a system service not provided by the resident system.

Entry in the periodic table indicates it is time to run a virtual system task.

Within the respective queue, all virtual tasks have equal priority and are run on a first in/first out basis.

PRIVILEGED USER TASKS

Privileged user tasks run under special user identification; they can make either normal user calls or privileged system calls, and can modify system tables only through calls.

PERIPHERAL SYSTEM

The peripheral system for all station processors has two parts:

A resident basic system, the NUCLEUS, common to all stations

A set of overlays for performing tasks related to the responsibilities of the individual stations.

The NUCLEUS is controlled by the SCANNER. This program uses scan bits and an associated table to determine which routines to call. If a particular routine is not in core, a resident overlay driver calls in the routine from the station's microdrum.

STAR JOB MANAGEMENT CONVENTIONS

Tables used by STAR-OS to control job processing within the system may be affected or altered by user programs. In all cases, access to the tables must be through system messages.

FILE INDEX

The File Index is a catalog of the mass storage files for all active users in the system. The catalog for inactive users are maintained on a mass storage unit. When a user becomes active, his catalog is brought into the active table. Each entry in the File Index consists of a 16-word table, of the following format:

1	buser 64							
2	name 64							
3	ptrpfil 16	packid 48						
4	unit 8	type 8	act 8	siev 8	unused 16	iodlen 16		
5	torg 32	tlr 32						
6	acs 8	lok 8	bva 48					
7	owndiv 32	fact 24	fgive 4	mcat 4				
8	ref 16	f 1	f 2	f 3	f 4	unused 12	fi 1mcat 4	pfi ptr 28
9	slen 16	saddr 18	unused 30					
12	slen 16	saddr 18	unused 30					
13	slen 16	saddr 18	unused 6	pri 12	unl 12			
14	slen 16	saddr 18	un2 30					
15	slen 16	saddr 18	* 1	* 1	fifc 12	fiic 4	fiec 4	fidc 8
16	slen 16	saddr 18	sid 16	tid 12				

*	ficre 1	fidef 1

buser Binary user number or pool name. The entire word containing this field is replaced with the pool name when file is given to a pool.

name File name in ASCII. File names must be of format described under File Names in section 4.

ptrpfil Pointer to pack file index

packid Six-character identifier of pack on which file resides

unit Logical unit number

type File type:

0 = Physical data file

1 = Virtual data file

2 = Virtual code file

Note: File type does not imply file access mode. See mode field of CREATE and OPEN calls.

act Number of active input/output connectors for this file

slev Security level of this file (0-255)

lodlen Length of drop file in number of pages, if different from file length, except for files given to USER1. For these files, this field contains the binary user number of the original owner of the file

torg Time in 16-second units at which file originated

tlr Time in 16-second units at which file was last referenced by opening it

acs File access permission:

1 = Write access

2 = Read access

May be used in logical combinations

lok File lockout protection:

0 = No access locked out

1 = Write access locked out

2 = Read access locked out

4.= Execute access locked out

May be used in logical combinations

bva Base virtual address of file's first word

owndiv ASCII code identifying file owner (from user directory)

fact ASCII account designator for charge liability

fgive File acquisition method:

0 = User created this file

1 = User was given this file

mcat File management category:

0 = Permanent file

1 = Scratch file

2 = Output file

3 = Write temporary file

5 = User-generated drop file

6 = System-generated drop file

7 = Batch file

ref Count of number of references to this file made by opening it

f1 File index modified bit (imod)

f2 File index destroy bit (ides)

f3 Name changed or file given bit (nefg)

f4 Pack file index delete bit (idel)

filmcat File management category associated with first OPEN command for this file

pfiptr Block number relative to the first block of the permanent file index

slen Length of physical segment in small pages

saddr First page address of physical segment

pri Priority level for a file to be output as its originating terminal or front end processor

un1
un2 A seven character user number from the front-end processor. The two high order 6-bit characters are stored in un1; the five low order 6-bit characters are stored in un2.

ficre Reserved for future system usage.

fidef A bit indicating that the file is to be routed when it is released.

fiic A numeric value in the file index, indicating the internal format of the file. The numeric values and their corresponding mnemonic representations are as follows:

Value	Mnemonic	Format
0	AS	8-bit ASCII
1	BI	binary
2	PA	8-bit ASCII with ASCII control characters

fiec A numeric value in the file index, indicating the external print or punch representation of the file. The numeric values and their corresponding mnemonic representations are as follows:

Value	Mnemonic	Format
0	29	029 Keypunch format
1	26	026 Keypunch format
2	80	80-column binary format
3	*B	STAR binary format

fidc A numeric value in the file index indicating how the file is to be disposed. The numeric values and their corresponding mnemonic representations are:

Value	Mnemonic	Disposition
0		No disposition
1	PR	Print on any available printer
7	PU	Punch
D	IN	Input for Batch processing
E	SC	Scratch; destroyed at end of task

sid A three-character (6 bit) site identifier for the front-end processor.

tid A two-character (6 bit) identifier for a terminal

MINUS PAGE

All virtual files in the system must have a minus page preface; it consists of 512-word segment (small page) containing information to control program execution and data access. (Physical files do not have a minus page.)

The operating system uses a minus page during program execution to store such information as the execute and interrupt invisible packages, time-slicing data, input/output connection blocks to high-speed storage devices, maps of defined virtual space, time sharing data, page fault statistics, etc.

For a virtual code file, the loader creates and fills the minus page. For a virtual data file, the minus page and associated maps must be created and filled by the user. Since the minus page is part of a physical file structure, it may be created or altered with explicit input/output messages.

MINUS PAGE LAYOUT

Words	Name	Explanation
0-24	pip	Executing program invisible package
25-27	ros	Program restart temporary buffers
28-29	slot	Time information required by operating system for alternator and message management
30-31	ros	As above
32-56	iip	Interrupt program Invisible package
57-63	ros	As above
64-127	uioc	Input/output connectors for user disk or tape files
128-131	sioc	Input/output connector for source file
132-135	dioc	Input/output connector for drop file
136-138	mapp	One-word directories (each) for bound explicit map, bound implicit map, and drop file map
139	err	Error number of fatal error condition, and address where it occurred
140-150	time	Time usage and accounting entries

Words	Name	Explanation
151	eio	Explicit input/output information
152-159	iadd	Interrupt address stack for input/output and terminal interrupts
160-175	bpfm	Bound explicit maps (of files opened for explicit input/output)
176-255	bvfm	Bound implicit maps (of files opened for implicit input/output)
256-511	dfm	Drop file map

INPUT/OUTPUT CONNECTORS

An input/output connector is a four-word block used to establish a link between the program and an input/output device. The operating system uses an input/output connector to keep track of the activity of a specific file and a program's use of that file. Each time a program issues a create or open file request, an input/output connector is created.

Each program may have up to 16 connectors, numbered 0 to 15. The program source file is numbered 16 and the program drop file is 17.

Formats of the input/output connectors are shown below:

Input/output connector for a mass storage file opened for explicit input/output:

name 64									
mcat 3	mode 2	iok 3	pmp 16	nmp 8	length 16	unit 8	acs 4	unused 2	own 2
bad1 32					bl1 16	unused 16			
bad2 .32					bl2 16	unused 16			

Input/output connector for a mass storage file opened for implicit input/output

name 64							
mcat 3	mode 2	iok 3	unused 40	unit 8	acs 4	unused 2	own 2
unused 64							
pma 18	len 16	iocn 5	unit 6	con 3	lma 16		

For a drop file, word four of the input/output connector serves the same purpose as the second word of a bound implicit map entry for a user or a source file. Word four of a user or source file IOC is zero.

Input/output connector for magnetic tape:

<table>
<tr><td colspan="5">name 40</td><td colspan="3">unused 24</td></tr>
<tr><td>mcat 3</td><td>mode 2</td><td>lok 3</td><td colspan="3">unused 40</td><td>unit 8</td><td>unused 8</td></tr>
<tr><td colspan="4">bad$_1$ 32</td><td colspan="2">bl$_1$ 16</td><td colspan="2">unused 16</td></tr>
<tr><td colspan="4">bad$_2$ 32</td><td colspan="2">bl$_2$ 16</td><td colspan="2">unused 16</td></tr>
</table>

Fields in input/output connectors:

name — Name of the file in ASCII. File names must be in the proper format as described under File Names in section 4.

mcat — File management category:

0 = Permanent file

1 = Scratch file

2 = Output file

3 = Write temporary file

4 = Magnetic tape file

mode — Mode of input/output

0 = Open for explicit input/output

1 = Open for implicit input/output

lok — File lockout protection:

0 = No access locked out

1 = Write access locked out

2 = Read access locked out

4 = Execute access locked out

} May be used in logical combinations

pmp — Pointer to bound explicit map for this file

nmp — Number of bound explicit map entries for this file

length — Length of file segment in small page blocks

unit — Logical unit number

acs — File permission granted:

1 = Write access

2 = Read access

} May be used in logical combinations

own File ownership: (See section 4)

0 = Private
1 = Public
2 = Pool

bad1 Virtual page address of first input/output buffer

bl1 Length in pages of first input/output buffer

bad2 Virtual page address of second input/output buffer

bl2 Length in pages of second input/output buffer

pma Physical mass storage sector address of this file's first page

len Length of file in small page blocks

iocn Input/output connector (IOC) number

con Control field of the following format:

C1	C2	C3

C1 = 1	Write access permitted
C2 = 1	Read access permitted
C3 = 0	File is contained on small pages
C3 = 1	File is contained on large pages

lma Logical mass storage sector address of this file's first page

MAP DIRECTORIES

Each map directory contains information relating to the location and length of its file map. Each directory occupies the second half-word of its location in the minus page. The bound explicit map directory is at word 136; the bound implicit map directory is at word 137; the drop file map directory is at word 138. Each is formatted as follows:

Unused 32	Count of entries 8	Pointer to first entry of this type of map 24

EXPLICIT INPUT/OUTPUT INFORMATION

Word 151 of the minus page contains information required by the operating system for processing explicit input/output messages. It is formatted as follows:

IO1	IO2	IO3	IO4	IO5	IO6	lgpg	smpg
8	8	8	8	8	8	8	8

IOn — Six explicit input/output requests may be outstanding at any time, one request for each IOn field. An explicit I/O message may have up to eight Beta† requests; an outstanding Beta input/output request is indicated by an IOn bit being on.

lgpg — Number of large pages with input/output outstanding

smpg — Number of small pages with input/output outstanding

INTERRUPT ADDRESS STACK

Words 152 - 159 are used by the operating system for processing interrupts from terminals and input/output requests. Words 152 - 157 are related to the six 8-bit IOn fields in word 151. Each of the six words contains control fields and a pointer to the Alpha† word for the I/O request. The stack is formatted as follows:

Words					
152-157	IOr 8	ist 8	ctl 6		alfwd 42
158	IOr 8	ist 8	a 1	4	cerp 43
159	IOr 8	ist 8	a 1	4	ceep 43
No. of Bits				4	43

IOr — Last I/O Beta request processed or in process

ist — One bit for each of the eight potentially outstanding Beta requests, or one bit for the controller or controllee of this program. A set bit indicates an interrupt is stacked.

ctl — Control bits of the following format:

C0	C1	C2	C3	C4	C5

C0 = 1 GIVE UP CPU message issued for this request by the main level.

C1 = 1 Unused.

†Alpha and Beta words are conventions through which programs communicate under STAR-OS. Their general formats are described in section 3. Since the use of certain fields within Alpha and Beta words varies between messages, each field is described under the appropriate system message, in sections 4 and 5.

C2 = 1 GIVE UP CPU message issued for this request by the interrupt level

C3 = 1 Unused

C4 = 1 Program currently in interrupt level for this request

C5 Unused

alfwd Alpha word pointer input/output request; if there is an interrupt address, it is in the first Beta word.

a If this bit is set, only messages preceded by the characters (sc)I will interrupt.

cerp Controller interrupt program address counter (half-word virtual address).

ceep Controllee interrupt program address counter (half-word virtual address).

MINUS PAGE FILE MAPS

The file maps in the minus page relate files to physical mass storage areas, and for files opened for implicit input/output, they relate physical mass storage areas to virtual address areas.

BOUND EXPLICIT MAPS

The bound explicit maps are related to files opened for explicit input/output, mode = 0. Each explicit file input/output connector has a map entry; positions are related through the connector number. Format of the bound explicit map entry is:

pma	length	iocn	unit	con	lma
18	16	5	6	3	16

pma Physical mass storage sector address of this file's first page

length Length of file in small page blocks

iocn Input/output connector (IOC) number

unit Logical unit number

con Control field of the following format:

C1	C2	C3

C1 = 1 Write access permitted

C2 = 1 Read access is permitted

C3 = 0 File is contained on small pages
= 1 File is contained on large pages

lma Logical mass storage sector address of this file's first page

BOUND IMPLICIT MAPS

A bound implicit map is related to files opened for implicit input/output (mode=1), which may consist of discontinuous virtual address ranges. Up to 40 virtual address space segments may be mapped simultaneously. All the segments may be associated with one input/output connector, or each segment may be so associated. In any case, each segment points to the input/output connector with which it is currently associated. The format for a bound implicit map entry is:

unused 17	vpa 32	unused 15

pma 18	length 16	iocn 5	unit 6	con 3	lma 16

vpa Virtual page address of this segment

pma Physical mass storage sector address of first page of this segment

length Length of this segment in small page blocks

iocn Input/output connector (IOC) number associated with this segment

unit Logical unit number

con Control field (see con under Bound Explicit Maps)

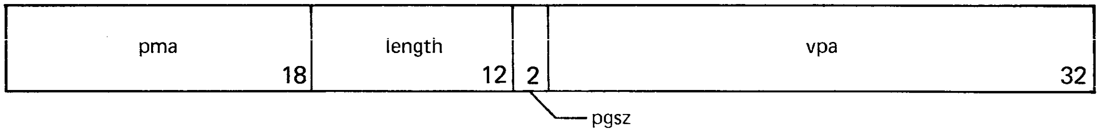
C3 = 0 Indicates small page segment
= 1 Indicates large page segment

lma Logical mass storage sector address of first page of this segment

In bound implicit map entries, all first words are in the first half of the map space, and all second words are in the second half. Entries are sorted by ascending virtual address; blank entries are squeezed out.

DROP FILE MAP

The drop file map is related to free space attached by the program, modified source pages, modified write temporary pages, the program minus page, and the program virtual page zero. It always is associated with input/output connector number 17. Each drop file map entry consists of one full word and one half word. Up to 170 entries may be made in the drop file map, and each entry may have up to 31 associated pages. The 170 full word entries occur first in the map space. The format of these entries is:



pma Physical mass storage sector address of first page of this space

length Length of this space in small page blocks

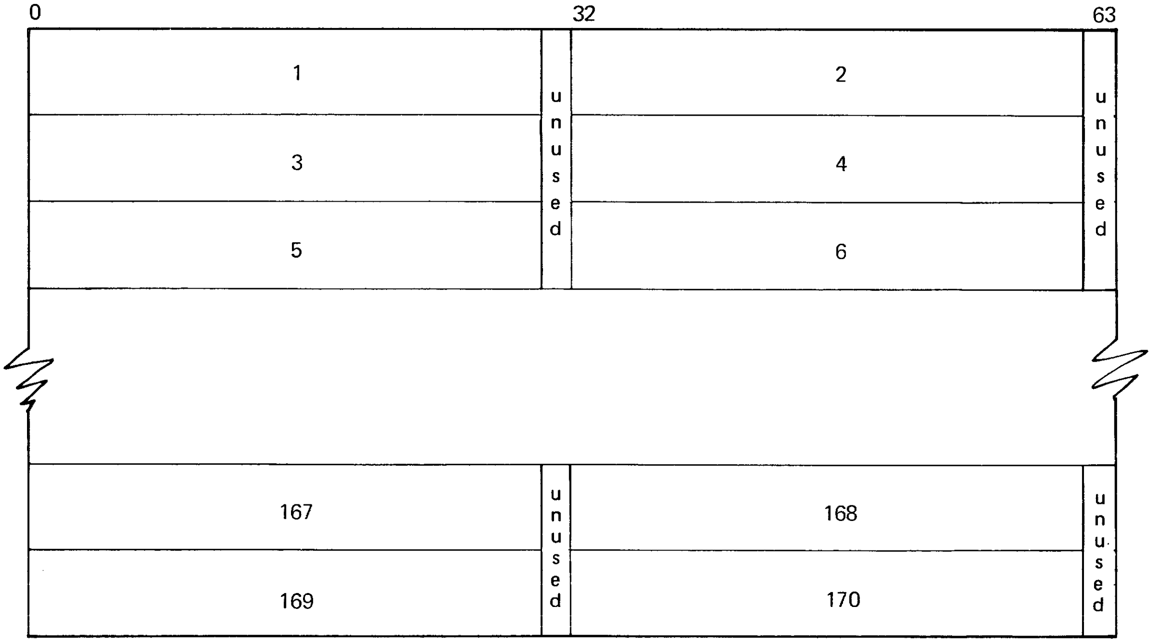
pgsz Size of pages in this space

0 = small

1 = large

vpa Virtual page address of first page in this space

The 170 half-word entries that follow the full-word entries correspond as follows:



Each half-word entry consists of a bit corresponding to a page in the entry. If the bit is zero, the page is either undefined or exists in main memory or on the paging device; if the bit is one, the page has been written to mass storage on the drop file. Bits 0 and 32 represent page 1 of a segment; bits 30 and 62 represent page 31 of a segment.

SYSTEM MESSAGES 3

ALPHA AND BETA WORDS

User programs may issue messages which result in the performance of system functions. To issue a message, the user. presets a 2- or more word block according to the Alpha and Beta conventions, and performs an Exit Force instruction† that transfers control to the KERNEL, and forces the operating system to change to monitor mode.

Immediately following the Exit Force instruction in the instruction stream is either a 32-bit indirect or a 64-bit direct message pointer. Hexadecimal format of indirect message pointer:

OOEEOOrr

rr is the register (see appendix E) containing the virtual address of the message.

The hexadecimal format of direct message pointer:

OOFFaaaaaaaaaaaa

a's are the virtual address of the first full word of the message.

The message has a two-part standard format. The Alpha (first portion, specifies the function to be performed, length of parameter list, and where to proceed for error processing. The Alpha portion has the same general format for all messages.

The Beta (second) portion, contains the parameters. The format of the Beta portion depends on the function, as described later for each function code. Alpha and Beta words must start on full word boundaries and may not exist in the user's page zero. Rather, they must exist in virtual space which has read/write or write temporary access. They may not cross large page boundaries.

When a message is processed without error, the operating system returns control to the next half or full word immediately following the message pointer. Thus, calls can be chained by placing one message pointer directly behind another.

†STAR standard instruction. A complete list of instructions is included in appendix B. Exit Force can be used to switch between monitor and job mode.

Alpha format:

	0	16	32	48 63
Alpha (1)	r	len	c	f
Alpha (2)	n	eea		
Alpha (3) (optional)	Bl	Ba		

r Hexadecimal response code returned by the operating system when message has been processed. If no error occurred, the response code is zero (exceptions: f=0013, f=0016 and f=0017). The significance of a non-zero response code varies as described for each function code.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion. Otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion.

c This field varies with the message; usually, it specifies function options or controls.

f Specifies function to be performed (hexadecimal message code).

n May specify option or control, may contain a parameter for the message, or may be a parameter returned during message processing.

eea Virtual bit address that receives control if error occurs. This address must lie within the program issuing the message. If eea = 0, the error is considered fatal to the further execution of this user process.

B1 If the Beta and Alpha portions are not contiguous (len = FFFF), this parameter indicates Beta length in full words.

Ba If Beta and Alpha portions are not contiguous (len = FFFF), this parameter indicates virtual address of Beta portion's first full word.

CONTROLLER AND CONTROLLEE

The terms "controller" and "controllee" have specific meaning relative to STAR-OS. For example, a batch processor also controls actions of a user program; the former becomes the controller and the latter, controllee. This relationship between programs can exist in other ways as well: one program can initialize and/or direct the actions of another.

FILE SYSTEM AND INPUT/OUTPUT MANAGEMENT 4

Since STAR-OS is a file-oriented system, file management is an important aspect of the operating system. Although STAR-OS takes little direct responsibility for action on a given file, a set of user messages allows a fair degree of latitude in directing STAR-OS processing for a given file. These messages are detailed later in this chapter.

FILE CREATION

The five management categories are: permanent, scratch, write temporary, drop, and output.

Permanent file creation and disposition is controlled by the originating user. The operating system protects these files from access or destruction by any other user; however, the operating system may replace a file at the request of a privileged user task.

Scratch file may be created only by a user program. They exist during the originating program's activity. When a program terminates normally, all scratch files are destroyed unless they are open to other programs of that user. When the operating system terminates the program or the program terminates and saves its drop file, scratch files are saved. A message to close a scratch file destroys it. All scratch files have read/write access.

Write temporary files have read-only access when in mass storage. When brought into main memory, such a file may be modified; the modified image becomes a part of the drop file. Subsequent references to that page address will access the modified page. The original source image may be referenced again only by removing the modified image from the drop file.

The drop file is that mass storage space set aside for writing the altered pages of an executing program. When a new program is started, the operating system automatically creates a drop file. If no pages have been written to the drop file, a program may create its own drop file (any pre-existing drop file will be destroyed). The drop file is preserved upon abnormal termination and may be preserved or destroyed at the option of the program upon normal termination.

Output files contain information suitable for processing to some special output device, such as a printer, card punch, or microfilm device. These files may be created only by a user program. Upon normal program termination, the operating system gives all properly named output files to system privileged user tasks for processing to output devices. After the files have been processed, they are destroyed. If the file name is not proper for an output device, it will remain as a private file. A message to close a properly named output file will cause the operating system to give the file to a system privileged user task for output processing.

FILE NAMES

File names may be up to 8 letters and/or numbers long and must start with a letter. They must be left-justified and blank filled. Currently, the length of tape file names is restricted to not more than five letters and/or numbers.

The drop file name will be taken from the source file name. To become the drop file name, it is shifted right two characters, and the new first (leftmost) character created is a number based on the suffix of the logged on user. (If suffix is A, number will be 1, if suffix is B, number will be 2, etc.) The second character created is the controllee level number of the program as follows:

1 Job control processor
2 Program initiated by terminal or job control processor
3 Program initiated by Level 2 program
4 Program initiated by Level 3 program
5 Program initiated by Level 4 program

DEVICE TYPE OUTPUT FILES

Output files are further categorized by output device. Each device type output file has its own system privileged user job to handle writing tasks. Output files must be created by the user or given to user number 999999 through a GIVE FILE or ROUTE message. Constraints on file names for each device type are as follows:

Device	Name Length (maximum characters)	First Letters
Line Printer	8	P
STAR Binary Punch	8	B
ASCII Punch	8	A
026 Punch	8	AS
029 Punch	8	AN
Unformatted Punch	8	U

For the line printer and punch, output files can be saved in families for consecutive processing. Family names must have the same first-letter conventions shown above. For punch files, the second and third characters must be either decimal numbers 0 to 9 or the character X. Punch family files are held in an unprocessed state until a file name with X in both the second and third character is encountered; then, the family is processed as a unit. For print files, the family name can be a single or double digit name. A single digit family name is PDFAMNAM where P=P, D=either decimal numbers 0 to 9 or the character X, and FAMNAM=family name, the first character of which must be any letter except X. A double digit family is PDDFAMNM where P=P, D=either decimal numbers 0 to 9 or the character X and FAMNM=the family name. PRINTOUT family files are held in an unprocessed state until a file name with X in the second and third character (second character only for a single digit family) is encountered; then, the family is processed as a unit. File names beginning with AN or AS may not be grouped in families.

FILE OWNERSHIP

File ownership may be public, private, or shared private.

Public files are accessible to the entire body of users. Public files are intended to be the assemblers, compilers, and other general purpose routines that generally augment the operating system for a particular installation. The names of public files constitute a kind of job control language for the operating system.

Private files are accessible only to the originating user. Only he may manipulate the contents of the file, the file access privileges, the security level, the lifetime of the file, etc. The operating system maintains the right to control utilization of file resources; for instance, it retains the right to delete files based on lack of activity.

Shared private files are accessible only to a specific subset of users as governed by the subset member who controls file integrity and disposition for this group. Such a group of files is called a pool; the controlling user is known as a pool boss.

ACCOUNTING FOR FILE OWNERSHIP

Each private disk file cataloged in the system is recognized as belonging to a particular user number and account identifier. Liability for file ownership depends on its account identifier. When a file is given by one user to another with the GIVE FILE message, the user number associated with that file is changed immediately. Liability for the file, however, remains with the giver until the recipient references the file. At the first reference to the given file, the account identifier is changed to that of the file recipient; and the system accounting tables will indicate the total time file ownership was held by the originating account identifier.

FILE ACCESS CATEGORIES

File access parameters define what may be done with a given file. The access categories under STAR-OS are read, write, and execute.

A mass storage file having read only access may not be written upon. Any explicit attempt to do so will produce an error, as will any attempt to modify a page from a read only virtual file. Writing on the pages of read only files in memory can occur if the user opens a file having a write-temporary management category. Such modified pages will be written to the drop file.

A write access on a file allows the user to modify the file at will. Essentially, no restrictions are imposed on file modification under this type of access.

An execute only access parameter means that the file can be accessed only for execution. Users may not read or write an execute only file. Combinations of read, write, and execute file access parameters are permitted. Generally, public files are read/execute files. Only an operating system task or a privileged user task can alter a public file.

ACTIVITY

Each file has an activity counter, which is set to zero initially. A file is considered active if a program, active in the system, has the file open. A program may open the same mass storage file in more than one input/output connector (IOC). Each open call increments the file activity counter. The activity counter is decremented for each CLOSE call and at program termination for each IOC pointing to the file.

A file may be given to another user only if the file activity counter is zero. The file may be destroyed if the activity counter is zero, or if the counter is one and the file is open to the user attempting to destroy the file.

INPUT/OUTPUT

Input and output on mass storage may be implicit or explicit. Implicit input/output is the normal mode of STAR-OS; it uses the file map to associate virtual address space with mass storage space. Obtaining data from a virtual address is an implicit read from a mass storage file; storing data into a virtual address is an implicit write to a mass storage file. (The operating system handles the explicit transfer between main memory and mass storage.)

Explicit input/output requires specific user action; one or more buffers in the program's virtual space are defined and associated with a region of mass storage. After making the association, the program may write the virtual space to mass storage or read the mass storage into virtual space. To access another part of the file, the mass storage region is redefined, leaving the virtual space buffer fixed.

FILE MANAGEMENT MESSAGES

SIMPLE FILE CONTROL

The most commonly used system messages deal with file creation, opening and closing, and the map function.

The CREATE FILE message reserves space on mass storage, names the space, and defines parameters for it. For a physical file, the user program's minus page has enough information to initiate explicit input/output functions; if so specified, a virtual base address can be associated with this file for implicit input/output functions. If a virtual base address is specified, the file may later be used in implicit mode. For a virtual file, sufficient information is stored in the user program's minus page to associate a virtual base address with the first mass storage address. Continuous virtual address space is assumed.

The OPEN FILE message connects the user to a pre-existing mass storage file for explicit or implicit input/output functions. In opening a file, the user may accept the parameters given to the file when it was created; or he may alter them if he has permission. Both physical and virtual files may be opened for either kind of input/output. Once opened for explicit input/output, however, a file cannot be accessed implicitly, and vice versa; however, a file may be open in several input/output connectors at the same time – some for explicit input/output, and others for implicit input/output.

The MAP message makes virtual address space accessible to a program. This space may be associated with an open mass storage file, or it may be free space. Implicit access is established by a map-in function, which puts into the program minus page a map relating virtual address space and the mass storage space of some open file. For free space, the relationship is established through the program drop file.

Virtual address space is released with the map-out function. Before the associated virtual space is mapped out for a mass storage file with write access, all modified pages of that space are rewritten on the original mass storage file. If the file did not have write access, the modified pages are lost. Virtual address space that has been mapped out is no longer accessible by the program. The mass storage file, itself, is not closed, but the mass storage space corresponding to the mapped out virtual space becomes available for redefinition. The MAP message applies only to files opened in the implicit mode; it has no significance for files opened in the explicit mode.

At the conclusion of processing on a mass storage file, the connection between program and file is severed by a CLOSE FILE message. When a file is opened for implicit input/output and write access is closed, any modified pages of the file are rewritten on mass storage. If the file did not have write access, modified pages are lost. When the file is closed, the program no longer has access to the file through the severed connection, although other unsevered input/output connections may remain. Virtual address space associated with a file is no longer defined when the file is closed.

CREATE FILE (f = 0001)

With the CREATE FILE message, a program reserves space on or access to an input/output device and defines the connection to that device. Unless otherwise noted, all values in the following description are hexadecimal.

Message format:

Alpha (1)	r 16	len 16	unused 16	0001 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	name 64							
Beta (2)	IOC 8	mcat 8	type 8	lok 8	acs 8	mode 8	slev 8	unit 8
Beta (3)	packid 48						frag 8	ss 8
Beta (4)	length 16		bva 48					

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1 = Error code was returned in an ss field of Beta

211 = Number of creates in this message is illegal (n = 0 or n > 16)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) exists and contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion of the message

n Number of creates in this message; maximum = 16

eea Virtual bit address to receive control if error occurs while this message is processed (r ≠ 0). If eea = zero when an error occurs, the error is considered fatal

B1
Ba
If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

name
File name in ASCII. File names must be of the proper format. See section 4.

IOC
Input/output connector (IOC) number (0-15 or 17)

mcat
File management category:

0 = Permanent file

1 = Scratch file

2 = Output file

3 = Write temporary

4 = Magnetic tape file

5 = User-generated drop file. Return the drop file name in Beta (1).

7 = Batch file

For mcat = 0 through 4, standard file name conventions apply; for mcat = 5, drop file name returned begins with a number.

type
Type of device or file:

0 = Physical file

1 = Virtual data file

2 = Virtual code file

7 = Seven track tape

9 = Nine track tape

lok
File lockout protection:

0 = No access locked out

1 = Write access locked out

2 = Read access locked out

4 = Execute access locked out

May be used in logical combinations

acs
File access permission:

1 = Write access permission

2 = Read access permission

May be used in logical combinations

mode Input/output mode:

0 = Open for explicit input/output

1 = Open for implicit input/output

slev Security level (1 to 255) to be given to the file if non-zero. If zero, use security level belonging to caller of this message.

unit Logical unit number

packid Six-character identifier of pack on which the file is to be created. When the call is made and this field is binary zeros, the system creates the file on a system pack where space is available. The packid of the pack will be returned to this field.

frag Reserved for future system usage.

ss Error response field:

01 = File already exists

02 = No available mass storage space for this file

03 = mcat is illegal

04 = Parameter or format error occurred

05 = Operator-initiated tape error occurred

06 = Input/output connector is already in use

07 = File index is full

08 = Standby job may not create a tape

09 = Invalid file name

0A = ss was preset

0B = Existing drop file cannot be destroyed; dmap full or file open to another PP

0C = Illegal type field

0D = For mcat = 05, pages have been written already to the existing drop file; no new file was created

0E = For mcat = 05, drop file length is not adequate to hold space already in the drop file map

0F = Unable to find requested packid

15 = Bound implicit map area in minus page is full

16 = Virtual address overlap

length Length of file in small page blocks

bva Base virtual bit address; correspond to first word of this file

Notes:

1. The name parameter in Beta is not required for mcat = 05.
2. The IOC, type, acs, mode, and slev parameters in Beta are set by the operating system when mcat = 05.
3. Beta (4) is not needed for a tape create request; however, all requests must provide four Beta words.
4. If a tape requested is not currently known to the operating system, a message to the operator requests the tape. The program issuing the tape request is set at the point of re-issuing the call and is removed from main memory. When the operator makes the tape known to the operating system, the program is reactivated.

OPEN FILE (f = 0003)

The OPEN FILE message is issued by a program to connect itself to a pre-existing mass storage file. If a file is opened in the explicit mode, the specified IOC in the program minus page is filled in as required, and an entry is made in the explicit file map area of the minus page. This allows initiation of explicit input/output. Before actual input/output operations can begin, however, the EXPLICIT I/O message must be used to access the file physically.

When a physical file is opened in implicit mode, the specified IOC in the program minus page is completed, and an entry is made in the implicit map area of the program minus page. No entry is made in the bound explicit map. Explicit input/output cannot be accomplished on a physical file that is opened in implicit mode. A physical file opened in implicit mode is considered to begin at the virtual address given in the working virtual address field of the system call. The virtual address space represented by the file is considered contiguous over the entire length of the file beginning at file word zero. When the open is complete, the user may map out the space just defined and map in the file in any manner. All implicit input/output attributes normally pertaining to virtual files are applied to physical files in implicit mode.

When a virtual file is opened in implicit mode, the IOC in the program minus page is filled in; and, optionally, implicit map entries are completed. If the file maps are to be used as recorded with the file on disk, the entries are simply copied to the bound implicit maps. If the file maps are to be copied into the program's call buffer, only the IOC is filled in, no implicit map entries are made, and the program cannot access the file implicitly. This kind of open call should be succeeded by a map-in call that tells the system how to relate virtual space to the physical disk file. A user also may map a file into contiguous space (beginning at word zero of the minus page) according to parameters supplied with the call.

When a virtual file is opened in explicit mode, all input/output must be done explicitly through the program's buffers. The IOC specified in the program minus page is filled in, and one entry is made in the explicit map. The file is mapped, beginning with word zero of the file minus page. With this call, sufficient information is recorded to allow the program to initiate explicit input/output. No implicit access is possible to any of the virtual space usually represented by the file when it is opened in explicit mode.

Message format:

Alpha (1)	r 16	len 16	unused 16	0003 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	name 64										
Beta (2)	IOC 8	map 8	C1 1	mcat 3	C2 1	type 3	lok 8	acs 8	mode 8	slev 8	unit 8
Beta (3)	packid 48	own 2	st 4	w 2	ss 8						
Beta (4)	length 16	wva 48									
Beta (5)	blength 16	bva 48									

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1 = Error code was returned in an ss field of Beta (3)

211 = Number of opens in this message is illegal (n = 0 or n > 16)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len=FFFF,Alpha (3) exists and contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to be located beginning at Alpha (3) and len is the length of the Beta portion of the message

n Number of files to be opened in this call; maximum = 16. At times, it may be more efficient to open more than one file at a time. When this is to be done, the Alpha portion for the OPEN message is used once, with n = the number of files to be opened; this is followed by groups of Beta words, one set per file.

eea — Virtual bit address to receive control if error occurs while this message is processed (r ≠ 0); if eea = 0 when an error occurs, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the first full word of the Beta portion.

name — File name in ASCII. File names must be of the proper format as described in section 4. If format is not proper, error response 21 is returned in SS field.

IOC — Input/output connector (IOC) number (0-15)

map — Determines the disposition of maps residing in minus page of virtual file being opened for implicit input/output.

0 = Read minus page from the file and obtain mapping information to be inserted into bound implicit map area of the program minus page.

1 = Read mapping information from minus page of file and return maps to the address specified in Beta (5). The IOC is filled; no entries are made into bound implicit map area of the program's minus page. File is not mapped in.

2 = Do not read minus page from file; obtain mapping information according to W option of this message.

A program requesting delivery of the implicit maps must supply a buffer to hold the entire map. Although the operating system will not store entries beyond the end of the buffer supplied, neither will it squeeze out empty map entries. The map format in the program's buffer is two words per map entry as follows:

unused 17	vpa 32	unused 15

pma 18	length 16	iocn 5	unit 6	con 3	lma 16

vpa — Virtual page address

pma — Physical mass storage sector address

length — Length of this segment in small page blocks

iocn — Input/output connector (IOC) number

unit — Unit

con — Control

lma — Logical mass storage sector address

See Bound Implicit Maps, pg. 2-12

The C1 and C2 options in word Beta (2) of the OPEN message enable the user to modify fields in the file index. Permission to modify these fields is granted by the system if the file ownership is private; pool, and the user is the pool boss; or if the file is public and the user is priviledged.

C1 = 0 Open file as specified in mode field, but do not change the file index type

C1 = 1 Open file as specified in mode field and change the file index type to that in the type field.

mcat File management category to be associated with the file

0 = Permanent file
1 = Scratch file
2 = Output file
3 = Write temporary file

Copies mcat field into the mcat field of the input/output connector

C2 = 0 Open file as specified in mode field but do not change the file index access

C2 = 1 Open file as specified in mode field and change file index access to that specified by acs and lok fields

type File type; if C1 = 0, the operating system will return the file type to this field; if C1 = 1, change file index type to:

0 = Physical data
1 = Virtual data
2 = Virtual code

lok File lockout protection field to be set into the file index if C2 is 1. If C2 is zero or if a public file is opened, this field is set from the lok field in the file index. Whatever value is set in this field is also set into the IOC.

0 = No access locked out
1 = Write access locked out
2 = Read access locked out
4 = Execute access locked out

May be used in logical combinations

acs — File access permission granted. A logical AND is performed using the complement of the lok field in the file index and this field as operands; the result returned to this field and to the IOC determines the access granted for this OPEN call. If C2 is 1, the result is also stored in the acs field of the file index.

1 = Write access permission
2 = Read access permission
} May be used in logical combinations

mode — Input/output mode:

0 = Open for explicit input/output

1 = Open for implicit input/output

slev — Security level of this file (0–255); set by the operating system

unit — Logical unit number; set by the operating system

packid — Six-character identifier of pack on which file exists; set by the operating system

own — File ownership; set by the operating system

0 = Private

1 = Public

2 = Pool

st — Management category of the file; set by the operating system

0 = Permanent file

1 = Scratch file

2 = Output file

3 = Write temporary file

5 = Drop file created by the user

6 = Drop file created by the system

7 = Batch file

w — Location of working virtual address to be used by the system when a physical type file is opened for implicit input/output. If map=2, this field is also used when opening a virtual type file for implicit input/output.

w=0 — Working virtual address is specified in field wva of this message. Use the length specified in this message if the size is non-zero and less than the length value given in the file index; otherwise, use the length specified in the file index.

w≠0 — Working virtual address and length to be used by the system are in the file index. They are returned to the wva and length fields of this message by the system.

ss Error response field:

0 = Normal completion

21 = Either no name was given, or the name was not in the file index

23 = Virtual address overlap

24 = Input/output connector is already in use or not 0 through 15

25 = Lockout protection, access code, or file type is illegal

26 = Mass storage error occurred while reading the minus page

27 = Bound implicit map area in minus page is full

28 = File requested has a security classification higher than that of caller

29 = bva + blength is greater than maximum user virtual address

36 = More than one virtual address overlap. Usually indicates an uninitialized minus page.

length Length of this file is small page blocks; set by operating system or as indicated under the w option of this message when w=0.

wva Working virtual bit address corresponding to virtual start address of the file

blength
bva If the user requests file maps to be delivered to his program space (map = 1), these fields specify the length and virtual address where the operating system will store the maps; if blength = 0, the maps will not be transferred to the user's program space.

MAP (f = 0004)

The MAP message gives a program access to virtual space by connecting it with mass storage space. The mass storage space may be in a file already opened or it may be free space not associated with a file. This message may also be used to release virtual space which may be redefined later.

In order to implicitly access virtual space, the definition of that space must be in the implicit map area of the program minus page. This can be done by using MAP with the map-in option. Up to 40 discontinuous address regions may be cataloged with MAP. The user relates some virtual starting address and length with some disk address of an open file and indicates the access rights pertaining to that virtual region. The system makes the necessary entries in the implicit space map of the program. Overlaps of space are signaled as an error. If all 40 entries of the map are full, an error is signaled and no further map-in calls are permitted until some space is released with a map-out. There is sufficient data available as a result of this call to allow the system to process page exceptions for the defined space. In the case of a free space attachment, the defined virtual space is given a part of the program drop file on which it may reside if a core-to-disk swap becomes necessary. Free space attachments, therefore, are not given an entry in the bound implicit map, but are cataloged in the program drop file map. This map can hold up to 170 entries of up to 31 pages each. This allows for as many as 170 non-contiguous address spaces to be part of the drop file.

The map-out option allows for release of virtual address space. This may be a release of space associated with an open disk file or a release of free space. Virtual address space which has been mapped out is no longer accessible to the program. The corresponding disk region may be redefined in other virtual space, but the disk file itself is not closed; i.e., the IOC is left intact. Mapping out free space causes the corresponding drop file map entries to be deleted and frees the disk space for reassignment. If the disk file region represented by a virtual space has write access and is mapped out, all modified pages of that space will be written on that disk file before the map-out process is complete. If the disk file itself did not have write access, all modified write temporary data is lost through the map-out process.

The MAP call is illegal when dealing with files opened for explicit input/output.

All values given below are hexadecimal unless otherwise specified.

Message format:

<table>
<tr><td>Alpha (1)</td><td>r
16</td><td>len
16</td><td>c
16</td><td>0004
16</td></tr>
<tr><td>Alpha (2)</td><td>unused
16</td><td colspan="3">eea
48</td></tr>
<tr><td>Alpha (3)</td><td>Bl
16</td><td colspan="3">Ba
48</td></tr>
</table>

<table>
<tr><td>Beta (1)</td><td colspan="4">vpa
32</td><td colspan="4">lma
32</td></tr>
<tr><td>Beta (2)</td><td colspan="2">length
16</td><td colspan="3">unused
24</td><td>IOC
8</td><td>con
8</td><td>ss
8</td></tr>
</table>

r Response code returned by the operating system when this message has been processed. If no error occurred, the response code will be zero; otherwise:

1 = Error (See ss field)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message; otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion.

c Option field:

0 = Map-in file specified in the IOC field

1 = Complete map-out of file specified in the IOC field

2 = Map-out drop file only

eea Virtual bit address where control transfers if an error occurs during message processing ($r \neq 0$). When eea is zero, an error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

vpa First virtual small page address for space defined

lma Logical mass storage sector address in file associated with virtual page address

lma = FFFF Free space will be appended as defined by virtual page address and length fields.

length Length of virtual range in small page blocks. If this call is not for free space, the space on the mass storage file must be contiguous.

IOC Input/output connector (IOC) number for mass storage file being mapped

0-15	Pointer to mass storage file being mapped
16	Must be specified for source file map-out
17	Must be specified for free space map-in or map-out

con A set of eight bits providing control information for pages formatted as follows:

c1	c2	S/L	c4	c5	wa	ac

c1,c2,c4,c5		Not used
S/L	= 0	Small page
	= 1	Large page

The following two fields are examined by the system when mapping-in files associated with IOC 0 through IOC 15:

wa	= 0	Get access rights determined when the file was opened from input/output connector
	= 1	Get access rights from the ac field if the lok field from the IOC permits
ac	= 0	No read or write access
	= 1	Read access
	= 2	Write access
	= 3	Read/write access

ss Error response field:

0 = Normal completion

1 = Virtual address overlap in bound implicit map

3 = Length field in a map-out message is zero or greater than the length in map

4 = Sector count in large page request is not modulo 128

5 = IOC does not exist or mode of IOC is not implicit

6 = Virtual address is the same as that of an existing ADVISE call

7 = Bound implicit map was full at map-in

8 = Logical mass storage address plus length exceeds file length

9 = Page requested for map-out is locked down

A = Space is undefined at map-out

B = Map entry is a large page but virtual address is not

C = Bound implicit map is full at map-out

D = Input/output connector is not proper for a free space request

E = Free space map is full at map-out

F = -Drop file is too small for free space map-in; not enough room in drop file for free space

10 = Mass storage file index cannot be found

11 = Virtual address overlap exists in free space map

12 = For map-in, no read or write access requested; map-in not done

CLOSE FILE (f = 0005)

A program issues the CLOSE message to sever its connection to a mass storage file. Existence of the mass storage file is not affected, but some file attributes in the file index may be modified.

When a file is closed, the file is given to the output processor if the activity count is zero and the management category is output. The first character of the file name indicates the particular output processor, described under DEVICE TYPE OUTPUT FILES, at the beginning of this section.

Message format:

Alpha (1)	r 16	len 16	unused 16	0005 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	IOC 8	mcat 8	C1 1	C2 1	C3 1	C4 1	type 4	lok 8	acs 8	flag 8	unused 8	ss 8
Beta (2)	length 16		bva 48									

r Response code returned by the operating system when this message has been processed. If no error occurs, response code will be zero. Other values are as follows:

1	=	Error response has been returned to ss field of Beta
211	=	Number of files specified in this call is illegal (n = 0 or n > 16)
214	=	Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3); and len is the length of the Beta portion

n Number of files closed by this message. Maximum = 16

eea Virtual bit address where control transfers if an error occurs during message processing (r ≠ 0). When eea is zero, the error is considered fatal

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters will indicate the Beta portion's length and virtual bit address .

IOC Input/output connector (IOC) number of file being closed

The following flags can cause changes to be made in the file index if the file ownership is private; or pool and the user is the pool boss; or, if the file ownership is public and the user is priviledged.

mcat File management category of the file being closed; stored in the file index if flag = 2

0 = Permanent file

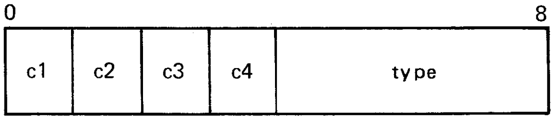
1 = Scratch file

2 = Output file

3 = Write temporary file

5 = User created drop file

7 = Batch file



c1 = 0 Close file, but do not change file type in file index

= 1 Close file and change file type in file index to type

c2 = 0 Close file with no change to file index, file access, and/or lockout.

= 1 Close file and change file index, file access, and lockout to that given in acs and lok field of this request.

c3 = 0 Close file; do not change drop file size

= 1 Close file and change file index; drop file size to that given in length field of this request

c4 = 0 Close file; do not remove drop file length from file index

= 1 Close file and remove drop file length from file index

type = 0 Physical data file

= 1 Virtual data file

= 2 Virtual code file

lok File lockout protection; stored in file index if c2 = 1

0 = No access locked out
1 = Write access locked out
2 = Read access locked out
4 = Execute access locked out

} May be used in logical combinations

acs File access permission stored in the file index if c2 = 1:

1 = Write access
2 = Read access

} May be used in logical combinations

flag Flag for special action:

0 = Ignore contents of second Beta word of this entry and the mcat field.

1 = Change file index base virtual address to that given in bva field of this request.

2 = Change file index file management category to that given in mcat field for this request.

3 = Do both flag = 1 and flag = 2.

ss Error response code:

1 = Input/output connector was not for a mass storage file.

2 = Input/output connector number was out of range.

3 = Attempt to alter a public file index entry. Input/output connector is cleared, but no information is altered in file index (file closed).

4 = File type, access right, or lockout specified by this request is illegal (file closed).

5 = File whose close was requested, but a file page is still locked down.

6 = A scratch or output file is open to another program of this user (file closed to this PP but not destroyed or given).

7 = Invalid name for a file with a management category of output (file closed).

8 = Specified IOC was not open

9 = Drop file map full

length Length of drop file in small page blocks; set into the file index if c3 = 1

bva Base virtual bit address corresponding to first word of mass storage file; set into the file index when flag = 1 or 3.

Notes:

1. This call cannot be used to close drop files or source files (IOC = 16 and 17).

2. When the file being closed has write access, modified pages are rewritten in mass storage before the close function is completed. When the file is write protected, modified pages are deleted before the close function is completed.

3. All outstanding input/output requests are completed before any file index changes are made. The file index entry will exist in its new state only at the completion of the CLOSE function.

EXPLICIT I/O AND INTERRUPT

The operating system supports one level of software interrupt for the program. The user can issue explicit input/output requests for mass storage or magnetic tape, continue processing, and be notified when the input/output operation has been completed. The program can be interrupted by messages from its controller or job control processor; but interrupts must be enabled before the operating system will take action on the specified conditions. Two system messages, EXPLICIT INPUT/OUTPUT (f = 0050), and PROGRAM INTERRUPT (f = 001C), accomplish the enabling function.

The program may specify interrupt on either successful or unsuccessful completion of an input/output request. The user may issue from one to eight input/output requests with each message. Up to six requests may be outstanding for a program at any one time. Within a message, requests may be contingent on other requests; and since requests are issued sequentially, the contingency must be upon previously issued requests. Interrupt upon completion of a request, therefore, may be contingent upon the completion of other requests. The operating system is responsible for checking all possibilities in determining whether the interrupt condition has been met.

Interrupt processing involves the program's minus page. The minus page has space for two invisible packages: one for current execution (level zero), and the other for the interrupt routine (level one). When the program is interrupted, control is passed to the virtual address specified by the user when he enabled the interrupt. The level one invisible package becomes the current invisible package; the level zero invisible package is saved in the minus page. The operating system saves the register file image for level zero and places, in register three, a pointer to the Alpha portion of the message and an index to the Beta portion causing the interrupt. The interrupt routine may use this information as necessary. The operating system puts into register 1E the length and address of the data base to be established, provided that information was supplied in the request for interrupt. Initializing the other registers is the responsibility of the interrupt routine. (A detailed discussion of the register file appears in appendix E.)

Since the interrupt routine (level one) cannot be interrupted by any other software interrupts, it will run until it issues a RETURN FROM INTERRUPT message (f = 0051). The current interrupt is then released and its invisible package is lost. The level zero invisible package becomes current, and its register file image is restored by the operating system. All information from the level one register file is lost.

When interrupts occur and the interrupt routine is already in control, the operating system stacks the interrupt information in an area of the program's minus page. When the interrupt routine issues a RETURN FROM INTERRUPT message, level zero is not restarted. Instead, the next interrupt on the stack is taken. This process is repeated until the stack is empty.

An option in the RETURN FROM INTERRUPT message allows level one to become the new level zero after all additional interrupts stacked for this and any other level one routines have been processed. In this case, the register file image for level zero will be lost. The new level zero may have its own level one interrupt routines.

Additionally, so that processing of other programs will not be delayed while a given program is awaiting completion of an I/O operation, another message is available: GIVE UP CPU UNTIL I/O COMPLETES. It suspends a user program until an I/O operation has signalled completion.

EXPLICIT INPUT/OUTPUT (f = 0050)

Input/output for magnetic tapes cannot be done implicitly. The EXPLICIT INPUT/OUTPUT message provides an explicit capability of transferring specific blocks of data. The parameter (Beta) portion of the message may contain a combination of buffer definitions, data transfer requests, and an interrupt request. If an interrupt request exists, it must be in the first two Beta words. Double buffering within a file is easily accommodated and encouraged as a practice.

Message format:

Alpha (1)	a	r 15	len 16	unused 16	0050 16
Alpha (2)	j 8	b 8	eea 48		
Alpha (3)	Bl 16	Ba 48			

Interrupt Request

Only one interrupt request per Alpha message is allowed; it must be the first Beta word-pair associated with the Alpha message. The request defines the interrupt and data base addresses to be used when the INT field of a data transfer request is either 1, 2 or 3.

Beta (1)	op=5 8	unused 8	ia 48
Beta (2)	dbl 16	dba 48	

Data Transfer or Function Request

Beta(1)	op=1, 2, or 3 8	sop 8	busy 1	unused 7	IOC 8	cerr 8	serr 24			
Beta(2)	con 6	int 2	tpm 8	issu 1	unused 7	ret 8	unused 3	bits 4	tape 1	fadd 24

Buffer Definition Request

Beta (1)	op=4 8	sop 8	busy 1	unused 7	IOC 8	cerr 8	unused 24
Beta (2)	blen 8	unused 16	badd 40				

a This bit is cleared by the operating system to indicate to the caller when the Alpha and Beta words are no longer required. The caller should set this bit if this feature is to be utilized.

r Response code returned by the operating system when this message has been processed. If no error occurs, a response code value of zero is returned; otherwise:

1 = Illegal interrupt address

2 = More than eight requests

4 = Error occurred in one or more requests

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If the value of len is FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion

j Index indicating which Beta entry was last processed or is currently in process. This field is stored by the operating system

b Index indicating which Beta entry caused the interrupt; this field is stored by the operating system

eea Virtual bit address where control transfers if an error occurs during message processing (r ≠ 0). When eea is zero, the error is considered fatal

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word

op Operation to be performed:

1 = Read

2 = Write

3 = Function

4 = Buffer operation

5 = Enter interrupt

ia Virtual bit address where control transfers upon interrupt (such as end of tape operation)

dbl Length of data base to be established if an interrupt occurs

dba Address of data base to be established if an interrupt occurs

sop Sub-operation to be performed:

op = 1,2

1 = Buffer 1

2 = Buffer 2

op = 3

3 = Rewind

4 = Unload

5 = Write end-of-file

6 = Space forward records (count is in fadd)

7 = Read to end-of-file

8 = Backspace records (count is in fadd)

9 = Backspace file

A = Set density

fadd = 0 200 bpi
= 1 556 bpi
= 2 800 bpi
= 3 1600 bpi

B = Seek

C = Erase (fadd contains number of erasures to be performed)

F = Read status (fadd will contain status upon return)

op = 4

1 = Open buffer 1

2 = Close buffer 1

3 = Open buffer 2

4 = Close buffer 2

busy If the caller has set this bit, it is cleared when request is completed.

IOC Input/output connector (IOC) number

cerr Errors detected by central system before request is sent to peripheral system:

1 = Non-existent input/output connector

2 = Buffer size is greater than 24 small pages or is zero pages for a small page buffer definition request

3 = This file is not on physical mass storage

4 = Illegal tape density requested

5 = Illegal operation or sub-operation

6 = Illegal tpm or mode field

7 = No buffer assigned

8 = File address out of bounds

9 = Illegal attempt to access a read only or write only file

A = Interrupt requested, but no interrupt address specified

B = Request for greater than 128 small pages (for a large page buffer definition request)

C = Buffer crosses a large page boundary (for a large buffer)

D = Read or write access denied for this buffer

E = Attempt was made to open a buffer that was already open

F = Buffer already in use. Previous input/output, which uses the same buffer, not complete.

10 = Attempt to reuse alpha before the previous call, which uses the same alpha address; is complete.

serr Error response returned by the peripheral station. Multiple errors are OR'ed together; values are hexadecimal.

XXXXX1 = Device not ready

XXXXX2 = Tape, SBU, or transmission parity error

XXXXX8 = End of tape

XXXX1X = End of file

XXXX2X = Attempt to write on file protected tape

XXXX4X = Channel failure (disk only)

XXXX8X = Lost data on tape record

XXX1XX = Attempted backspace at load point

XXX2XX = Error in positioning mass storage device (disk only) int Interrupt conditions. This field has meaning only if the interrupt and data base addresses are defined with opcode 5 in the first Beta word pair of the Alpha/Beta word package:

1 = Interrupt on good completion of the request

2 = Interrupt if error on this request

3 = Interrupt on either condition

con Contingency field. Specifies how to handle requests in the Alpha/Beta package.

0 = No contingency. Proceed immediately to the next request.

3 = Contingency. Wait for completion of this request before processing the next.

The caller becomes a candidate for CPU use once the system encounters a data transfer or function request having the con field set to 3 or when no more Beta word pairs exist. Users of the 50 message (explicit input/output) are responsible for detecting data transfer completion by selecting an interrupt condition, checking when the busy bit in the Beta word pair is clear, or issuing a 52 system message to give up the CPU until input/output is complete.

tpm Tape mode:

0 = BCD (7-track tape)

1 = Binary (7- or 9-track tape)

2 = Binary ASCII (7-track tape)

issu Bit set by operating system when central system finds no error, and request is sent to peripheral system

ret Retry field:

0 = Use standard error recovery

1 = No retry on error

2 = Use standard error recovery and system noise records (tapes)

3 = No retry on error and system noise records (tapes)

tape Transmission directions:

0 = Truncate record to 64-bit word bound

1 = Transmit entire record

bits When reading tapes not generated on STAR systems, this field indicates the number of bits read from tape (less than 16), in addition to the byte count returned in fadd. When reading STAR-generated tapes, if this field is non-zero, the byte count in fadd must be decremented by one to indicate the amount of valid data read from tape.

fadd For mass storage, logical page (block) address where data transmission is to begin; for magnetic tape, the number of 16-bit bytes to be transmitted (if zero, transmit entire buffer). When op=3 and sop=A, fadd is used for density selection (see sop=A). When op=3 and sop=6 or 8, fadd must contain the count of records to skip.

blen Length of virtual range, in small pages, to be associated with this buffer (see note 8).

badd Starting virtual page address of buffer where data transfer requests will deposit or obtain information.

Notes:

1. For tape read or write operations; when the file address field (fadd) is zero, the entire image will be transferred to or from tape; otherwise, fadd specifies the number of 16-bit bytes to be transmitted.

2. Following a tape read operation, fadd will contain the number of 16-bit bytes in the physical record. If the record is larger than the buffer, only as much data as fits will be translated.

3. For the backspace and space forward operations, fadd will contain the number of records to be spaced over. When an end of file is encountered, the operation stops and the actual number of records spaced over is returned in fadd.

4. After the read status operation, fadd will contain the unit status. The unit status bits are:

000XX1	Ready
000XX2	Busy
000XX4	Write enable
000XX8	End of file
000X1X	Load point
000X2X	End of tape
000X0X	200 bpi density
000X4X	556 bpi density
000X8X	800 bpi density
000XCX	1600 bpi density
0001XX	Lost data
0002XX	End of operation
0004XX	Parity error
0008XX	Reserved

5. For erase mass storage functions, the right half of the second Beta word is treated as two 16-bit fields. The leftmost 16 bits contain a sector count; the rightmost 16 bits specify a pattern to be written. If the sector count is zero, the entire file is to be erased.

6. For the erase tape function, fadd contains the number of erasures to be performed. Six inches of tape are blanked for each erasure.

7. Interrupt routines may not be interrupted; they are stacked and processed one at a time.

8. Unless the buffer is in a large page, the maximum buffer size is 24 small pages. The maximum large page buffer size is 128 small pages. No large page buffer may cross a large page boundary (for example, each large page buffer must fit within one large page, or 65,536 words).

9. Following a tape read operation, if the record length is not equal to the buffer length, the remainder of the buffer is undefined.

10. When the central system detects an error before a request is sent to the peripheral system, control passes to the error exit address and message processing terminates. For data transfer requests, an error detected by the peripheral system does not cause control to pass to the error exit address. Further action depends on the value in the contingency field (con):

    con = 0 Following Beta requests are processed normally; however, the response code field r is set to 4 and the serr field is filled appropriately.

    con = 3 Following Beta requests are not processed. The response code field r is set to 4 and the serr field is filled appropriately.

11. The interrupt request (op = 5) must be used if an interrupt condition is selected. The interrupt request must be in the first two Beta words. If Beta (2) is non-zero, Beta (2) will be placed in register 1E when control passes to the interrupt address, ia.

12. Only six explicit input/output requests can be processed at one time. If six requests are outstanding when a seventh request is issued, the issuing program is put in a wait state (using the DISCONNECT message) until at least one request is completed.

13. For the density function, the fadd settings are 0, 1, 2, or 3 for 200 bpi, 556 bpi, 800 bpi, and 1600 bpi, respectively.

14. A buffer may be closed as soon as the data transfer request is issued; input/output need not be completed.

15. The Beta index and the Alpha bit address causing the interrupt are stored in register 3 by the operating system when the interrupt occurs. The zero level program counter is stored in register 4.

16. The Alpha and Beta words should not be modified until all input/output described by the call is complete.

17. Following a mass storage read or write operation, the fadd field will contain the actual number of small pages transmitted.

18. The end-of-file bit in the serr field will be set if, for a mass storage read or write operation, the buffer extends past the last word in the file.

19. If the serr field is set to non-zero for any of the conditions described, it will cause an interrupt if the interrupt-on-error bit in the mode field has been set.

20. When a file is closed, all buffers defined for that IOC are closed.

21. Error codes appropriate for certain operations are shown below:

Mass Storage I/O

		cerr													
		1	2	3	4	5	6	7	8	9	A	B	C	D	E
op	1	X	X			X	X	X	X	X	X	X	X	X	
	2	X	X			X	X	X	X	X	X	X	X	X	
	3	X				X	X			X	X				
	4	X		X		X									X

The serr bits for parity error, end of file, channel failure, and positioning mass storage are appropriate for mass storage I/O.

Magnetic Tape I/O

		cerr													
		1	2	3	4	5	6	7	8	9	A	B	C	D	E
op	1	X	X			X	X	X			X	X	X	X	
	2	X	X			X	X	X			X	X	X	X	
	3	X			X	X	X				X				
	4	X				X									X

For all tape operations except status, the serr bits for tape not ready and parity error are appropriate. For read/write tape, data exceeds buffer, end of tape, end of file (read only), and file protected (write only) serr bits may be set. For write end of file and erase, the end of tape and file protected,serr bits may be set. For forward space record and forward space file, the end of tape and end of file serr bits may be set. For backspace record and backspace file, the end of file and load point serr bits may be set.

GIVE UP CPU UNTIL I/O COMPLETES (f = 0052)

This message allows the program to give up the central processor unit until all or part of its input/output is complete. It is used in conjunction with the EXPLICIT I/O message (f = 0050).

Message format:

Alpha (1)	r 16	len 16	unused 16	0052 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		
Beta (1)	vadd 64			

r Response code returned by the operating system when this message has been processed. If no error occurs, a response code value of zero is returned; otherwise:

1 = vadd does not contain the address of an EXPLICIT I/O message (f = 0050).

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

215 = No error exit address when error occurred.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

n Options for this message:

0 = Give up the central processor until all I/O is complete. No Beta words are required for this option.

1 = Give up the central processor until I/O calls specified by the Beta portion of the message are complete.

eea Error address.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

vadd If the central processor has been given up until part of the I/O is complete (n = 1), the Beta portion of the message contains the virtual bit addresses of the EXPLICIT I/O messages comprising that part of the input/output (the address of Alpha (1) for those messages).

Notes:

1. The operating system keeps a record only of outstanding I/O messages. If the address in Beta does not match an outstanding I/O message, the message is presumed to be completed.

2. If an interrupt routine gives up the central processor until all I/O completes (n = 0), the routine will be restarted when all I/O has completed; interrupts are stacked and processed after the routine issues a RETURN FROM INTERRUPT message (f = 0051). If a program at level zero issues the same call, it will be restarted when all I/O is complete, and all interrupts have been processed.

3. If an interrupt routine gives up the central processor until all specified I/O completes (n = 1), the routine will be restarted when all specified I/O completes; interrupts are stacked and processed after the routine issues a RETURN FROM INTERRUPT message (f = 0051). If a program at level zero issues the same call, it will be restarted when all specified I/O completes and all interrupts have been processed.

PROGRAM INTERRUPT (f = 001C)

With this message, a program informs the operating system that it should, or should not, be interrupted by a message.

Message format:

<table>
<tr><td>Alpha (1)</td><td colspan="2">r 16</td><td>len 16</td><td>unused 16</td><td>001C 16</td></tr>
<tr><td>Alpha (2)</td><td>j 8</td><td>b 8</td><td colspan="3">eea 48</td></tr>
<tr><td>Alpha (3)</td><td colspan="2">Bl 16</td><td colspan="3">Ba 48</td></tr>
</table>

Beta (1)	unused 16	ia 48
Beta (2)	dbl 16	dba 48

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero. Otherwise:

1 = Value or program interrupt address is greater than virtual address range

2 = Program selected an illegal interrupt option

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

j Type of interrupt option:

00 = b field refers to messages from a controller

b Interrupt option:

00 = Any message will interrupt this program.

01 = Any message from a terminal preceded by a left justified (sc)I will interrupt this program.

02 = Inhibits interrupts from messages.

eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea = 0, the error will be considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

ia Virtual bit address to where control transfers upon occurrence of selected interrupt

dbl Length of the data base to be established if a selected interrupt occurs

dba Address of the data base to be established if a selected interrupt occurs

Notes:

1. When this message is issued for options b = 00 or b = 01, this program will be interrupted by all subsequent messages and interrupts arising from a terminal until this program either terminates or issues this message with b = 02.

2 Whenever control passes to the program interrupt address, a message is always waiting.

3. To release the program interrupt, it is necessary to issue a RETURN FROM INTERRUPT message (f = 0051).

4. For b = 01, the (sc)I preceding the message will be stripped; and the message will be repositioned at the beginning of the word.

5. The (sc)I interrupt causes any output messages to be released.

6. (sc)I preceding a message will interrupt the highest level controller issuing this message with b = 01.

RETURN FROM INTERRUPT (f = 0051)

This message is issued by a program at the conclusion of an interrupt routine for either an input/output or program message interrupt.

Message format:

Alpha (1)	r 16	unused 16	unused 16	0051 16
Alpha (2)	n 16	eea 48		

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero. If r = 1, this message was issued by other than an interrupt routine.

n Options for this function:

0 = The current interrupt is released and the next interrupt is taken. If no further interrupts exist, control returns to the interrupted program at the point of interruption.

1 = The current interrupt is released and control returns to the point following this message if no error occurred. If no other interrupts are stacked, control returns immediately. If interrupts are stacked, control returns after the RETURN FROM INTERRUPT has been issued for the last interrupt in the list.

2 = Release the current interrupt and take the next interrupt on the list, if one exists. If no other interrupts exist, return to the originally interrupted program at the address in register 4. The registers of the originally interrupted program are preserved.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

MISCELLANEOUS FILE MANAGEMENT

An additional six messages are concerned with file control: REDUCE FILE LENGTH, DESTROY FILE, GIVE FILE, ROUTE FILE, GIVE TAPE ACCESS TO CONTROLLEE, and CHANGE FILE NAME OR ACCOUNT.

When a mass storage file is inactive (no input/output connectors exist between this file and any program), the user may issue a REDUCE FILE LENGTH message to reduce the length of a mass storage file. The released space is then available for reassignment. (This message modifies the LENGTH parameter specified when the file was created.)

DESTROY FILE indicates to a program that previously used mass storage space is currently available for reassignment, or that a particular tape drive is available for reassignment. At the conclusion of DESTROY FILE message processing, any mass storage file referenced by the message has ceased to exist, as have any modified pages of the file. Virtual address definitions pertaining to this file are no longer defined, and the input/output connection and map entries are erased. A file need not be closed before the DESTROY FILE message is issued.

A user may deposit a file with another user by issuing a GIVE FILE message. The recipient user is allowed to perform actions on the file so named. A common use of this message is to hand over files to an output processor for later release to an appropriate device.

ROUTE FILE allows the user to specify the destination of a file and the manner in which it is disposed. The message frees the user from the burden of using naming conventions to accomplish routing and disposition of a file.

GIVE TAPE ACCESS TO CONTROLLEE allows one program to enable another program to access certain tape drives through the first program's input/output connectors. The controllee establishes its own input/output connector for the action but is limited by the controllee in how it may affect information on the tape.

CHANGE FILE NAME OR ACCOUNT allows a user to change the name or account number of a given file.

REDUCE FILE LENGTH (f = 000A)

A program issues this message to reduce the length of an existing private mass storage file. Reduction occurs at the largest absolute address end of the file.

Message format:

<table>
<tr><td>Alpha (1)</td><td>r
16</td><td>len
16</td><td>unused
16</td><td>000A
16</td></tr>
<tr><td>Alpha (2)</td><td>unused
16</td><td colspan="3">eea
48</td></tr>
<tr><td>Alpha (3)</td><td>Bl
16</td><td colspan="3">Ba
48</td></tr>
</table>

<table>
<tr><td>Beta (1)</td><td colspan="3">name
64</td></tr>
<tr><td>Beta (2)</td><td>ss
8</td><td>length
24</td><td>unused
32</td></tr>
</table>

r — Response code returned by the operating system when this message has been processed. If no error occurs, the response code will be zero. If r = 1, an error response code was returned in an ss field of Beta. Otherwise,

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len — If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion of the message.

eea — Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea = 0, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

name — Name of file in ASCII; file names must be of proper format as described in section 4.

ss Error response field:

1 = New length specified for file is greater than existing length.

2 = File name cannot be found in file index.

3 = File is still open to an active program.

length New file length in small page blocks

Note: Before a file can be reduced in length, it must not be active for this or any other program in the system.

DESTROY FILE (f = 0002)

This message is issued by a program to sever its connection with a tape drive and release the drive for re-assignment, or to sever its connection with a mass storage file and release the mass storage space.

Message format:

<table>
<tr><td>Alpha (1)</td><td>r
16</td><td>len
16</td><td>unused
16</td><td>0002
16</td></tr>
<tr><td>Alpha (2)</td><td>n
16</td><td colspan="3">eea
48</td></tr>
<tr><td>Alpha (3)</td><td>Bl
16</td><td colspan="3">Ba
48</td></tr>
</table>

<table>
<tr><td>Beta (1)</td><td colspan="6">name
64</td></tr>
<tr><td>Beta (2)</td><td>IOC
8</td><td>dev
8</td><td>saddr
24</td><td>unit
8</td><td>own
8</td><td>ss
8</td></tr>
</table>

r — Response code returned by the operating system when this message has been processed. If no error occurs, the response code will be zero; otherwise:

1 = Error code was returned in an ss field of Beta.

211 = Number of destroys in this message is illegal (n = 0 or n > 16)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

n — Number of requests in this message (maximum = 16)

eea — Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

name Name of file in ASCII; file names must be in proper format as described in section 4.

IOC For a tape destroy function, the program must specify the input/output connector (IOC) number. If a mass storage file is being destroyed, the operating system will return, in this field, the inclusive OR of all input/output connector numbers connected to this file.

dev Device:

0 = Mass storage file

4 = Tape drive

Fields saddr, unit, and own are inspected by the system only when the call is made by a privileged user.

saddr Beginning sector address of the file to be destroyed. This field is checked when unit is non-zero.

unit Logical unit number of file to be destroyed.

own Ownership of file to be destroyed.

0 = Destroy file of private ownership.

1 = Destroy file of public ownership; valid only for privileged users.

2 = Destroy file of pool ownership; valid only for pool boss.

ss Error response field:

0 = Normal completion.

1 = File name does not exist.

2 = File name given is in conflict with that in the input/output connector.

3 = Another active program has the file open.

4 = Parameter or format error.

5 = Non-privileged task tried to destroy a public file.

6 = User other than pool boss tried to destroy a pool file.

7 = Illegal IOC number specified.

8 = Drop file map full.

Note: If a mass storage file is at a sufficiently high security level, it will be overwritten with a pattern when it is destroyed. Some installations can choose to overwrite all files when they are destroyed.

GIVE FILE (f = 0008)

This message is issued when a program gives one or more of its private, inactive files to another user.

Message format:

Alpha (1)	r 16	len 16	c 16	0008 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	name 64		
Beta (2)	ss 8	unused 8	auser 48
Beta (3)	pool name 64		

r response code returned by the operating system when this message has been processed. If no error occurs, the response code will be zero; otherwise:

1 = Refer to Beta field ss for the specific error

211 = Number of files given was zero or greater than 16

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

c Use:

0 = Files are to be given to the private user number shown in Beta (2).

1 = Files are to be given to a pool named in Beta (3).

n Number of files to be given (maximum of 16) eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

name Name of file in ASCII; file names must be in proper format as described in section 4.

ss Error response code

0 = No error; normal completion

1 = File of this same name already exists at destination of this file.

2 = File has the same name as a public file.

3 = No such file exists.

4 = No such user number exists.

5 = Output file is improperly named.

6 = File to be given is still active.

7 = User number specified is that of the public list.

8 = File to be given is a source or drop file.

9 = Recipient of the file has a security classification less than that of the file.

A = Pool name specified does not exist, or giver is not a member of this pool.

auser ASCII user number to receive file; not used when c = 1.

pool name Name of pool, left justified and blank filled; used only when c = 1.

Notes:

1. No file may be given to the public list. No file having the same name as a public file may be given.

2. Files given to user 999999 for output processing must be properly named. Output processing routines are run only on a demand basis. Running a system output processor is the responsibility of the routine that processes this message. Processing for family files will be deferred until the end name is recognized; other files will be processed at once.

3. The user number of the donor will be preserved in the file index table only for files given to user 999999 for output processing.

ROUTE AND FILE DISPOSITION (f = 000D)

This message is issued by a program to set the routine destination and type disposition of an existing file.

Message format:

Alpha (1)	r 16	len 16	unused 16	000D 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	name 64			
Beta (2)	flags 40	def 8	el 8	ss 8
Beta (3)	fc 16	ec 16	ic 16	dc 16
Beta (4)	tid 16	sid 24	8	pri 16
Beta (5)	fid 64			
Beta (6)	unused 8	un 56		

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1 = Error code was returned in ss field of Beta

211 = Number of files routed by this message was illegal (n = 0 or n greater than 16)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

n Number of files to be routed by this call (maximum is 16).

eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the first full word of Beta.

name File name in ASCII. File names must be of the proper format as described in section 4.

flags Flag bits. Bit = 1 indicates the specified parameter is to be processed. Bit = 0 indicates the specified parameters is to be ignored.

Beta (2) bits from left to right starting from 0:

BIT 0	fc	BIT 5	sid
BIT 1	ec	BIT 6	pri
BIT 2	ic	BIT 7	fid
BIT 3	dc	BIT 8	un
BIT 4	tid		

def Indicates file disposition is to be deferred. The system stores the information about the file and disposes it as requested when the file is released. Files are released at a GIVE FILE to user 1, CLOSE file, end-of-task, or by a ROUTE that specifies immediate release (def = 0). Routing of files to the input queue cannot be deferred. With deferred routing, the user can redefine the same file with subsequent ROUTE calls.

el Beta entry length; must be at least 2 and less than 7.

ss Error response field:

01 Immediate release (def = 0) of an active file

02 Immediate release (def = 0) of a non-allocated file

03 Beta entry (el) length error

04 File must exist before route (temporary)

05 Immediate release (def = 0) with no disposition set

06 Could not write PFI

07 Illegal disposition code

08 Deferred routing (def = 0) specified for an input file

09 Illegal file name

fc Forms code; a two character mnemonic. This parameter indicates special card or paper forms are to be used for output. This field is defined but not currently supported.

ec External characteristic. A two-character mnemonic giving the print or punch representation of the file.

26 Punch in 026 keypunch format

29 Punch in 029 keypunch format

*B STAR binary format

80 Punch 80 column binary format

ic Internal characteristic. A two-character mnemonic indicating the format of the file.

AS Format is 8-bit ASCII. If the file has dc=PR, the file has ANSI carriage control characters.

BI Format is binary

PA Format is 8-bit ASCII with ASCII control characters.

dc Disposition code. A two-character mnemonic indicating how the file is to be disposed.

SC Scratch; destroyed at end of task

PR Print on any available printer

PU Punch

IN Input for Batch processing

sid Site identifier. A three-character identifier for the front-end processor. This field is defined, but not currently supported.

tid Terminal identifier. A two-character mnemonic identifying a terminal. This field is defined, but not currently supported.

pri Priority level for a file to be output at its originating terminal or front-end processor. This field is defined, but not currently supported.

fid File name in ASCII, to be used in the output queue. This is ignored if deferred routing is specified. If fid = 0, the file name, while the file is in the output queue, is the same as the job name (default).

GIVE TAPE ACCESS TO CONTROLLEE (f = 000C)

With this message, a program gives its controllee access to one or more magnetic tapes currently existing through the controller's input/output connector area.

Message format:

Alpha (1)	r 16	len 16	c 16	000C 16
Alpha (2)	n 16	eea 48		
Alpha (3)	BI 16	Ba 48		

Beta (1)	ss 8	IOC 8	unused 8	tname 40

r — Response code returned by the operating system when this message has been processed. If no error occurs, the response code will be zero; otherwise:

1 = Error code was returned in ss field of Beta

211 = Number of tapes given by this message was illegal (n = 0 or n > 16)

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

c — Controls tape access:

0 = Give tape access

1 = Recall tape access

n — Number of tapes to be given by this call (maximum is 16).

eea — Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

ss — Error response field:

00 = Normal completion

01 = No tape name specified

02 = Wrong input/output connector indicated or IOC is not in range 0–15.

03 = Controllee already owns a private tape with specified input/output connector number.

04 = No controllee exists.

06 = Controller does not own a tape by specified name.

07 = Controllee input/output connector is already in use.

IOC Input/output connector number for this tape.

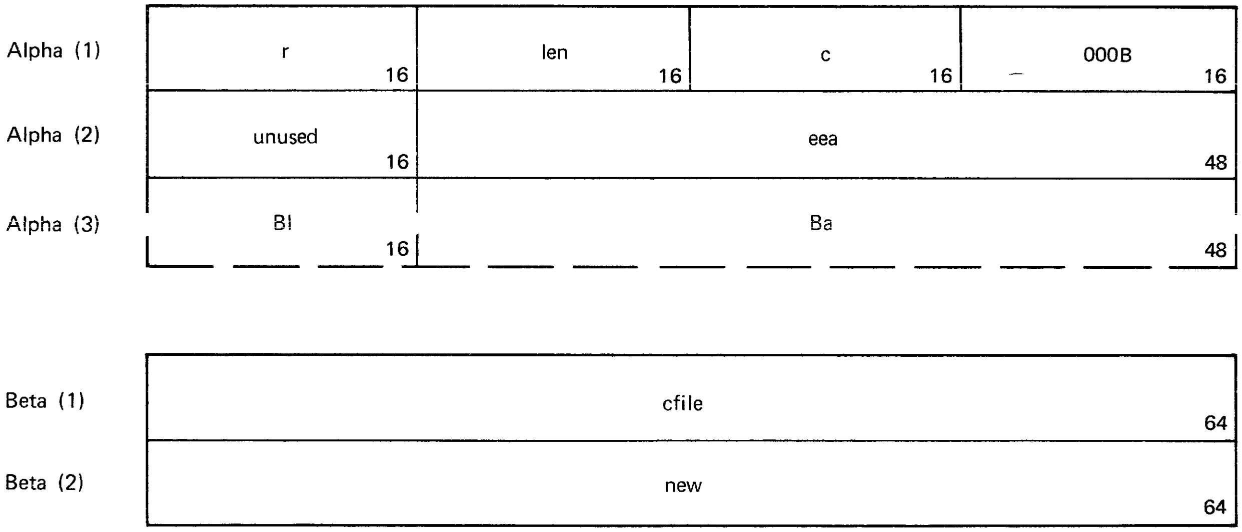
tname ASCII name of this tape; maximum is five characters, left justified and blank filled.

Note: The controllee gains access to the tape through the same input/output connector number as the controller uses. The controllee's input/output connector does not, however, have the tape name. This prevents the controllee from destroying the tape.

CHANGE FILE NAME OR ACCOUNT (f = 000B)

This message allows a program to change the name or account number of an existing private file.

Message format:



Alpha (1)	r 16	len 16	c 16	000B 16
Alpha (2)	unused 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	cfile 64
Beta (2)	new 64

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code will be zero; otherwise:

1	=	Current file name is still active
2	=	Current file name does not exist
3	=	New file already exists
4	=	New account number is not valid
5	=	New file name is invalid
214	=	Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small.

len — If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

c — Type of change:

00 = Change file name

01 = Change account number

eea — Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

cfile — Current file name in ASCII, left-justified and blank filled.

new — Contains the new file name in ASCII, left-justified and blank filled. For an account number, new contains the new account number in ASCII.

OTHER SYSTEM MESSAGES 5

When programs operating in controller-controllee mode need to communicate with each other, standard messages are used to transmit certain information between programs.

SEND A MESSAGE TO CONTROLLEE (f = 0015)

A program may start its controllee with or without a message, but the controllee must have been previously initialized.

Message format:

<table>
<tr><td>Alpha (1)</td><td colspan="2">r
16</td><td colspan="2">len
16</td><td>m
8</td><td>c
8</td><td colspan="2">0015
16</td></tr>
<tr><td>Alpha (2)</td><td>unused
8</td><td>b
8</td><td colspan="6">eea
48</td></tr>
<tr><td>Alpha (3)</td><td colspan="2">Bl
16</td><td colspan="6">Ba
48</td></tr>
<tr><td>Beta (1)</td><td colspan="8">message</td></tr>
</table>

message maximum length is 4096 characters.

r Response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero; otherwise:

1 = Length of Beta in character bytes is either zero or greater than 4096.

2 = Illegal option selected for this message.

3 = No controllee matches the value of b (for b ≠ 0).

4 = For b = 0, no controllee exists.

7 = Error exit has been taken, since controllee already has a text string from controller.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length in character bytes and virtual address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

m Options for sending message:

00 = Start controllee program with the message. If controllee already has a message from controller, replace it with message in Beta.

01 = Start controllee program with the message. If controllee already has a message from controller, return control to the error exit address.

02 = Start controllee without a message (Beta portion of message is not required for this option.)

c Control field:

00 = Stop running this program and start the controllee program immediately.

01 = Remove this program from main memory before starting the controllee program.

b Descriptor number of controllee; if zero, the message will be sent to the next lower controllee.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length and virtual bit address of the Beta portion's first full word.

message Maximum message length 4096 characters.

Notes:

1. Receiving a message causes the operating system to copy the message from Beta into a system buffer and to start the controllee.

2. If a controllee is running, a message from a terminal to its controller will stop the controllee and start the controller.

3. If any controllee other than the immediate controllee of a job control processor issues GET MESSAGE FROM CONTROLLER (f = 0016) and no controller message is waiting, the controllee will stop running, and will be put in a state of waiting for a controller message. The next higher level controller will be started.

GET A MESSAGE FROM CONTROLLEE (f = 0017)

This message may be used to obtain a message from a controllee program. Depending on the option selected, the message may be simply a copy of the message sent or the message sent may be processed into a set of symbols.

Message format:

Alpha (1)	r 16	len 16	m 8	c 8	0017 16
Alpha (2)	j 8	b 8	eea 48		
Alpha (3)	Bl 16	Ba 48			

(Beta formats are discussed under the m format option.)

r Response code returned by the operating system when this message has been processed. If no error occurs, control proceeds normally and this field contains the number of character bytes (if a message was obtained) or words (if symbols were obtained) returned in Beta. If control returns to the error exit address, nothing is returned in Beta and the significance of r is as follows:

1 = Count of items to be returned was either zero or greater than 4096.

2 = Illegal option was specified for this message.

3 = No message from controllee was waiting.

4 = Message from controller was waiting.

6 = This program started because the controllee whose level and descriptor number is stored in j and b is waiting for a message from controller.

7 = More than 200 delimiters defined by this program.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length in words (if symbols are to be obtained) or character bytes (if a message is to be obtained) and virtual address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion in words (if symbols are to be obtained) or character bytes (if a message is to be obtained).

m Message format options:

For all m options , except m=00, fill and justification may be specified by setting bits as follows:

XXXX1XXX	=	null fill
XXXX0XXX	=	blank fill
XX1XXXXX	=	right justified
XX0XXXXX	=	left justified

00 Message is copied from system buffer to Beta. No end-of-message character byte is added to the text. If character bytes in the message exceed the number requested, only the number of character bytes requested will be copied to Beta. If they are less than the number requested, the entire message will be copied to Beta and the remainder of Beta will be cleared.

Beta (1)	message

Maximum message length is 4096 characters.

01 (XXX0X001) Message is processed into symbols; delimiters are defined by the program and their number must not exceed 200. Symbols are stored in Beta, one symbol per word.

Beta (1)	nd 16	vadb 48
Beta (2)	symbol	

nd Number of delimiters

vadb Virtual bit address of delimiter buffer. Delimiters are stored left to right, character byte by character byte in the buffer.

02 (XXX0X010) The message is processed into symbols; delimiters are period, comma, slash, equal, plus, minus, and left and right parentheses. Symbols are stored in Beta, one per word.

Beta (1)	symbol

03 (XXX0X011) Message is processed into symbols. Delimiters are defined as installation parameter options.

Beta (1)	symbol

c Control field indicating whether the message space in the system buffer is released:

00 = Process and return the message to Beta; release the system buffer space currently occupied by the message.

02 = Process and return the message to Beta, but do not release the system buffer space currently occupied by the message.

j Operating system supplies the level of controllee that sent message.

b Operating system supplies descriptor number of controllee that sent message.

eea Virtual bit address to receive control if an error occurs during message processing. If eea = 0, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in words (if symbols are to be obtained) or character bytes (if a message is to be obtained) and virtual bit address of the Beta portion's first full word.

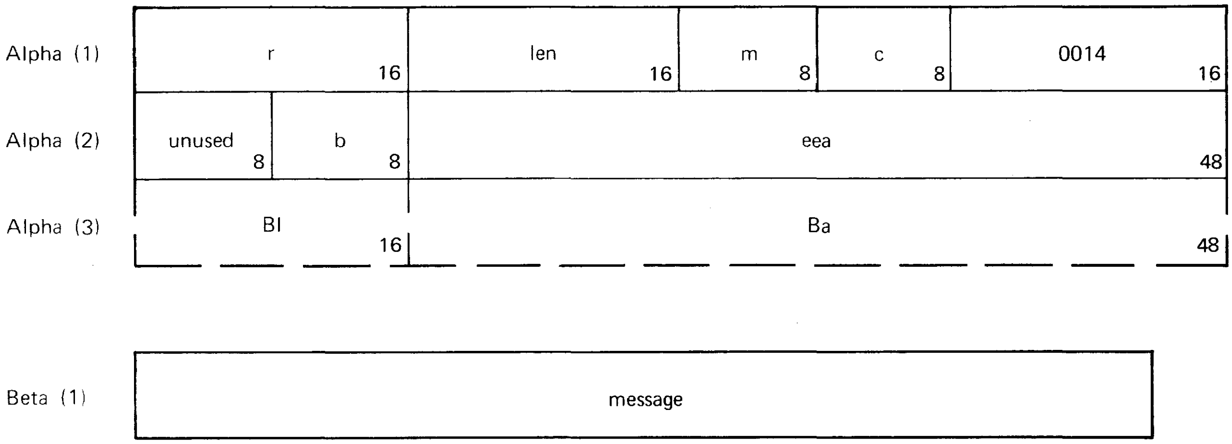
Notes:

1. Multi-word symbols are permitted and processed without any special treatment.
2. If the number of symbols exceeds the number requested, only the number of symbols requested are stored in Beta. If they are less than the number requested, all symbols are stored in Beta; the operating system does not add an end-of-message character.
3. Delimiters are always returned right justified with null fill. Blanks are never treated as a special case (i.e., if a space is a delimiter, all occurrences of blank will result in a delimiter being returned. If space is not a delimiter, then spaces are processed the same as any other character).
4. The level and descriptor number supplied by the operating system have no meaning if the controllee that sent the message has been disconnected.

SEND A MESSAGE TO CONTROLLER (f = 0014)

This message may be used by a program to send a message to a program controller or the job control processor.

Message format:



r Response code returned by operating system when this message has been processed. If no error occurs, the response code will be zero. Other values are as follows:

1 = Length of Beta in character bytes is either zero or greater than 4096.

2 = Illegal option selected for this message.

3 = No controller matches the value of b ≠ 0, or this task is already at level 1; therefore, sending a message would be pointless.

4 = If the notify option was selected (m = 01), the controller designated was a job control processor for a logged out terminal.

5 = If the notify option was selected, the controller designated was a job control processor for a terminal that has been logged in under a different suffix.

6 = If the notify option was selected, the system output buffer is full.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length in character bytes and the virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length in character bytes of the Beta portion.

m Options available if messages cannot be sent to controller.

00 = If message was for a logged out terminal, replace any existing message. If the job control processor buffer is full, stop running this program until buffer is free.

01 = If message cannot be sent to controller, return control to error exit address.

02 = If message cannot be sent to controller, stop running this program until message can be sent.

c Control field:

00 = Send message to controller. For a program controller, start running the program controller and stop running this program. For a terminal controller, continue to run this program.

01 = Send message to controller. For a program controller, remove this program from main memory before starting controller. For a terminal controller, continue to run this program.

02 = Send message to job control processor; for a terminal controller, continue to run this program; for a batch processor controller, stop this program.

b Descriptor number of the controller. If the message is to be sent directly to the job control processor (c = 02), or if this program's controller is a job control processor, this field is ignored. If this field is zero, the message is sent to the next higher controller.

eea Virtual bit address to receive control if an error occurs during message processing (r $\neq$ 0). If eea is zero, the error is considered fatal.

Bl If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these
Ba parameters indicate the length in character bytes and virtual bit address of Beta portion's first full word.

Notes:

1. When a message is received, the operating system copies it from Beta into a system buffer.

2. When output messages go to a job control processor for a logged in terminal and the wait or replace option is used, the system buffer will hold up to five messages or 4096 character bytes, whichever occurs first. For a logged out terminal, only one message can be held in the buffer.

3. Output messages for a job control processor initiated for a terminal are grouped in blocks of 151 character bytes and sent one block at a time to the terminal. If the last block is less than 151 character bytes, an end-of-message character is added after the last message character byte. The originator is responsible for any formatting of multi-line messages by inserting the line feed and carriage return characters.

GET A MESSAGE FROM CONTROLLER (f = 0016)

This message is used to obtain a message from a controller program. Depending on the option selected, may be simply a copy of the message sent or it may be processed into a set of symbols.

Message format:

Alpha (1)	r 16	len 16	m 8	c 8	0016 16
Alpha (2)	j 8	b 8	eea 48		
Alpha (3)	Bl 16	Ba 48			

(Beta formats are discussed under the m format option.)

r Response code returned by operating system when this message has been processed. If no error occurs, control proceeds normally; and this field contains the number of character bytes (if a message is obtained) or words (if symbols are obtained) returned in Beta. If control returns to the error exit address, nothing was returned in Beta; and the significance of r is as follows:

1 = Count of items to be returned was either zero or greater than 4096.

2 = Illegal option specified for this message.

3 = Error exit has been taken, since no controller message existed.

7 = More than 200 delimiters defined by this program exceeded.

9 = Get message issued from a level 1 task.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length in words (if symbols are to be obtained) or character bytes (if a message is to be obtained) and the virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion in words (if symbols are to be obtained) or character bytes (if a message string is to be obtained).

m Message format option:

For all m options, except m=00, fill and justification may be specified by setting bits as follows:

XXXX1XXX = null fill
XXXX0XXX = blank fill
XX1XXXXX = right justified
XX0XXXXX = left justified

00 Message is copied from the system buffer to Beta; no end-of-message character byte is added to the text. If the number of character bytes in the message exceeds the number requested, only the number of character bytes requested will be copied to Beta. If they are less than the number requested, the entire message is copied to Beta; and the remaining portion of Beta is cleared.

Beta (1)	message

Maximum message length is 4096 characters.

01
(XXX0X001) Message is processed into symbols; delimiters are defined by the program, and their number must not exceed 200. Symbols are stored in Beta, one symbol per word.

Beta (1)	nd 16	vadb
Beta (2)	symbol	

nd Number of delimiters

vadb Virtual bit address of delimiter buffer. Delimiters are stored left to right, character byte by character byte in the buffer.

02
(XXX0X010) Message is processed into symbols; delimiters are period, comma, slash, equal, plus, minus, and left and right parentheses. Symbols are stored in Beta, one symbol per word.

Beta (1)	symbol

03
(XXX0X011) Message is processed into symbols. Delimiters are defined as installation parameter options.

Beta (1)	symbol

c Control field:

00 = If no controller message is present, stop running this program until a message arrives. If a controller message is present, it resides in a system buffer. Process and return the message to Beta, and release the system buffer space occupied by the message.

01 = If no controller message is present, return control to the error exit address. If a message is present, it resides in a system buffer. Process and return the message to Beta and release the system buffer space occupied by the message.

02 = If no controller message is present, stop running this program until a message arrives. If a controller message is present, it resides in a system buffer. Process and return the message to Beta, but do not release the system buffer space occupied by the message.

03 = If no controller message is present, return control to the error exit address. If a controller message is present, it resides in a system buffer. Process and return the message to Beta, but do not release the system buffer space occupied by the message.

j Operating system supplies the level of the controller that sent the message.

b Operating system supplies the descriptor number of the controller that sent the message.

eea Virtual bit address to receive control if an error occurs during the message of processing. If the value of eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), this parameter indicates the length in words (if symbols are to be obtained) or character bytes (if a message is to be obtained) and virtual bit address of the Beta portion's first full word.

Notes:

1. Multi-word symbols are permitted and processed without any special treatment.

2. If the number of symbols exceeds the number requested, only the number of symbols requested will be stored in Beta. If they are less than the amount requested, all symbols are stored in Beta; the operating system will not add an end-of-message character.

3. Delimiters are always returned right justified with null fill. Blanks are never treated as a special case (i.e., if a space is a delimiter, all occurrences of blank will result in a delimiter being returned. If space is not a delimiter, then spaces are processed the same as any other character).

4. If the message arises from a job control processor, the operating system will set j = 1, and, if the job processor is a terminal, b = FF.

5. If no message is present, fields j and b will contain the level and descriptor number of the immediate controller.

INITIALIZE OR DISCONNECT CONTROLLEE (f = 001B)

This message is used by a program to initialize another program as a controllee. It also may be used to disconnect a previously initialized controllee. Up to five levels of program controllees are permitted.

Message format:

Alpha (1)	r 16	len 16	unused 16	001B 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	filename 64		
Beta (2)	b 8	unused 8	tl 48

r Response code returned by operating system when this message has been processed. If no error occurs, the response code is zero. Other response codes for initializing a controllee program ($n \neq 10$) are as follows:

1 = Controllee program is already present

2 = Illegal option

3 = Controllee program file was not found

4 = Insufficient time to run the controllee program

5 = Illegal priority

6 = Error in creating system drop file

7 = Controllee program file is not executable

8 = Mass storage device error

9 = Full system tables inhibit initialization of controllee program at this time.

10 = Drop file cannot be verified

11 = Disk READ MESSAGE full

12 = BAD minus page in controllee file

13 = Undefined error in drop file verification

A = Abnormality in the controllee program file or drop file I/O connector entry.

B = Five levels of controllee program are present already.

C = No controllee present (for disconnect only). If a controllee program is being disconnected (m=0), this response code indicates no controllee program is present.

D = Controllee program drop file is too small.

E = Unable to destroy existing drop file.

F = Unable to restart controllee because interrupt register table is full.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion of the message.

n Options for this message:

0 = Initialize controllee program and restart this program.

1 = Initialize controllee program and immediately begin executing. Stop executing this program.

10 = Disconnect controllee program and restart this program. (Beta portion of message is not required for this option.)

eea Virtual bit address to receive control if error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

filename Name of controllee program (must be an executable file), left justified and blank filled.

b When the message is issued, the operating system returns the controllee program's descriptor number. If the controllee program is disconnected and re-initialized, this number may change.

tl Time limit for the controllee program, in microseconds. When this time limit is zero, the controller's time limit is used.

INITIALIZE CONTROLLEE CHAIN (f = 001D)

This message is used by a program to initialize a chain of controllees. Up to five levels of controllee programs are permitted. Control is returned to the program after the call.

Message format:

Alpha(1)	r 16	len 16	unused 16	001D 16
Alpha(2)	n	eea		
Alpha(3)	Bl	Ba		

Beta(1)	s 8	t 8	c 8	unused		ss 8	m 8	d 8
Beta(2)	k 8	unused 8	tl 48					
Beta(3)	source 64							
Beta(4)	drop 64							

r Response code returned by the operating system when this message has been processed. If no errors occur, the response code is zero. Other values are as follows:

1 = Controllee program already present

2 = Full system tables inhibit initialization of controllee program at this time

3 = Controllee levels exceeded five

4 = Error in attempt to initialize controllee; see ss field

len If len = FFFF, Alpha(3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha(3), and len is the length of the Beta portion of the message.

n Number of parameter sets less than or equal to 4.

eea Virtual bit address to receive control if an error occurs during message processing (r≠0). If eea=0, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portions first full word.

s Absolute level value (2-5) of the program named in the third Beta word of the entry is returned to this field by the system. Beta words appear in the order in which the controllees are to be initialized.

t Descriptor number of program named in the third Beta word of this entry is placed in this field.

c Descriptor number of this program's controller is placed here.

ss File initialization error

- 3 Controllee program file was not found
- 4 Insufficient time to run the controllee program
- 6 Error in creating dropfile
- 7 Controllee program file is not executable
- 8 Mass storage error
- 9 Abnormality in the controllee program file or dropfile I/O connector entry

m Task level, relative to controllee specified in Beta(5), to receive messages from above.

d Task level, relative to controllee specified in Beta(3) to receive message from below.

k Descriptor number of this programs controllee is in this field. This number may be zero.

source ASCII name, left justified and blank filled, of the executable source file to be initialized if dropfile name field is zero. Returned by system if dropfile name field is other than zero.

drop ASCII name, left justified and blank filled, of the dropfile. If specified, the task is started from this dropfile.

tl Time limit in microseconds, for the controllee program. When controllee's time is exhausted, tl is zero.

Note: Any error in the request causes the entire chain to be ignored, and none of the controllees will be initialized.

MESSAGE CONTROL (f = 0018)

This message informs the operating system that messages directed to this program should be sent to another controllee or controller in the chain.

<table>
<tr><td>Alpha (1)</td><td colspan="2">r
16</td><td colspan="2">unused
16</td><td>m
8</td><td>c
8</td><td colspan="2">0018
16</td></tr>
<tr><td>Alpha (2)</td><td>j
8</td><td>b
8</td><td colspan="6">eea
48</td></tr>
</table>

r A response code returned by the operating system when this message has been processed. If no error occurs, the response code is zero. If r = 1, no controller or controllee has that descriptor number.

m Options for bypassing input and output messages:

01 = Manipulate input bypass according to control switch c.

02 = Manipulate the output bypass according to control switch c.

03 = Manipulate both input and output bypass according to control switch c.

04 = Determine message destination from controllee descriptor number in j.

05 = Determine message destination from controller descriptor number in b.

06 = Examine both b and j to determine message destination.

c Bypass control switch:

00 = Turn off bypass

01 = Turn on bypass

j Descriptor number of destination controllee.

b Descriptor number of destination controller.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Notes:

1. The controllee may direct messages to a job control processor by setting b = FF.

2. Messages sent to this program specifically by descriptor number are not redirected.

3. If the input bypass is set, controller messages not specifically directed to this program are sent to this program's controllee.

4. If the output bypass is set, controllee messages not specifically directed to this program are sent to this program's controller.

REMOVE CONTROLLEE FROM MAIN MEMORY (f = 0019)

This message may be used to swap a controllee program or the user program from main memory to mass storage.

Message format:

Alpha (1)	r 16	len 16	unused	0019 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		
Beta (1)	b 8	56		

r Response code returned by operating system when this message has been processed. If no error occurs, the response code is zero.

1 = No controllee exists of this descriptor number

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or len is zero.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

n Specifies the controllee to be removed

00 = Remove the next lower controllee program to mass storage. No Beta portion is required for this option.

01 = Remove the controllee specified in Beta (1) to mass storage.

02 = Remove this program to mass storage.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

b Descriptor number of controllee to be removed to mass storage.

Note: The controller program stops running and is put in a write controllee state until all controllee pages are written to mass storage.

TERMINATE (f = 0006)

A program issues a TERMINATE message to signal the operating system that it has completed execution. All lower level controllees are also terminated. The message consists of an Alpha portion only.

Message format:

<table>
<tr><td>Alpha (1)</td><td colspan="2">unused 16</td><td>0000 16</td><td>c 16</td><td>0006 16</td></tr>
<tr><td>Alpha (2)</td><td>rc 8</td><td>unused 8</td><td colspan="3">resume 48</td></tr>
</table>

c Indicates disposition of drop file when program is removed from main memory:

0 = Drop file, scratch files and output-type files are preserved so the program can be restarted. All modified pages belonging to write access files overwrite their current disk images and all other modified pages are written to the drop file. The resume address is stored in the drop file minus page.

1 = Drop file and scratch files are destroyed. Give output-type files to the output processor.

2 = Same as option 0 except the terminate state is set to report an abort (3D)

rc Return code. Can be any 8-bit value; the values are system standard values for product set numbers.

0 = Successful completion

4 = Non-fatal errors

8 = Fatal errors

resume Virtual bit address at which program is to be resumed when it is restarted.

SEND A MESSAGE TO THE OPERATOR (f = 001A)

A program may communicate with the operator through this message.

Message format:

Alpha (1)	r 16	len 16	unused 16	001A 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	message

Maximum message length is 80 characters.

r Response code returned by operating system when this message is processed. If no error occurs, the response code is zero; otherwise:

1 = Specified length of Beta portion in character bytes was out of range (either zero or greater than 80).

2 = Message could not be sent to the operator because no system buffer was available or operator was not logged on; error processing is indicated by the value of n.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or len is zero.

len If len = FFFF, Alpha (3) contains the length in character bytes and the virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length in character bytes of the Beta portion ($0 < \text{len} \leqslant 80$).

n Indicates action to be taken if message cannot be sent;

0 = Program will stop running until the message can be sent.

1 = Control will pass to error exit address in eea.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea is zero, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

Note: The only reasons a message cannot be sent to the operator are that the system message buffer is full or the operator is not logged on.

ADVISE (f = 0007)

A program issues the ADVISE message to inform the operating system of an anticipated need for virtual space in an attempt to avoid page faulting for the space, or to advise the system of pages no longer be used by this program.

Message format:

Alpha (1)	r 16	len 16	unused 16	0007 16
Alpha (2)	unused 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	ss 8	pgct 8	vba 48

r — Response code returned by operating system when this message has been processed. If no error occurs, the response code is zero.

1 = The ss field of Beta contains the error response

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len — If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

eea — Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea = 0, the error is considered fatal.

Bl
Ba — If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

ss Error response field:

0 = Normal completion

1 = System boundary violated

2 = Page count too large, or zero

3 = Page locked down and cannot be removed

pgct Page control fields further divided as follows:

pio	psz	pn
1	1	6

pio = 0 Attach or load pages
pio = 1 Remove pages

psz = 0 Small pages
psz = 1 Large page

pn Page count with a maximum value of eight ror small pages and one for large pages.

vba Virtual bit address referred to by the ADVISE action.

Notes:

1. When the virtual bit address is not defined in any virtual map, it is considered to be a definition of new free space. An appropriate entry will be made in the drop file map, and core space will be allocated.

2. If the virtual bit address is defined, a message sent to the page fault processor gives the address and length of the space indicated by ADVISE; treated as a page fault.

3. If more than one small page is indicated, they should be contiguous in mass storage, so as to generate only one read request.

4. When space is no longer needed, the operating system rewrites all modified pages in that space to mass storage. Unmodified pages are deleted from the paging system, thereby increasing the space available for paging.

RECALL (f = 0025)

The RECALL message allows a program to suspend its own execution for not less than 30 seconds nor more than 30 minutes. At the end of suspension, the program is recalled to an active status. No program that has initialized another program (is a controller) or has been initialized by another program (is a controllee) may issue this message. No user program that owns tapes may issue this message (this restriction does not apply to system privileged user programs).

Message format:

Alpha (1)	r 16	len 16	unused 16	0025 16
Alpha (2)	unused 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	unused 32	time 32

r Response code returned by operating system when this message has been processed. If no error occurs, a response code value of zero is returned.

1 = This message is not allowed for this program.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea=0, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word

time Period of suspension, specified as an integer in (hexadecimal) micro-seconds. The value of time must be no less than 30 seconds nor more than 30 minutes. Values outside this range will be set to the nearest interval limit.

LIST CONTROLLEE CHAIN (f = 0013)

A program can obtain a list of the controllee chain including the program level and descriptor number, the executable source file name, drop file name, and so forth.

Alpha (1)	r 16	len 16	m 8	unused 8	0013 16
Alpha (2)	j 8	b 8	eea 48		
Alpha (3)	Bl 16	Ba 48			

Beta (1)	s 8	t 8	c 8	unused 24	n 8	d 8
Beta (2)	k 8	tl 56				
Beta (3)	source 64					
Beta (4)	drop 64					

r Response code returned by the operating system when this message has been processed. If no error occurs, control proceeds normally; and this field contains the number of words returned in Beta. If control returns to the error exit address, the following values of r are significant:

1 = The length specified for Beta is zero.

2 = An illegal option was selected.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion of the message.

m Options for this message:

0 = List all controllees in the chain. The operating system returns four Beta words per entry list. Controllees are listed in ascending order, starting with the job control processor.

01 = List only this program. The operating system returns four Beta words.

02 = List only this program's controller. The operating systen returns four Beta words.

03 = List only this program's controllee. The operating system returns four Beta words.

j In this field, the operating system places this program's level in the controllee chain.

b In this field, the operating system places this program's descriptor number.

eea Virtual bit address to receive control if an error occurs during message processing ($r \neq 0$). If eea = 0, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

s The level of the program whose name is in the third Beta word of this entry will be placed in this field. (Level numbers range from two to five.)

t Descriptor number of the program whose name is in the third Beta word of this entry will be placed in this field.

c Descriptor number of this program's controller will be placed here.

n Beta (3) contains the name of a program that has informed the operating system to direct messages from the controller to n if they are not specifically directed to the controllee. The operating system places the descriptor number of n, a program in the chain, in this field. The number may be zero.

d Beta (3) contains the name of a program that has informed the operating system to direct messages from the controllee to d if they are not specifically directed to the controller. The operating system places the descriptor number of n, a program in the chain, in this field. The number may be zero.

k Descriptor number of this program's controllee will be placed in this field. This number may be zero.

tl Time limit of program whose name is in the third Beta word of this entry.

source ASCII name of the executable source file

drop ASCII name of the drop file

Notes:

1. The issuing program can determine its own position in the chain by comparing fields j and b with fields s and t.
2. Five program controllee levels is the maximum. Level 1 is a job control processor or terminal.
3. The descriptor number is unique and is associated with the program until it is disconnected.

LIST FILE INDEX OR SYSTEM TABLE (f = 0009)

With this message, a program can get a copy of its private file index, the public file index, or other specified system tables.

Message format:

Alpha (1)	r 16	len 16	c 16	0009 16
Alpha (2)	n 16	eea 48		
Alpha (3)	Bl 16	Ba 48		

Beta (1)	name 64					
Beta (2)	mcat 8	saddr 24	unit 8	dup 8	wlen 16	
Beta (3)	user/ref 32	idchr 8	type 8	slev 8	acs 8	
Beta (4)	torg 32	tlr 32				
Beta (5)	dc 16	ic 16	ec n 16	pri 16		
Beta (6)	tid 16	sid 24	unused 8	fc 16		
Beta (7)	unused 8	un 56				
Beta (8)	unused 64					

For options c = 0, 1, and 8, only the first four Beta words are returned. For option c = C, all eight Beta words are returned. Unused fields are reserved for future system usage.

r Response code returned by the operating system when this message has been processed. If no error occurs, a response code value of zero is returned. Otherwise,

211 = Number of files was 0.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified.

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion; for c = 2 through c = 5, len specifies available program buffer size for the requested system table.

c Option specifying the file index or system table to be listed.

0 = Public file index

1 = Private file index

2 = Timecard buffer

3 = Statistics buffer

4 = Bank update table

5 = Miscellaneous table

6 = Batch input files

7 = Specified part of private file index

8 = Specified filenames in file index

9 = Disk status table

A = Private files beginning with string specified in name

B = Public files beginning with string specified in name

C = The portion of the private file index whose disposition code is the same as the dc field specified in Beta (5).

n For c = 0, 1, 8, and C, n is the number of file index entries to be listed. The Beta area should be at least 8n words long for c = C and at least 4n words long for c = 0, 1, or 8. For the other options, n is the size of the table to be listed and the Beta area should be at least n words long. The program will move words from the table into the Beta area until either the table or the Beta area is exhausted. The value of n must always be greater than 0.

eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea = 0, the error is considered fatal.

Bl
Ba
If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

name
Operating system places here the ASCII file name from the table requested. If option c = 7 has been chosen, the listing begins at the file entry following name given, or first entry if name = 0. If option c = 8 has been chosen, the user places here the filename for which information is to be returned. Operating system places 0 here if name given is not found when c = 7 or c = 8. For c = A or B, name contains a string of characters which are left-justified and zero filled.

mcat
Operating system places here the management category for this file (all will be mass storage files):

0 = Permanent file

1 = Scratch file

2 = Output file

3 = Write temporary file

5 = User-created drop file

6 = System-created drop file

7 = Batch file

saddr
First page address of physical segment is placed here by operating system.

unit
Logical unit number on which file was found.

wlen
File length (in small pages); placed here by operating system.

dup
Reserved for future system usage.

user/ref
Operating system returns user number of original owner if file was given to USER 1 and option c = 7; otherwise, the operating system returns the ref field from the file index table.

idchr
If non-zero when c = 7, only files whose first non-blank non-null character matches idchr are listed. If no such files are found, the operating system returns 0 to the name file. Not used otherwise.

type
File type; placed here by the operating system:

0 = Physical data

1 = Virtual data

2 = Virtual code

slev
Security level of this file (0 to 255).

acs
File access permission taken from file index:

1 = Write access }
2 = Read access } May be used in logical combinations

torg Time in 16–second units at which the file originated.

tlr Time in 16–second units at which the file was last referenced by opening.

dc Disposition code. A two-character mnemonic indicating how the file is to be disposed.

SC = Scratch; destroyed at end of task

PR = Print on any available printer

PU = Punch

IN = Input for Batch processing

For c = C, this field specifies that portion of the private file index which is to be returned.

ic Internal characteristics. A two-character mnemonic indicating the format of the file.

AS = Format is 8-bit ASCII. If the file has a disposition code of PR, the file has ANSI carriage control characters.

BI = Format is binary.

PA = Format is 8-bit ASCII with ASCII control characters.

ec External characteristics. A two–character mnemonic giving the print or punch representation of the file.

26 = Punch in 026 keypunch format

29 = Punch in 029 keypunch format

*B = STAR binary format

80 = Punch 80 column binary format

pri Priority level for a file to be output at its originating terminal or front-end processor. This field is defined, but not currently supported.

sid Site identifier. A three-character identifier for the front-end processor. This field is defined, but not currently supported.

tid Terminal identifier. A two–character mnemonic identifying a terminal. This field is defined, but not currently supported.

fc Form code. This field is defined but not currently supported.

un User number. A seven-character user number from the front-end processor. This field is defined but not currently supported.

NOTE: Because entries in FILEI may be moved about during the execution of a job or task, the user of contiguous LIST FILE INDEX requests, option c = 7, cannot be sure of results. Since the first request, when scanning is stopped, will give the filename, the user cannot rely on the first request to indicate where scanning is to begin for a subsequent request.

MISCELLANEOUS (f = 0024)

This message allows a program to manipulate its time limit and to determine a variety of information concerning itself, its controller, and its controllees.

Message format:

<table>
<tr><td>Alpha (1)</td><td>r 16</td><td>len 16</td><td>c 16</td><td>0024 16</td></tr>
<tr><td>Alpha (2)</td><td>unused 16</td><td colspan="3">eea 48</td></tr>
<tr><td>Alpha (3)</td><td>Bl 16</td><td colspan="3">Ba 48</td></tr>
</table>

Beta is described with the information options.

r   Response code returned by operating system when this message has been processed. If no error occurs, the response code will be zero. If r = 1, an error has occurred.

214 = Beta buffer length error. Either the first word address of Beta plus length is greater than the maximum user virtual address, or the Beta buffer is too small for the number of requests and length specified

len   If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3), and len is the length of the Beta portion.

c   Information option:

00 = Modify time limit as follows:

Beta (1)	New time limit in microseconds
Beta (2)	Existing time limit; returned by operating system

01 = Get user ID number and bank account as follows:

Beta (1)	ASCII user ID number
Beta (2)	Amount of time in the user's bank account (integer in microseconds); returned by operating system

02 = Reserved for future use

03 = Get time limit and priority; both returned by operating system:

Beta (1)	Existing time limit in microseconds
Beta (2)	Existing priority

04 = Reserved for future use

05 = Get controllee's termination state

Beta (1) See diagram below:

Beta (1)	unused 48	rc 8	cts 8

rc Return code

0 Successful

4 Non-fatal error

8 Fatal error

cts Controllee's termination state

0 Still active

1 User terminal break to exit card

2 Operator break to exit card

3 Operator job drop break to 6/7/8/9 card

4 Operator user drop entire batch deck

3D Abort

3E Normal termination

39 Normal termination

06 = Get controllee name and place; both returned by operating system:

Beta (1) Source file name for controllee
Beta (2) Drop file name for controllee

07 = Get controllee name and place; both returned by operating system:

Beta (1) Source file name for controller
Beta (2) Drop file name for controller

08 = Get this program's name and place; all are returned by operating system:

Beta (1) Source file name for this program
Beta (2) Drop file name for this program
Beta (3) Suffix, level in controllee chain, and ASCII ID number

09 = Get elapsed time and page fault count; all are returned by operating system:

Beta (1)	pgflt 16	cpuchg 48
Beta (2)	drflt 16	memchg 48
Beta (3)	exio 32	remio 32
Beta (4)	imio 32	syschg 32

pgflt	Count of total page faults
cpuchg	Central processor charge
drflt	Count of page faults found on drum
memchg	Memory charge
exio	Explicit input/output charge
remio	Remote input/output charge
imio	Implicit input/output charge
syschg	System call charge

0A = Get clocks; all are returned by operating system:

Beta (1)	Master clock value
Beta (2)	ASCII clock value, expressed as HH:MM:SS (hours:minutes:seconds)
Beta (3)	Calendar value, expressed as MM/DD/YY (month/day/year)
Beta (4)	Value of millisecond station clock
Beta (5)	Value of millisecond central processor clock

0B = Not used; reserved for future use

0C = Grab time from repository

Beta (1)	rc 16	unused 48
Beta (2)	gtime 64	

rc Return code

0 Successful

1 No repository access

2 Not enough time available

gtime Time value, in floating point form, representing number of minutes for which grab is to be made. For rc = 2, maximum time available for grab is returned by the system.

eea Virtual bit address to receive control if an error occurs during message processing (r ≠ 0). If eea = 0, the error is considered fatal.

Bl
Ba If the Beta portion of the message is not contiguous to the Alpha portion (len = FFFF), these parameters indicate the length in full words and virtual bit address of the Beta portion's first full word.

POOL FILE MANAGER (f = 0026)

This message includes a variety of options related to pool files.

Message format:

Alpha(1)	r 16	len 16	c 16	0026 16
Alpha(2)	unused 16	eea 48		
Alpha(3)	Bl 16	Ba 48		

Beta is described with the message options.

r Response code returned by the operating system after message is processed. If no error occurs the response code is zero. A non-zero response code does not necessarily mean an error has occurred, but the error exit is taken.

0 = Successful

11 = Pool name already attached by this user

12 = Pool name undefined

13 = Already in four pools

14 = Pool not attached

15 = May not attach to pool; user has no access to pool

16 = Undefined user number

17 = Duplicate pool name

18 = Unable to destroy pool

19 = Pool access directory full

1A = Pool list full

1B = Invalid pool

1C = Invalid pool name

1D = Not pool boss

214 = Beta buffer length error

len If len = FFFF, Alpha (3) contains the length and virtual bit address of the Beta portion of the message. Otherwise, Beta is assumed to begin at Alpha (3) and len is the length of the Beta portion.

c Message options:

In each of the following options, Pool name occupies the entire Beta (1) word. Pool name contains up to 8 alphanumeric characters and must start with a letter; it is left-justified and blank filled. User numbers each occupy an entire word and are right-justified and zero filled. User numbers can range from 1 to 999999.

1 = Create pool. Adds the pool name to the pool list and clears (zeros) the pool access directory from that pool. The creator becomes the pool boss.

Beta (1) Pool name

2 = Destroy pool. If no users are attached and no files are in the pool, the pool name is deleted from the pool list.

Beta (1) Pool name

3 = Grant access to pool. Places the specified user numbers into the pool access directory. If either len or Bl is 1, all users may access the pool.

Beta (1) Pool name
Beta (2-i) User numbers to be granted access

4 = Attach to pool. Attaches requestor to the named file pool.

Beta (1) Pool name

5 = Detach requestor from pool.

Beta (1) Pool name

6 = Remove access privilege. Specified user numbers are removed from the pool access directory.

Beta (1) Pool name
Beta (2-i) User numbers to be removed

7 = List users having access to the pool.

Beta (1) Pool name

Beta (2-i) List of user numbers having access to the pool; as permitted by the r specification.

8 = List pools and pool boss. All non-zero entries in the pool list file are copied from Beta (1) to Beta (i). The number of words copied is indicated in r.

Each pool list entry returned has the following format:

Beta (1)	pcount 16	pptr 16	pfree 12	pboss 20
Beta (2)	poolnm 64			

pcount Count of users attached to this pool

pptr Pointer into PAD that contains a list of user numbers which can attach to this pool

pfree Currently unused

pboss User number of the boss of this pool

poolnm Name of this pool in ASCII

eea Virtual bit address to receive control if an error occurs during message processing (r = 0). If eea = 0, error is considered fatal.

Bl

Ba If the Beta and the Alpha portions are not contiguous (len = FFFF), these parameters indicate the length (Bl) in full words and virtual bit address (Ba) of the Beta portion's first full word.

Error codes related to this message are described in appendix H.

JOB PROCESSING 6

Under STAR-OS, jobs may be processed by either the batch processor or the interactive processor. Under the batch processor, all communication concerning job processing is carried on with the system through control card information submitted as part of the job. Under the interactive processor, the user may communicate directly with the system from an interactive terminal. The user can call on the system to store programs and data on mass storage units; once filed, the information may be accessed through either the batch or interactive processors.

Both processors recognize three general classes of control commands from the user: identification, execution, and termination. The identification command identifies the user and makes his files available to him – in batch mode, through the STORE card; in interactive mode, through the LOGON line.

The execution command requests specific system action by submitting, on a batch mode control card or on an interactive control line, the name of an existing file, such as the name of the file containing the FORTRAN compiler, or the name of a file containing a user procedure.

Job termination depends upon the user mode. Under the batch processor, a job terminates when all control cards have been processed, or an EXIT card is encountered and no error has been detected. If the batch processor is activated from an interactive terminal, the job may terminate normally when an end-of-job character is detected at the end of the file. The interactive user may terminate his job with a BREAK character, or he may disconnect the terminal from the system with the disconnect request.

INTERACTIVE PROCESSING

The interactive processor allows the user to request information about his job and communicate with a program regarding functions in addition to the initiate, execute, and terminate control lines.

ESTABLISHING USER IDENTITY

At an interactive terminal, a user establishes identity with the LOGON line in the following format:

```
LOGON user-number suffix account-id level password
```

Blanks separate fields in the line. The new line key terminates the line. The parameters are:

user-number	A six-digit number, such as 999997, which uniquely identifies the terminal user.
suffix	Letters A, B, C, or D under which the user can operate. Since each user may have up to four programs active in the system at one time, each program must be identified uniquely by the suffix.
account-id	An identifier of 1 to 6 characters used for accounting purposes.

level — Single character defining a user's security access level; required if a password is given but is otherwise optional. The following is a list of characters and their corresponding numeric security level.

Character	**Level**
Undefined	0
P or omitted	2
A	3
S	5
K	7

The file management functions and utilities permit a range of security levels. The user cannot, however, read or execute a file whose security level is greater than the one at which the user logged on. If the user creates files without specifying a security level, the security level at which he is logged on will be used.

password — A predetermined string of characters used to determine legality of access to information having security classification. This is an optional parameter which is defined but not currently implemented.

Example of LOGON line:

```
LOGON 999997 A 400SDS
```

When the identification line is entered, the interactive processor verifies the items in the line; and, if no entry for the user exists, it assigns and fills a user table entry. Appropriate entries are made in the active file index in central memory for the user's private files.

INITIATING PROGRAM EXECUTION

After the user has entered identification and the interactive processor has ascertained that no other program is active under the suffix entered, a program may be initiated by an execute line. In the form shown below, Δ indicates a required space.

task-nameΔ/Δt $\left\{\begin{matrix}\Delta \\ ,\end{matrix}\right\}$ cΔ/Δmessage

task-name — The task-name is required; the remainder of the line is optional. Task-name is the name of a virtual code file to be placed in execution; the first 6 characters must be unique.

/ — This character delimits the start time and job class information; when present, it is preceded by a blank.

t — Decimal number giving time limit, in seconds, to be allocated for this task; it is preceded by a blank. Default time is 10 seconds.

$\left\{\begin{matrix}\Delta\\ ,\end{matrix}\right\}$ A separator is required between time and job class information; it may be a space or comma.

c Characteristic of the task which influences the way in which the system allocates resources to the task. The four characteristics available are:

- P A priority task requiring rapid response. The use of this characteristic is controlled administratively.
- I An interactive task typically involves a large amount of communication between the user and his program. This is the default characteristic for interactive processing.
- B A batch task typically involves little or no communication between the user and his program. This is the default characteristic for batch processing.
- S A standby task typically involves no communication between the user and his program. Such a task is run only when P, I, or B characteristic jobs are not utilizing all of the available resources.

This character delimits end of time and job class information; it is preceded by a blank.

message The message format varies according to the program; it may be used to pass parameters to the program. It must be preceded by a blank. By convention, addresses on control cards or execute lines are assumed to be hexadecimal and all other values are assumed to be decimal. A hexadecimal value may be used in place of a decimal value by preceding it with the # sign. If an address is preceded by #, the sign is ignored.

All characters present in the execution line after the second slash are passed to the task-name program as a message. If the optional time/job class sequence is omitted, the message, through the end of the line, is passed to the task-name program; and default values are assigned for time limit and job class.

INTERACTING WITH PROGRAM

When a program is active under one of the user's suffixes, the user may send a message to that program from a terminal. A user cannot send a message to a program active under a suffix of another user. The message may have any format, but it must not cause the operating system to take action other than sending the message to the active program. If a second message is sent before the first message has been accepted by the program, the second message will replace the first, and the first message will be lost.

MAKING REQUESTS OF OPERATING SYSTEM

The user makes requests of the operating system, using a request line. Each request line contains a message which must be prefaced by a special character that distinguishes it from other classes of messages and directives. The special character (sc) is defined by the installation. To make a system request from a terminal, the user need not have a program active under his suffix entered in the LOGON line. The system requests are as follows:

(sc)T Get current time and date.

(sc)S Get current state of program active under user's suffix; possible responses appear in Appendix G.

(sc)BB List current accounting information for program active under user's suffix.

(sc)? Get current time, date, program state, and accounting information for program active under user's suffix.

(sc)SU List current activity of programs active under all user's suffixes, A, B, C and D.

(sc)BP List time remaining in repository to which the user belongs. Time remaining reflects balance after initial time increment is granted at log on, and additional time is drawn from the repository.

(sc)Gxx Draw xx minutes from the repository.

(sc)U List current time consumed by user this session, including any time drawn from the repository during this session.

(sc)PR List number of job tasks in interactive (I) class waiting to be connected to an alternator slot.

(sc)I Send program to current interrupt routine, if program is so enabled.

(sc)OP message Send message to operator's terminal.

TERMINATING CONNECTION

The user at the terminal ends communication with the operating system by using the system request (sc)BYE. Also, the user may break the connection with the logged on suffix and remain active on the terminal under a new suffix by using the system request (sc) suffix. The suffix given in the message is the new one under which the user will become active. Any programs active when either form of the disconnect message is given will remain active, although the results may be affected by the fact they are no longer connected to an interactive terminal.

An installation may define a special BREAK character, which also is used to terminate the task currently running at the active suffix. Task execution will be aborted and control transferred to the next level controller. If the aborted task has no controller, it can be restarted by executing its drop file.

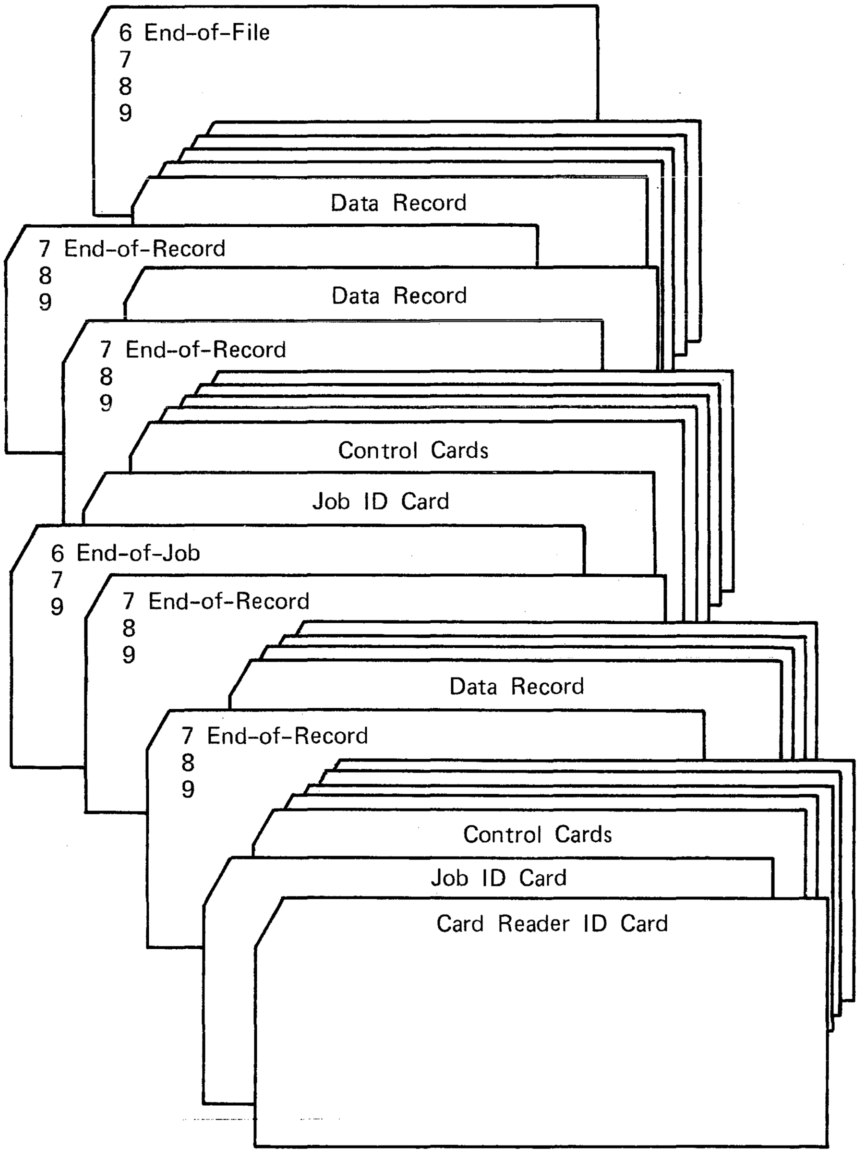
BATCH PROCESSING

All jobs are presented to the batch processor in the form of physical files. These files must be record structured type; such files are described in Appendix C. The first record is assumed to contain job control information. The physical files may be submitted to the batch processor through a card reader, or they may be directed to the batch processor by an execute line entered at an interactive terminal.

BATCH PROCESSOR DECK

For batch processing through the card reader, a deck must be prefaced by a card reader identification card and contain at least one job. The file must end with an end-of-file card. Each job in the deck must have its own job identification card and terminate with an end-of-job card. Each job must contain at least one record. The control card record containing the job identification and related control cards must be the first record. Each record is terminated by an end-of-record card. The end-of-file card may be preceded by end-of-record and end-of-job cards, or it may be used alon·, representing all three functions.

BATCH PROCESSOR DECK



CARD READER IDENTIFICATION CARD

This card identifies the user and presents required information to the processor. The format is:

Parameter	Card Columns	Contents
Card reader identification	1-5	STORE
	6	Blank
User number	7-12	Expressed as a set of 1 to 6 digits
	13	Blank
Account identifier	14-19	1 to 6 characters, used for accounting purposes
	20	Blank
File name	21-28	File name to be assigned to entire deck following card reader identification card; 1 to 8 characters. If the batch processor is scheduled, this file will be destroyed by the batch processor when the last job in the deck has been processed.
	29	Blank
File type	30	A File named is an absolute binary file R Record-structured file; default
	31	Blank
File organization	32	Blank. Initially, all files will be set up as physical files.
	33	Blank
Processing mode	34	B File to be scheduled for processing by batch processor Blank File not to be scheduled for batch processing; default
	35	Blank
Security level		Reserved for future system use. Currently, all files are stored at security level 0. If the batch processor is scheduled for this file, it will run at security level 2.
	37-45	Blank
File size	46-47	In hexadecimal number of blocks; default value is 8 blocks. Maximum size is 70 blocks to contain the entire card deck submitted.
	48	Blank

Parameter	Card Columns	Contents
Directory size	49-50	Number of blocks to be added to end of file to hold file directory.
	51-60	Blank
	61-76	Identification information for record-structured files. 1 to 16 characters are appended to end of file directory.
	77-78	Blank
Conversion type	79-80	Blank; 29 for 029 punch; 26 for 026 punch. Blank implies no keypunch code conversion is to take place.

Batch jobs containing mixed mode (both ASCII and STAR binary) cards should contain an R in column 30 to produce a record-structured file.

JOB IDENTIFICATION CARD

The job card must contain the job name, and can contain a time limit and a termination value (TV) for the job. The job name can be 1-8 letters or numbers, and must start with a letter. The time limit is currently the maximum task time (cpu, IO, etc.) and is specified by a T followed by the (decimal) number of seconds. If a time limit is specified, it is separated from the job name by a comma. The terminate value is used to establish initial criteria for normal and abnormal job termination. The last parameter on the job card must be followed by a period or right parenthesis.

CONTROL CARDS

All cards in the control card record of a job have the same general format. The first element must be a task name of 1 to 8 characters. Parameters may follow on the control card; their formats are determined by the task name. Separator characters (between parameters, or between the task name and parameters) may be any of the following:

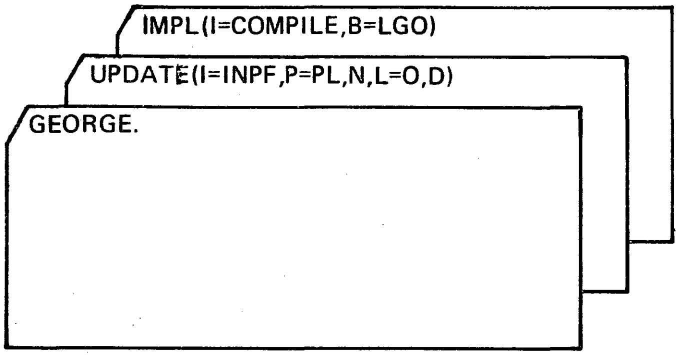
```
( , / = + -
```

Blanks can precede task names. Blanks to the right of the last character will be ignored.

The control card information is always terminated with a period or right parenthesis. If no terminator appears on the first control card, the system assumes that control information is being continued on the next card, starting in column one.

Examples of Control Cards:

General format: task name,message



```
IMPL(I=COMPILE,B=LGO)
UPDATE(I=INPF,P=PL,N,L=O,D)
GEORGE.
```

The batch processor explicitly searches for the following four task names. All others are assumed to be executable private or public files.

TV,value.

EXIT.

READCC,lfn.

COMMENT.

The TV control card gives the user considerable latitude in controlling the execution of his job. All job steps, when they terminate, return a code which can be used to alter the execution sequence within a job. By convention, the following codes have special meaning:

0 = Successful completion
4 = Non-fatal errors
8 = Fatal errors

The terminate value, 0 to 255 (decimal), is set initially by the TV parameter on the job card or, if omitted, to the installation-defined default value. The terminate value can be changed during the execution of a job via a TV control card. At the end of each job step, the terminate value is compared against the code returned. If the code returned is greater than the terminate value, abnormal job termination processing will occur; otherwise, processing will continue normally. In addition, the value of the highest code returned is maintained by the system and can be tested via the TV control card. If the value of the highest code returned is greater than the value on the control card, abnormal job termination processing will occur; otherwise, processing will continue normally. The value of the highest return code is initially set to zero.

To set the terminate value, the card format is:

TV,value+.

For example, to set the terminate value to 7:

TV,7+.

To test the terminate value against the highest code returned, the card format is:

TV,value.

This card format is also used to set the terminate value.

For example, to test whether fatal or non-fatal errors had occurred in previous job steps, one might write

TV,1.

When abnormal job termination results from a test of either the return code or the highest code returned, the terminate value is set to 255, the maximum value.

The EXIT control card is used to establish a control path to be followed when abnormal job termination results from a test of the return code or of the highest code returned. When abnormal job termination occurs, the batch processor does not process the next control card in the sequence. Instead, it searches for an EXIT control card. If an EXIT card is found, the terminate value is set to 255 and normal job processing resumes with the first control card which follows the EXIT card. When an EXIT card is encountered during normal job processing, the job is assumed to have reached its end and is terminated normally.

The READCC control card causes the batch processor to read control cards from the file name specified as the first parameter of the control card. The file is assumed to be in ASCII format. When an EOF control character is detected, the batch processor resumes reading control cards from the normal control card stream. The file specified in a READCC control card may contain any legal control card, including READCC, so that several levels of control and files can be used. Currently, 8 levels are permitted.

The COMMENT control is used to insert messages into the job dayfile. The message must be placed after the terminator which follows the word COMMENT and before the end of the control card. If the message is too long for one COMMENT card, it may be continued on as many COMMENT cards as necessary.

FILE, JOB, AND RECORD SEPARATORS

Each deck is terminated with an end-of-file card or a card having a 6/7/8/9 multiple punch in column one. All jobs between the card reader ID card and the end-of-file card will be run under the same user number.

Jobs within a deck are separated by an end-of-job card or group separator (6/7/9 multiple punch in column one).

Jobs are divided into records, separated by end-of-record cards (7/8/9 multiple punch in column one). Any record separator may also contain an ASCII name of one to eight characters, beginning anywhere after column one and starting with a letter. This name will be associated with the data record following it. An end-of-record card may be followed by an end-of-job card.

CONTROL CARD PROCESSING

The batch processor attempts to process control cards in the order that they appear in the control card record. The task names that the batch processor explicitly searches for will initiate special processing. For other task names, the batch processor searches for a file of that name, and first searches the list of private files for the user. If an executable private file is not found with the same name as the task name, then the batch processor searches the names of the public files in the system for an executable file with that name. If no such file is found, the batch processor aborts the job.

When an executable private or public file is found with the task name, the batch processor initiates execution of that file as a controllee of the batch processor, and it passes a message to the controllee that contains the information on the control card. The batch processor stops running until it is reinitiated by a message from the controllee. When a message indicates the controllee has successfully completed processing, the batch processor continues with the next control card.

Standard routines, such as the FORTRAN compiler and utilities, exist as public files. A user may have and execute his own compiler or utility with a private executable file having a syntactically correct name, since private files are scanned first.

OUTPUT FILES

Before it processes each control card, the batch processor checks for special file names. If a private file named OUTPUT is classified as an output type file, the batch processor changes the name of the file to Phjobnam, where h is a sequence number and jobnam is the leftmost 6 characters of the job name. When the job terminates, all output files will be printed as if they were one family of files. To print or punch a file directly, the file naming conventions described under DEVICE TYPE OUTPUT FILES in section 4 should be followed.

DAYFILE

Before it starts to process a job, the batch processor creates a file named PXDAYFLE in which it writes a record of the job's execution, including an image of each control card as it is processed and any message sent to it. Errors detected by the batch processor are indicated in the dayfile, as are error messages sent to the batch processor by any controllee it initiates. The dayfile also includes the time at which each control card or message was processed.

When the job terminates, the dayfile is printed as the last file in the family of output files.

DATA RECORDS IN A JOB (INPUT FILE)

All records following the first record of a job are considered to be data records. These data records can be named or unnamed and can contain ASCII data, STAR binary data, or a mixture of both.

Named data records are identified by a file name punched on the record separator card at the start of the data record. When a batch job is initiated, this file name is used by the batch processor to create a permanent file which contains the data record information. Data record information can be accessed during job execution by referencing the file, using the name given on the record separator. Private files of this nature are destroyed by the batch processor at job termination.

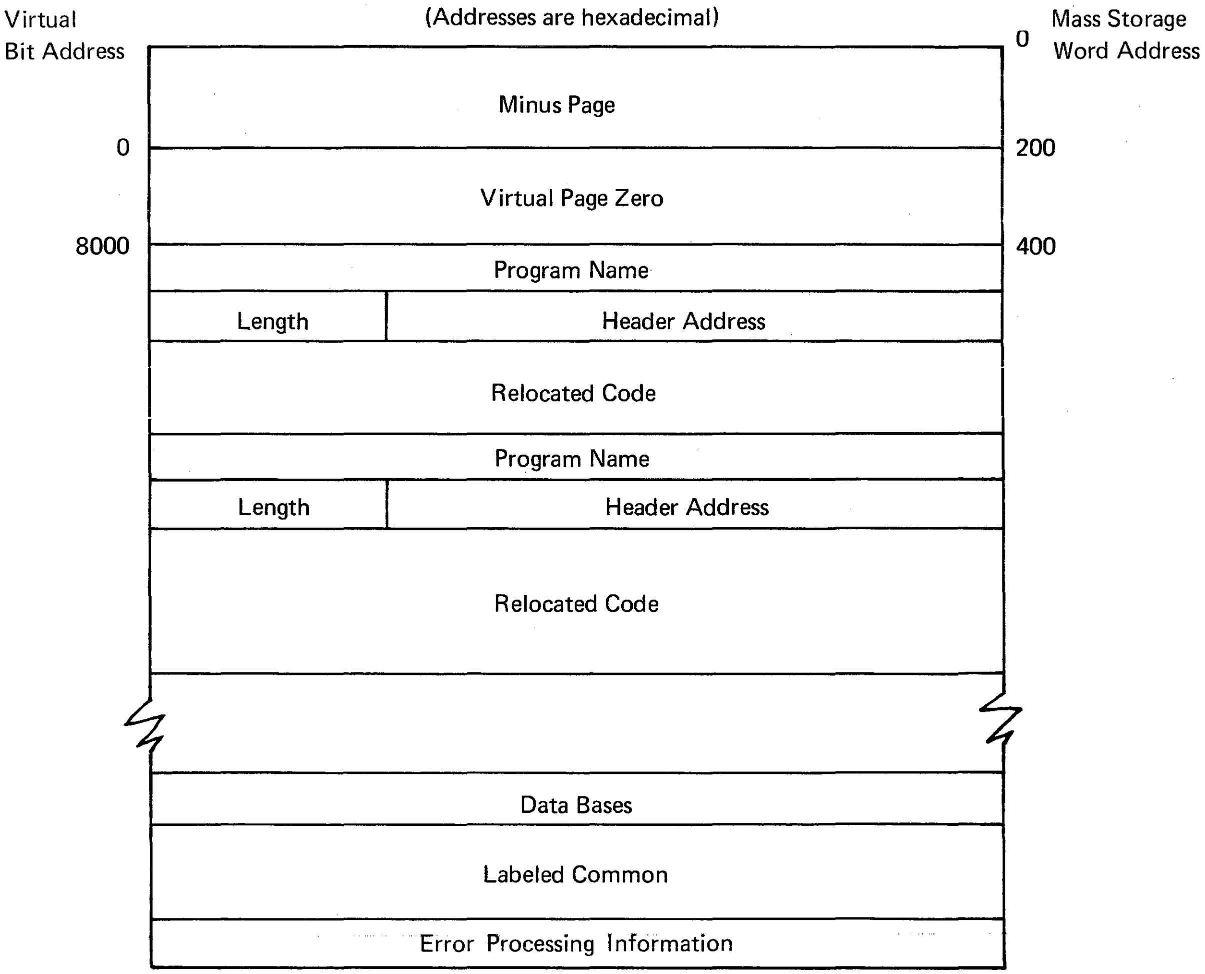
Unnamed data records are identified by the absence of a file name on the record separator card. When a batch job is initiated, the contents of the first unnamed data record are transferred to a permanent file named INPUT. Subsequent unnamed data records are transferred to file INPUT whenever the batch processor detects that file INPUT has been referenced by a job step (task). Access to unnamed data records is provided by referencing the INPUT file. At job termination, the INPUT file, if created, will be destroyed by the batch processor.

THE LOADER 7

The loader may be called by the batch processor via a control card or from a terminal as an interactive task. The loading process involves loading object modules from the user's file, followed by the loading of the user library, if one is specified, after which the remaining unsatisfied external references are fetched from SYSLIB, the system library.

Object modules may exist in either of two formats: library format which must be generated by the Object Library Editor OLE; and compiler or assembler-generated format.

The loader transforms one or more physical files into a virtual code controllee file. The physical files must contain object modules conforming to the conventions set forth in appendix D. The first module on each physical file must be a Module Header Table. The controllee file produced will be of the following form.



Information required by error processing routines and debugging routines is stored at the end of the virtual file. (See appendix I.)

In many cases, only one bound virtual map entry will be created for the controllee file. If discontinuous address spaces or large pages are defined, additional map entries will be made. Labeled common is preset to all zeros (64-bit words) and data is loaded as specified. Numbered and blank common areas are mapped into the drop file map. Common blocks that are multiples of 512 will be forced to small page boundaries. On the first access to this common area, the user will get a page created by the operating system and initialized to:

000C1F1C$_{16}$ 000C1F1C$_{16}$

Any modified pages of the controllee file will be mapped into the drop file by the operating system.

The files required by the loader are:

	Default Names	**User-specified Options**
Controllee file	GO	CNTROLEE=lfn_1
Output file	OUTPUT	OUTPUT=lfn_2
Input file	BINARY	lfn_1,lfn_2

The number of optional loader files may not exceed 13, except for user files which may not exceed 10:

SYSLIB	System library	Optional
LIBRARY	Files specified in library format	
USER FILES	lfn_1,lfn_2,lfn_3, . . .	

BATCH LOADER

The format of the batch loader control statement is:

LOAD(lfn_1,lfn_2, . . . , lfn_i,options)

The names of the primary files containing the object modules to be loaded is given for lfn_i. The files must be type physical, contain no minus page, and not in library format.

The loader options available to the user must be in the format keyword=plist and terminated by a right parenthesis or period.

INTERACTIVE LOADER

The control line is written as shown below. It activitates the loader in interactive mode. Blanks must separate all elements of the control line. The loader response requests user private file names.

LOAD / n I /

LOADER OPTIONS

These options can be used with either the batch or interactive loader. The underlined portion of the option keyword indicates the minimum characters that must be specified.

ORIGIN=bit-address

If this option is not specified, the first module encountered is loaded starting at the hexadecimal address 8000. If the origin option is specified, the first module encountered is loaded at location bit-address. If the bit-address is not at a page boundary, it will be adjusted downward until it is on a page boundary.

AGA=ƀ

This option provides automatic group allocation based on subroutine-to-subroutine call structures, as can be determined best by the loader. The allocation option allows the loader to attempt to allocate routines that call each other in groups. This option increases the amount of loading time, but it may reduce program execution time by reducing the frequency of page faulting. If the allocation option is not selected, routines will be allocated in the order encountered on the input file.

ENTRY=routine

The symbolic entry point name of a routine is given. The loader searches all input and private libraries for the entry routine, then uses the routine entry point as the transfer address. If no entry name is specified, MAIN.ƀƀƀ is used. (ƀ is a blank; the value 20_{16} is used for ƀ.)

CNTROLEE=lfn,len

The controllee file name is given for lfn and the file length in pages is given for len. If the user wishes to specify the name and/or length of his loaded file, this option should be used. If file name lfn already exists under this user number, an error is returned. The default file name is GO and default length is 25 hexadecimal pages.

LIBRARY=lfn_1,lfn_2, . . . , lfn_n

The user may specify one or more private and/or public libraries. Only modules referenced are loaded from the libraries; in satisfying externals, the libraries are searched in the order given. If no libraries are specified, only the public library is used to satisfy externals. (See Object Library Editor.)

EQUATE=nam_1,sub_1, . . . ,nam_n,sub_n

This option allows the external names (nam_i) generated by object modules to be replaced by externals having the sub_i names. The user can replace an external name with another name. Names are scanned in pairs; the first is the name to be sought, the second the name to replace it. In replacing external names, program names are searched before common names. Common names must be preceded by an asterisk, as for example, *COMNAME. An asterisk by itself indicates blank common.

CDF=len

The controllee drop file length option is used when executing the controllee file, if the drop file is to be made a specific length. Normally, the size of the drop file is equivalent to the length of the controllee file plus the length of both blank and numbered common plus two sectors. The resultant length is normally the amount of free space used by an executing task. If the task requires more free space than provided by default, then this option should be used.

OUTPUT=lfn

Load maps and error messages produced during load operations are sent to the lfn specified. A file name of 1 to 8 characters is given if maps are to be sent to the printer. The default file name is OUTPUT when the option is omitted in batch mode; when omitted in interactive mode, the load maps error messages are sent to the user's terminal.

lfn_1,lfn_2, . . . , lfn_n

This option designates user private input files are to be merged and then loaded.

lfn_i Names of private user files, listed in the order in which they are to be merged. The files must be type physical (no minus page) and must exist in either the user's private file chain or, if the POOLNAME was attached, in the pool file list.

If unsatisfied externals remain after a complete scan through these files, the library file is searched.

DEBUG=mod_1,mod_2,. . . ,mod_n

The user can specify one or more debug routines. The loader links to the debug versions; a call to NAME links to NAMEQ. This option applies only to library routines, and the routine and its debug version must exist on the same library file. When the debug routine NAMEQ calls NAME, NAMEQ links to NAME and not to NAMEQ.

VR=versname

This option aids the user in managing the development and usage of a program by providing an identification which is recognizable in a dump. The user provides a version name containing 1 to 8 characters which is stored, left-justified and blank filled, in register #A of page zero. In addition, the date and time are stored in registers #B and #C, respectively. If this option is not exercised, register #A will be blank filled.

The format of registers #B and #C is:

#B	mm/dd/yy
#C	hh:mm:ss

GROUP OPTIONS

The user may specify certain routines and common blocks to be loaded as groups. This may be accomplished by the following options:

GRSP=entry$_1$,entry$_2$,. . . ,entry$_n$ {,bit-address}

GRSP=com$_1$,com$_2$,. . . ,com$_n$ { ,bit-address}

Group on small page.

GRLP=entry$_1$,entry$_2$,. . . ,entry$_n$ { ,bit-address}

GRLP=com$_1$,com$_2$,. . . ,com$_n$ { ,bit-address}

Group on large page.

These options take routines or common areas specified by entry$_i$ and group them starting on a small or large page boundary, as designated by the option. A bit address may be specified by the #bit-address parameter; such an address must be on a page boundary, and for large page options, the address must be on a large page boundary. Common names must be preceded by an asterisk, as in *COMNAME. By itself, an asterisk indicates blank common.

GRLPALL=ƀ

Group everything for large pages.

This option groups code, data base, and labeled common on large pages and maps them into the bound implicit map. Numbered and/or blank common is grouped in large pages and mapped into the drop file map. The user can specify his code origin, which must be on a large page boundary and cannot be zero. The default code origin is #400000, with numbered and/or blank common being origined at the next large page boundary following the allocation of code, data base, and labeled common.

GROS=com$_1$,com$_2$,. . . ,com$_n$ { ,bit-address}

Group and relocate only on small page.

GROL=com$_1$,com$_2$,. . . ,com$_n$ { ,bit-address }

Group and relocate only on large page.

Common areas are to be relocated only without generating a map entry, allowing the user to do his own mapping at execute time. A bit address may be specified by the bit-address parameter; such an address must be on a page boundary, and for large page options, the address must be on a large page boundary.

BATCH LOADER EXAMPLE

The following is an example of a batch loader control statement:

```
LOAD(SYSF1,SYSF2,SYSF3,ENTRY=KERNEL,ORIGIN=40000,CNTROLEE=SYSTEM)
```

This statement directs the loader to use files SYSF1, SYSF2, and SYSF3 as object module files and to begin loading with the first module in SYSF1. The main entry point is KERNEL and the code is loaded beginning at 40000. The name of the controllee file produced by the loader will be SYSTEM.

INTERACTIVE LOADER EXAMPLE

In the following example, uppercase letters indicate communication from the interactive loader; lowercase represents the user's reply, which is terminated by a line feed (lf).

INPUT ?	Request from loader
lfn_1,lfn_2,lfn_3 (lf)	User supplied private file names
ORIGIN ?	Request from loader
28000 (lf)	First module loading bit address
ENTRY ?	Request from loader
(lf)	User indicates no option
ANY OTHER OPTIONS ?	Request from loader
CN=TONY	User indicates controllee option
CONTINUE	Answer from loader
OU=PRINTMAP	User indicates loadmap option
CONTINUE	Answer from loader
(lf)	Terminates options and starts load operations

DEBUGGING 8

DUMP

Two dump routines are available to the user:

DUMP The interactive or batch user can record characteristics of an executed job. For normal termination, a DUMP control statement with explicit DUMP directives is required to indicate known ranges and symbols. A dump is provided automatically upon abnormal termination; default ranges are used.

MDUMP The user can dump specified areas of mapped virtual memory. MDUMP is a binary object module callable by programs written in FORTRAN.

The user may initiate dumps in error, interactive or batch modes.

ERROR MODE

Error dumps are initiated by the batch processor when a fatal user error occurs. Standard information includes:

Program address (from invisible packages, see page F-2).

Contents of memory from 50 words preceding program address to 50 words following program address.

Subroutine traceback.

Alpha and Beta words if the word preceding program address contains an exit force instruction, implying that one of the system messages was used just prior to program termination.

INTERACTIVE MODE

DUMP may be called from an interactive terminal after job termination. A control statement requests a dump of specified areas of a user's virtual memory; this utility also may be used to dump an absolute binary file in loader created formats. If DUMP is called as a controllee, certain areas will not contain current values; the bound virtual map and drop file map are used; and files in open status at job termination, but subsequently closed by the system, are re-opened.

BATCH MODE

A control card is used with the batch processor to dump areas designated by address range or by name. DUMP is initiated as a controllee for the batch processor each time the DUMP control card is encountered in the control card stream.

Both the drop file and the source file may be dumped; the file name must be specified.

CONTROL STATEMENTS

Formats for entering a DUMP request from an interactive terminal:

DUMP / t c /

or

DUMP / t c /(lfn_1,lfn_2,len,I=lfn_3)

lfn_1 Name of source file or drop file.

lfn_2 Optional, output file name.

len Length of output file pages.

I=lfn_3 Input file name if directives are on a separate file; file must be physical (no minus page) in ASCII format with 1F line terminators.

Format for DUMP control cards for use with a batch processor:

DUMP(lfn,I=lfn_3)

lfn Name of source file or drop file.

I=lfn_3 Input file name.

DIRECTIVES

More than one set of directives may be entered interactively or through the input file. Sets are separated by a slash. Each set must have the following formats:

#fwa,length

Dumps in hexadecimal from the bit address specified for the length specified.

fwa First virtual bit address of area to be dumped in hexadecimal. First character of address must be # to distinguish it from other directives.

length Number of words (hexadecimal) in area starting at fwa. If not specified, dump will extend from virtual address 0 to fwa.

In the following directives, the underlined letters may be used as an abbreviated directive. Brackets indicate optional elements of a directive set; unless otherwise stated, one set of the elements in parentheses must be specified.

FILE

Returns status of opened files and the virtual address ranges of mapped in files.

MPAGE

Dumps a file minus page.

INTEGER[(fwa,length)]

Dumps an integer array in hexadecimal.

fwa First virtual bit address of area to be dumped.
Dump will be in hexadecimal and integer format (I10).

length Number of words (hexadecimal) in area starting at fwa.

FLOAT

Dumps a floating point array. Parameters are same as for INTEGER. Dump will be in hexadecimal and exponential (E16.10) format.

TRACE

Gives a subroutine traceback from the last executed address.

DISPLAY

For use with interactive processing only; allows user to display a specified set of information prior to dumping. All previously defined directives may be displayed by prefixing the directive with the letter D.

DREGISTER or DRE
DINTEGER or DI
DFLOAT or DFL
DTRACE or DTR
DFILE or DFI

ROLL

Displays the four lines which immediately follow the last display location.

BACK

Displays the four lines which precede the last display location.

END

Used to terminate DUMP.

ALTERNATE INPUT FILE FORMAT

If directives are not to come from the normal input stream, the file named(under I=lfn$_3$) must have the following characteristics:

1. No minus page.

2. File type may be physical or virtual.

3. File must have READ access.

4. Data must be in ASCII.

The directives must be structured thus:

directive set$_1$ $\begin{bmatrix}U\\S\end{bmatrix}$ directive set$_2$ $\begin{bmatrix}U\\S\end{bmatrix}$ . . . directive set$_i$ $\begin{bmatrix}U\\S\end{bmatrix}$ $\begin{bmatrix}R\\S\end{bmatrix}$

$\begin{bmatrix}U\\S\end{bmatrix}$ Is a unit separator (1F).

$\begin{bmatrix}R\\S\end{bmatrix}$ Is a record separator or end of data (1C).

Detection of the directive END on the alternate input file will also terminate DUMP's activities with this file.

MDUMP

MDUMP is a library object module callable by programs written in FORTRAN or META subroutines of a FORTRAN program. The module may be called as often as necessary, and will perform dumps of a specified mapped in area of virtual memory.

The dump is written to a file or files defined on the PROGRAM statement or in the execute line of a FORTRAN program. For example, if a call to MDUMP is made, indicating the dump is to be written to logical unit 3, then the PROGRAM statement or execute line must contain UNIT3=filename.

MDUMP may be called from META subroutines of a FORTRAN program if the call is written in the FORTRAN call format. The logical unit referenced in the call must be defined on the PROGRAM statement or execute line.

To use MDUMP, the user must include calls in his code. The module is linked to the user at load time. The call format is:

```
CALL MDUMP(fwa,len,opt,lun)
```

fwa — Starting bit address of area to be dumped.

len — Length in 64-bit words of area to be dumped.

opt — Optional:

O or unspecified	Hexadecimal dump
I	Integer dump
Ew.d Fw.d	Floating point dump, where w is the field width and d is the fractional decimal digit count

lun — Logical unit number of file to which dump is to be written. If lun=0, the dump will be written to the user's standard output file.

There are three output formats available through the opt parameter: hexadecimal, integer, and floating point.

DEBUG

The DEBUG program may be executed in either batch or interactive mode.

BATCH USE

The batch user must provide an input file (described in section 6) containing the DEBUG commands in ASCII. The user also must supply an output file where the DEBUG program will place the display information requested. The input and output files are specified on the DEBUG control card required by the Batch Processor. If the output file specified is named OUTPUT, the Batch Processor will print the file automatically; otherwise, the user must print the output file.

INTERACTIVE USE

Through the DEBUG commands, the user may: set or remove multiple breakpoints, display and modify user registers and virtual memory, dump registers and virtual memory to an output file, request execution to continue or stop, and reference user virtual memory by hexadecimal addresses.

The DEBUG program executes entirely within the user's virtual memory space. Since DEBUG uses virtual memory starting at hexadecimal virtual bit address 7FFF00000000 and extending upwards, the user program being debugged must not use or reference this area. Also, the user program being debugged must adhere to the register conventions assumed by STAR-OS software. (See appendix E)

The terminal user calls DEBUG into execution with the following request:

```
DEBUG(parameters)
```

After it is brought into execution, DEBUG will request file information from the user if it is not already provided on the EXECUTE line. When it is awaiting a command, DEBUG signals with a question mark. The DEBUG program remains in execution until an EXECUTE, STEP, or CONTINUE command causes DEBUG to relinquish control to the user program.

Control does not return to DEBUG until a user specified breakpoint occurs. If a breakpoint is not reached before the user program terminates, the DEBUG program is terminated also. To re-enter DEBUG, another STAR-OS EXECUTE line must be typed on the terminal.

The following parameters may appear in the DEBUG request:

(filename,IOC-number,optional-parameters)

filename — Name of file containing the executable user program to be debugged. The file is assumed to contain virtual code with first two pages containing the program's minus page and virtual page zero.

IOC-number — Available IOC number in hexadecimal which DEBUG can use to open the user program filename. The IOC cannot duplicate one used by the user program.

optional-parameters — I=ifn,IOC-number:

This parameter is for NON-TERMINAL USERS ONLY. ifn names the input file containing the user specified DEBUG commands. The filename must be followed by a comma and the IOC number in hexadecimal. For example: I=IN,3.

O=ofn,IOC-number:

ofn names the output file to be used for the SNAP command or BATCH output. The file name must be followed by a comma and the available IOC number in hexadecimal. The output file size must not exceed 14 (hexadecimal) blocks. For example: O=OUTPUT,5.

DEBUG COMMANDS

The general format of each DEBUG command is as follows; where the symbol ƀ appears, either a blank or a comma must appear.

command ƀ parameter set

The following conventions apply to the DEBUG command descriptions:

A command may not exceed one line on a terminal.

Any elements surrounded by brackets are optional.

Commas must delimit a skipped parameter if another parameter follows.

Defaults are assumed for optional or missing parameters on most commands. See each command definition.

All hexadecimal addresses are assumed to be absolute virtual addresses unless a routine name is specified; in that case, the hexadecimal address is relative to the load map address for that routine.

All location references can be followed immediately by a plus (+) or minus (-) word displacement hexadecimal number.

DEBUG commands are listed below. Underlines indicate the minimum characters that may be specified to call the command. Following this list, the commands are described in a logical grouping rather than in alphabetical order.

ASCII	Enter data in ASCII form.
BACK	Display the data preceding the last display location.
BKPT or BKPTR	Set or remove breakpoints.
CONTINUE	Continue execution from the last user breakpoint.
DDECIMAL	Display data in hexadecimal and decimal.
DECIMAL	Enter data in decimal form.
DFLOAT	Display data in hexadecimal and floating point.
DISPLAY	Display data in hexadecimal and ASCII.
DREG	Display register contents in hexadecimal.
END	Terminate execution of both DEBUG and user program.
EREG	Enter hexadecimal data into a register.
EXECUTE	Begin execution of user program at a specified location.
FLOAT	Enter data in floating point.
HEX	Enter data in hexadecimal.
IDISPLAY	Display the data contained at the address found at the specified location.
IDREG	Display the data found at the address specified in the given register.
ROLL	Display the data following the last display location.
SNAP	Dump to an output file.
STAT	Provide status information such as breakpoints set, last routine referenced, last command issued, etc.
STEP	Step through execution of user code one instruction at a time.

DISPLAY MEMORY COMMANDS

DISPLAYƀ[name=]location[ƀtype] [ƀnwords] Provides hexadecimal and ASCII display.

DDECIMALƀ[name=]location[ƀtype] [ƀnwords] Provides hexadecimal and decimal display.

DFLOATƀ[name=]location[ƀtype] [ƀnwords] Provides hexadecimal and floating point display.

IDISPLAYƀ[name=]location[ƀtype] [ƀnwords] Displays the data at the address given in the specified location. (Indirect display address.)

name= Name of a module within the file to which the location parameter refers. An equals sign must immediately follow the name and precede the location, in the form name=location.

location Hexadecimal address or plus or minus word displacement indicating location at which display is to originate.

type Single character defining type of address designated:

X	Hexadecimal bit address (default)	XH	For DFLOAT and DDECIMAL only. Indicates that data to be displayed is halfword data.
W	Hexadecimal word address	WH	
P	Hexadecimal page address	PH	

nwords Hexadecimal value designating the number of words to be displayed. Default value is 4; maximum allowed value is 10.

The following commands display virtual memory forward or backward from the last display command location:

ROLL[ƀnwords] Display area following last location

BACK[ƀnwords] Display area preceding last location

nwords Hexadecimal value designating the number of words to be displayed starting from last location displayed. Default value is 4; maximum allowed value is 10.

REGISTER COMMANDS

The user can display and alter the contents of the user program registers:

DREGƀ hex reg [ƀnregisters]	Provides a display of a register.
EREGƀ hex reg ƀ hex data	Allows user to enter hexadecimal data into a register.
IDREGƀ hex reg [ƀnwords]	Displays data found at address given in specified register.

hex reg — Full word hexadecimal register number which contains data to be displayed or into which data is to be entered.

nregisters — Specifies hexadecimal number of registers to be displayed, starting with hex reg. Default value is 4; maximum value allowed is 10.

hex data — Hexadecimal half-word data to be entered into n consecutive registers starting with hex reg. Values are right justified with zero fill.

nwords — Hexadecimal value designating the number of words to be displayed. Default value is 4; maximum value allowed is 10.

ALTER MEMORY COMMANDS

The user can alter virtual memory by entering:

HEXƀ [name=] location [ƀtype] ƀ halfhex	Enter hexadecimal data.
ASCIIƀ [name=] location [ƀtype] ƀ"ASCIIdata"	Enter an ASCII character string.
DECIMALƀ [name=] location [ƀtype] ƀ±decidata	Enter decimal data.
FLOATƀ [name=] location [ƀtype] ±flpt	Enter floating point data.

Parameter definitions:

name= — Name of a module within the file to which the location parameter refers. An equals sign must immediately follow name and precede the location, in the form name=location.

location — Hexadecimal starting address or plus or minus word displacement indicating where data is to be entered.

type — Single character defining type of address designated by location parameter:

X	Hexadecimal bit address (default)	XH	Indicates halfword data to be entered (DECIMAL and FLOAT only)
W	Hexadecimal word address	WH	
P	Hexadecimal page address	PH	

halfhex — Half-word hexadecimal data values to be entered into consecutive half-word memory locations starting at location specified. Values are right justified with zero fill.

"ASCIIdata" — String of ASCII data to be entered into consecutive character locations starting at the position given by location parameter. The ASCII data string must be enclosed in quotation marks.

±decidata — Full- or half-word decimal data to be entered into consecutive full- or half-word memory locations beginning at the location specified.

±fltpt — Floating point data to be entered into consecutive half- or full-word memory locations, depending on data type parameter, starting at location specified. E or F format may be used.

PROGRAM CONTROL COMMANDS

The user can set and remove breakpoints to start and stop program execution, to dump portions of virtual memory to an output file, and to find the status of DEBUG commands issued earlier.

BKPTƀ [name=] locationƀ [type] — Sets a breakpoint; user program execution stops after the instruction at the breakpoint location is executed.

BKPTRƀ [[name=] locationƀ [type]] — Removes breakpoints. If no parameters are given on the BKPTR command, all breakpoints set in the program are removed.

EXECUTEƀ [[name=] locationƀ [type]]

CONTINUE

STEPƀ [n instructions]

END — Terminates both the user program and the DEBUG program.

SNAPƀ { [name=] location [ƀtype] [ƀnwords] — Dumps contents of core.
SNAPƀ { hexregƀR [ƀnwords] — Dumps contents of register.

STAT

EXECUTE causes the DEBUG program to start executing the user program at the location specified. If no location is given, DEBUG starts at the transfer address.

CONTINUE causes user program execution to be continued from the last breakpoint encountered. If CONTINUE is given before EXECUTE, DEBUG starts at the transfer address.

STEP causes user program execution to be stepped through n number of instructions from the last breakpoint encountered. If no parameter is given, the default value is 1 instruction with a maximum of 10 hexadecimal instructions.

SNAP causes either a specified number of words starting from location in virtual memory or the contents of a specified number of registers to be dumped to an output file. If the word or register count is not specified, only four words or registers are dumped. Output data is in hexadecimal and ASCII.

STAT produces a list of the breakpoints set, the last DEBUG and BKPT command issued, the last routine name or common block referenced, and the next execution address in the user program.

name= Name of a module to which the location parameter refers. An equals sign must immediately follow the name and precede the location, in the form name=location.

location Hexadecimal address, used as a displacement at which DEBUG is to set or remove breakpoints, to start user program execution, or to start a SNAP dump of memory. A plus or minus word displacement may be given in this parameter.

type Single character defining the type of address designated by the start location.

X Hexadecimal bit address (default)

W Hexadecimal word address

P Hexadecimal page address

hexreg Register number in hexadecimal of the first register to be dumped. Number must be in the range of 0 to FF.

n instructions Hexadecimal number of instructions to be executed for the STEP command. Default value is 1; the maximum value allowed is 10.

nwords Hexadecimal number of words or registers to be dumped. Default value is 4.

Examples

DI 1 P 16

Display 16 words at page address 1 (hexadecimal bit address 8000).

DI SUBR=10000 W

Display 4 words (default taken) at hexadecimal word address 10000 in the module whose name is SUBR.

BKPT MAIN.=500+7

Set a breakpoint in module MAIN. seven words beyond bit address 500 (hexadecimal address 6C0).

H 54DE0 12345678 9ABCDEF0

Enter given hexadecimal data at hexadecimal bit address 54DE0.

UTILITIES 9

Several utility programs are provided with STAR-OS. They are public files that can be called by control cards (in batch processing mode), by various terminal entries, or by being utilized as a controllee by the user program.

The following utilities are available for use with STAR-OS:

CREATE	Create a disk file.
DESTROY	Destroy all or a subset of user private files.
GIVE	Give files to another user.
FILES	List a user's files, their lengths and access parameters.
COPY	Copy all or part of one disk file to another.
SWITCH	Change file names and/or access parameters.
COMPARE	Compare all or part of one disk file with another.
TCOPY	Copy files or records between tapes and disks.
EDITPUB	List and edit the public file chain.
OLE	Object library editor.
EDITT	Source file line editor.

CREATE

The user can create a disk file with control statements, rather than by structuring a system message. The general format is:

```
CREATE(filename,length,optional parameters)
```

filename — File name must be first parameter; it consists of 1 to 8 letters and numbers.

length — Length of file must be second parameter specified; stated as a number of 512-word blocks (small pages) needed to hold file.

Remaining parameters are optional, and may be specified in any order, separated by commas.

$$A=\begin{Bmatrix}R\\W\\X\\RW\\RX\\WX\\RWX\end{Bmatrix}$$

Access parameter:

R	Read access
W	Write access
X	Execute access

Any combination of no more than 6 characters may follow A=. Access is given for any R, W, or X character appearing at least once in the string. Default is RWX.

$$T=\begin{Bmatrix}V\\C\\P\end{Bmatrix}$$

File type:

V	Virtual data file (default)
C	Virtual coded file
P	Physical data file

A string of no more than 6 characters may follow T=. The first V, C, or P character in the string indicates the file type.

$$L=\begin{Bmatrix}R\\W\\X\\RW\\RX\\WX\\RWX\end{Bmatrix}$$

Lockout parameter. Indicates accesses for which the file may not be opened.

R	Read access
W	Write access
X	Execute access

Any combination of no more than 6 characters may follow the L=. Access is withheld for any R, W, or X character appearing at least once in the string. Default is no lockouts.

B=address — Base virtual bit address to be associated with this file, expressed as 1 to 12 hexadecimal digits. It is adjusted to a page boundary. If this option is omitted, the file is assigned virtual bit address 10000000.

S=number — Security level number 1-255 decimal. If this option is omitted, the file has the security level number found in the user's directory.

U=packid — Up to 6 ASCII characters (alphanumeric) indicate the pack to which the file is to be assigned. If the option is omitted, the file is assigned to first available system pack.

For terminal requests, the utility will write any error information or other output to the terminal. For batch requests, the utility will send all output to the dayfile.

The following output and error information will be returned to the user. Under normal conditions, the system will return the logical unit number on which the created file resides. If an error occurs, the system will return (depending on the circumstances):

1. The file name already exists.

2. No available mass storage space for this file.

3. A parameter or format error was found.

4. The file index is full.

DESTROY

DESTROY deletes one or more files from the user's private file index and releases the mass storage space. Three formats are available:

DESTROY(=ALL)

DESTROY(file list)

DESTROY(=LIST)

=ALL Destroy all of the user's private files.

file list User supplies list of one to sixteen file names of files to be destroyed, e.g. (lfn_1,lfn_2,lfn_3).

=LIST For terminal entries only. The utility will present the file names one at a time. If the file named is to be destroyed, the terminal user replies with a D. If the file is not to be destroyed, he hits the NEW-LINE key. When the user types in STOP, the routine will stop deleting files and terminate normally.

For terminal requests, the routine will write any output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

Under normal circumstances, the following output information will be displayed to the user:

1. The names of files not found if the user utilizes the "file list" parameter format.

2. A list of the files destroyed. If a parameter or format error was found, a message will be displayed indicating that no files were destroyed.

3. The names of files not destroyed.

GIVE

This utility allows a user to give one or more of his private and inactive files to another user or to a pool. The general format is:

$$\text{GIVE}\left(\text{=ALL, }\begin{Bmatrix}\text{U=number}\\ \text{P=name}\end{Bmatrix}\right)$$

$$\text{GIVE}\left(\text{file list, }\begin{Bmatrix}\text{U=number}\\ \text{P=name}\end{Bmatrix}\right)$$

=ALL Give all of a user's private and inactive files to another user/pool.

U=user number User number to receive files.

P=pool name Pool to receive files.

file list List of one to sixteen file names, separated by commas, to be given.

For terminal requests, the utility will write any output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

Normal output will be the name of files given. If there were files which could not be given, their names will be displayed, grouped as indicated below:

1. The names of the files for which the same name already exists in the receiver's file index.
2. The file names which are the same as a public file name.
3. The names of files which do not exist to give to another user.
4. A list of files for which no such user number or pool name exists.
5. A list of output files which are improperly named. (Only meaningful if receiver user number = 999999. Files can be given to user 999999 for output processing, but must be properly named.)
6. The names of files which are still active, thus not given.
7. The user numbers which belong to a public file list.
8. Illegal pool names.
9. The names of files that the receiver could not access, thus not given.

If a parameter or format error was found, the error message will be displayed as appropriate.

FILES

FILES lists all or a subset of a user's private or public disk file names, the length of each file in small pages, and the access parameters of each. Additional output information can be requested by options. Three formats are available:

FILES(=PRI)

FILES(=PUB)

FILES(file list,=O)

=PRI All the user's private files are listed with the lengths and access parameters.

=PUB All the public files are listed with length and access parameters.

file list User supplies a list of file names, separated by commas. Length and access parameters and indication as to whether the file is private or public are returned as output. Other information may be obtained with the output option.

=O Output option for private files. If option is specified with =PRI, this request will be ignored. If option is specified with a file list, the utility lists each file's management category (private, scratch, output, or write temporary), the logical unit number on which the file resides, its file type (physical data, virtual data, or virtual code), and the file's sector address.

=OP Output option for public files.

For terminal requests, the utility will write any output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

COPY

The COPY utility reproduces all or part of one disk file onto another. The format is:

COPY(infile,outfile,L=length,I=inadr,O=outadr)

infile The name of the input file which is to be copied.

outfile Name of the output file which will contain a copy of all or part of the input file. The utility will create this file, if it does not exist, as a private file with the same type and access parameters as the infile and with length equal to L, if specified, or the length of infile.

L=length Number of words to be copied. If this parameter is omitted, the entire input file will be copied, starting at inadr, if specified, or the beginning of infile.

I=inadr The first input word address which is to be copied. The address specified must be relative to the beginning of the file, where the beginning address of the file is zero. If this parameter is omitted, the copy will be from the beginning of the input file.

O=outadr — The beginning output word address where the copied file is to begin. The address specified must be relative to the beginning of the outfile, where the beginning word address of the file is zero. If this parameter is omitted, infile will be copied starting at the first word of outfile.

Note:

If the files to be compared are virtual files, the first 512 words of each file will be its minus page; if the files are virtual code files, the second 512 words will be virtual page zero for the file. Unless these are to be copied, I and O should be specified to reflect their omission; e.g., I=400, O=400.

For terminal requests, the utility will write any output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

The following error messages will be returned, as appropriate:

1. A parameter or format error was found.
2. The infile does not exist or cannot be opened.
3. Outfile does not have write access.

COMPARE

The COMPARE utility compares all or part of one disk file with another, word by word, to determine if they are identical. The user may specify that a certain number of non-matching words, and their relative location in the files, be displayed. The format is:

```
COMPARE(file1,file2,L=number,A=adr1,B=adr2,N=number)
```

file1,file2 — Names of files to be compared.

L=number — Number of words to be compared. If omitted, the length of file1 will be used.

A=adr1 — Relative word address in file1 from which to start the compare. The address must be relative to the beginning of file1. If omitted, the first word of file1 will be used.

B=adr2 — Relative word address in file2 from which to start the compare. The address must be relative to the beginning of file2. If omitted, the first word of file2 will be used.

Note:

If the files to be compared are virtual files, the first 512 words of each file will be its minus page; if the files are virtual code files, the second 512 words will be virtual page zero for the file. Unless these are to be compared, A and B should be specified to reflect their omission; e.g., A=400, B=400.

N=number Number of non-matching words (decimal) which will be displayed, along with their relative locations. Comparison of files will cease once the N= limit has been met. If omitted, the default value for N is 1.

For terminal requests, the utility will write output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

The following error information will be returned, as appropriate:

1. File1 does not exist or cannot be opened or accessed.
2. File2 does not exist or cannot be opened or accessed.
3. Parameter or format error was found.
4. The non-matching words and their locations in the file that did not match.

SWITCH

SWITCH can be used to change a private file's name, type, and/or access parameters. SWITCH can also be used to change the length of the drop file created when the file is executed. The format is:

SWITCH(oldname,newname,T=type,A=access,L=lockout,D=dropfile length)

oldname File name as file is currently known.

newname Optional; new file name. Maximum of eight letters and numbers. This parameter may be omitted if only the oldname file's access and/or type parameters are to be changed. When newname parameter is omitted, the comma following newname parameter need not appear.

$T = \left\{\begin{matrix} V \\ C \\ P \end{matrix}\right\}$ Optional,new file type:
virtual data file
virtual code file
physical file

A =		Optional; new access parameter:
	R	read access
	W	write access
	X	Execute access
	RW	read/write access
	RX	read/execute access
	WX	write/execute access
	RWX	read/write/execute access

L =		Optional; lockout parameter:
	R	read lockout protection
	W	write lockout protection
	X	execute lockout protection
	RW	read/write lockout protection
	RX	read/execute lockout protection
	WX	write/execute lockout protection
	RWX	read/write/execute lockout protection

D=value — Optional; drop file length parameter.
Value is a decimal number.

If T, A, L, or D are omitted, the file will maintain its current, drop file, and access parameters.

If T- is specified when L= is not, the FILEI table is changed to reflect no-lockout protection. If L= is specified when T= is not, the FILEI table is changed to reflect no-access permission. When both A= and L= are specified, access for R, W, and/or X is set in the FILEI table only if it is specified on the A= parameter and not on the L= parameter.

A string of 1 to 6 characters may follow A= and L= parameters. R, W, and/or X are considered to have been specified if they appear at least once in the string; other characters in the string are ignored.

Also, a string of 1 to 6 characters may follow the T= parameter. The file type is determined by the last V, C, or P character in the string; other characters are ignored.

When the D option is specified, the new drop file length is transmitted to the FILEI table and a message is issued to indicate that the change was made. When the file is subsequently executed, the operating system will create a dropfile of the length specified, or of the length of the source file, whichever is greater.

If the new file name has the proper format but there are errors in the other parameters, only the file name is changed. A message is issued to indicate the change made.

For terminal requests, the utility will write output or error information to the terminal. For batch requests, the utility will send all output to the dayfile.

The following error messages are returned as appropriate:

1. A parameter or format error occurred.
2. A parameter or format error occurred but the name was changed.
3. The oldname file does not exist.
4. The newname file already exists.

TCOPY

The TCOPY program copies files or records from tape to tape, tape to disk, disk to tape, and disk to disk. Binary, BCD, or ASCII tapes may be copied; multi-file tapes may be created. TCOPY also provides the capability to position tapes before and after a copy. TCOPY may be called to execution from an interactive terminal or from the batch processor.

BATCH CONTROL CARD FORMAT

The control card for calling TCOPY in the batch system is written:

```
TCOPY(lfn).
```

lfn — The name of a disk file containing the directives for TCOPY.

EXECUTE LINE FORMAT

TCOPY is called into execution from an interactive terminal by an execute line having the format:

```
TCOPY / t I /
```

Blanks must appear between each part of the execute line.

t — Execution time in number of seconds.

I — Flag indicating call is from interactive terminal

TCOPY DIRECTIVES

TCOPY directives are listed below; each directive must be completely spelled out:

BKSPF
BKSPR
CLOSE
COPYF
COPYR
END
OPEN
REWIND
SKIPF
SKIPR
WEOF

OPEN lfn,type,parameters

lfn — Logical file name

type — Storage device to be used.

DISC — disk file (default)
TAPE — tape file

A maximum of 16 files may be open at one time, when TCOPY is run interactively. Only 14 files may be open if TCOPY is run as a batch job. The remaining parameters for OPEN pertain to disk or tape as follows:

OPEN lfn,DISC,length,packid

Open the named disk file; if it does not exist, it is created.

length
Hexadecimal length of disk file in number of small pages; default is 200. This field should be omitted if file already exists.

packid
Identifier of disk pack where file is to be created. If omitted, the system selects an available unit.

OPEN lfn,TAPE,tracks,mode,density,blocksize,vrn

Open the named tape file. The actual file supplied is dependent upon the operator mounting the correct tape. All tape files are assumed to be unlabeled. File names for tape are limited to 5 characters.

tracks
Indicates 7- or 9-track tape units.

7	7-track
9	9-track (default)

mode
Defines tape recording mode.

ASCII	7- or 9-track tape
BCD	7-track tape only
BIN	Binary 7- or 9-track tape (default)

density
Recording density

2	200 bpi	(7-track)
5	556 bpi	(7-track)
8	800 bpi	(7- or 9-track)
16	1600 bpi	(9-track)

Default is no density change from current setting.

blocksize
Number of hexadecimal characters per physical tape record. Maximum (and default) is 7FF0. V may be specified for variable length ASCII or BCD records.

vrn
Optional visual reel number.

Once a file has been opened, the following directives may be issued for the file. In all cases, lfn is the logical file name of the file opened by the OPEN directive. The comma following the command is not required.

REWIND,lfn
Rewinds a tape file to its load point; sets disk file copy pointer to block 1.

SKIPR,lfn,n
Skips forward n physical records on that tape file; sets copy pointer forward n unit separators; default n = 200.

SKIPF,lfn,n — Skips forward over n end-of-file marks, leaving the tape positioned after end-of-file mark n; positions copy pointer forward n file separators. End position is after the nth file separator. Default n = 200.

BKSPR,lfn,n — Backspaces a tape file over n physical records; positions copy pointer back n unit separators. Default n = 200.

BKSPF,lfn,n — Backspaces a tape over n end-of-file marks, leaving the tape positioned immediately before the last end-of-file mark on tape; positions copy pointer back n file separators. Default n = 200. End position is before nth file separator.

WEOF,lfn — Writes an end-of-file mark on tape; places file separator on disk file.

CLOSE,lfn,RUN — Closes a file. The input/output connector is released. If RUN is specified, unit is rewound and unloaded. Tape mark or file separator is written for output file.

To perform the copy operation, either of the following directives may be used:

COPYR,from lfn, to lfn,n

> Copies n records from lfn to lfn; a tape record is considered to be a physical record. File separators may terminate the copy but will not be written (see TCOPY results).

COPYF,from lfn, to lfn

> Copies data from lfn to lfn; a tape file is considered to be all data between two end-of-file marks. Either or both lfn's may be tape or disk. A disk file is all data between two file separators.

A group of directives is terminated by the following:

END — Terminates TCOPY. If tape files have been written, and the last tape operation was a write, an end-of-file mark is written.

TCOPY RESULTS

When records and files are copied from tape to tape, tape to disk, or disk to tape, TCOPY restructures or converts their format according to the source or destination codes and recording device. The following notes are appropriate to the various formats copied. Implied in these notes is the code conversion that occurs when copying is done between ASCII and BCD tapes.

DISK TO BINARY TAPE

The disk file is formatted to the tape block size and written to tape. The disk data is not scanned; space allocated to disk file is assumed to define the file size.

No file separators are written on the output tape file. The last tape record is padded with a half-word bit pattern corresponding to hexadecimal 000C1F1C out to the block size.

Both COPYR and COPYF terminate when a file separator is detected.

COPYR creates n tape records.

COPYR and COPYF return an error if the tape file was opened with a variable block size.

DISK TO BCD OR ASCII TAPE

The disk file is scanned for unit separators that define the size of the variable-length tape record to be written. The fixed block size with which the tape file is opened defines the upper limit of the record sizes. Records smaller than four bytes, or larger than the tape block size, terminate the copy operation with an error.

Unit separators on the disk file are not copied to the tape file.

A file separator on the disk (input) file terminates a COPYR and causes an end-of-file mark to be written on the tape file.

The character pair consisting of ESC plus a numeral (1Bn) is expanded to place (n-hexadecimal 30) blank characters in the output record.

BINARY TAPE TO DISK

Tape blocks are concatenated into the disk file. When the end of the space allocated to the disk file is encountered, the copy operation is terminated.

COPYR is terminated when n tape records are copied or when a tape end-of-file mark is sensed; the tape mark is not copied to the disk file. The remainder of the disk block is padded with the half-word hexadecimal pattern 000C1F1C when the copy is terminated; however, the pattern is overlaid if more data is copied to the file.

End-of-file marks are not transferred to the disk file by COPYF. The last disk block used in a COPYF is padded with the half-word hexadecimal pattern 000C1F1C, which is overlaid if more data is copied into the file.

BCD OR ASCII TAPE TO DISK

Each tape record copied to the disk file is appended with a unit separator. Tape records are concatenated to the disk file. When two or more blanks are encountered on the tape file, they are compressed to the form represented by the ESC character plus a numeric value (1Bn) where n represents the number of blanks compressed plus hexadecimal 30. The detection of an end-of-file mark on tape terminates COPYR or COPYF. The last disk record into which data is copied is padded as described in Binary Tape to Disk.

BINARY TAPE TO BINARY, BCD, OR ASCII TAPE

In principle, this action is the same as described for Disk to Binary Tape and Disk to BCD or ASCII tape; also, an end-of-file mark on the input tape will terminate the copy.

BCD OR ASCII TAPE TO BINARY TAPE

Each input tape record is appended with an end-of-line (unit) separator (1F) and written to the output file. Records are concatenated to the output file; variable block size is not allowed on binary output tapes. Detection of an end-of-file mark places a file separator on the binary output file. Two or more blanks are compressed to the format 1Bn where n is the number of blanks compressed plus hexadecimal 30. Detection of an end-of-file mark on the input tape or an end-of-tape mark on either the input or output tape will terminate the copy operation. The last output record is padded as in Binary Tape to Disk.

ASCII TO ASCII/BCD TO BCD TAPE

A tape copy is performed without translation or conversion.

TCOPY EXAMPLES

These examples merely show the appropriate directives that may be used in copying files; information related to placing the directives into either an interactive or batch job are found at the beginning of this description of TCOPY.

1. Copy from a 7-track ASCII tape to a 9-track binary tape. Input records are 80 characters long; output records are 4096 characters long.

```
OPEN    FILEA,TAPE,7,ASCII,8,50
OPEN    FILEB,TAPE,,,,1000
COPYF   FILEA,FILEB
CLOSE   FILEA,RUN
CLOSE   FILEB,RUN
END
```

2. Merge two existing disk files to form a single binary tape file.

```
OPEN    DISK1
OPEN    DISK2
OPEN    TFILE,TAPE
COPYF   DISK1,TFILE
COPYF   DISK2,TFILE
CLOSE   TFILE
CLOSE   DISK1
CLOSE   DISK2
END
```

3. Create a multi-file tape containing two files copied from disk.

```
OPEN    DISK1
OPEN    DISK2
OPEN    TFILE,TAPE,9,BIN,16,1000
COPYF   DISK1,TFILE
WEOF    TFILE
COPYF   DISK2,TFILE
CLOSE   TFILE,RUN
END
```

4. Copy the second file from a binary tape to a new disk file that is 30 hexadecimal pages long.

```
OPEN    TFILE,TAPE
OPEN    DISKF,DISK,30
SKIPF   TFILE,1
COPYF   TFILE,DISKF
CLOSE   DISKF
CLOSE   TFILE
END
```

EDITPUB

Authorized users can list existing public file names and add or delete file names from the public file list.

```
EDITPUB(L,D=file-list,N=file-list)
```

L List public files option

D Delete public files option

N New public files option

file-list User supplied list of file names, separated by commas; used as parameter list for D and N options. When files are added, the new files must exist as private files that are to be changed to public status.

For terminal requests, the utility writes all EDITPUB output and/or error information to the terminal. Under the L option, the utility presents the public file names one at a time. If the file name is to be destroyed, the terminal user replies with a D. If the file is not to be destroyed, the NEW-LINE key is the proper response. When the user types END, the routine stops deleting files and terminates normally.

For batch requests, the utility creates a file named OUTPUT on which all EDITPUB output and/or error information is written. Upon termination of the utility, the batch processor closes the EDITPUB generated output and gives it to USER1 for subsequent printing. Normal EDITPUB output information includes public file names, length of files in number of small pages, and file access parameters.

OBJECT LIBRARY EDITOR (OLE)

The STAR loader uses library files as a source of modules to complete unsatisfied externals. OLE is used to create and modify such libraries. OLE can only be used to create private library files..

There are two basic functions performed by OLE: one is to create or expand a library file; the other is to delete modules from an existing library file.

The general format of the control statement for the Object Library Editor is:

```
OLE(libname,len,options)
```

The maximum length of an OLE control statement is 4096 characters. The required parameters are:

libname — Name of the library file being created or modified

len — Length of the library file specified by user; if omitted, OLE will set the length.

The optional parameters are:

UPI=lfn$_i$ — Names of non-library formatted (User Private or pool Input) files to be used in library file creation or extension. These are object modules that are output by FORTRAN or META.

LIBRARY=liblfn$_i$ — Names of library formatted files to be used in creating a new library or extending an old one.

OMIT=lfn,mod$_i$ — The object modules specified for mod$_i$ are not copied from file lfn and therefore are omitted from the library file being created or modified. The file name lfn may be any of the files specified by the LIBRARY= or UPI= parameters.

LIST=DIR — List library header table, all object module names with their length, and all entry points in the file directory.

LIST=lfn — List all object module names and their length, along with all entry point names associated with the logical file name given.

For terminal requests, OLE writes all output and error information to the terminal. For batch requests, OLE creates for the user a file named OUTPUT which has the file type output. All output and error information is written into it. If the user has already created a file named OUTPUT, it will be destroyed.

Acceptable optional parameter abbreviations are:

UPI=	OMIT=
LIBRARY=	O=
LIB=	LIST=

CREATING OR EXTENDING A LIBRARY FILE

SYSTEM1 is the name of a file containing object modules output by any of the language processors. The statement:

```
OLE(SYSLIB,UPI=SYSTEM1)
```

creates a library file, SYSLIB, using the file SYSTEM1 as input.

SYSTEM2 is the name of a file containing object modules from any of the language processors. Then the statement:

```
OLE(SYSLIB,UPI=SYSTEM2,LIBRARY=SYSLIB)
```

adds to an existing library file SYSLIB the modules from the file SYSTEM2.

Note:

If duplicate module names are found during input, the following rules are used to resolve which module will be placed on the new library file:

1. If duplicate modules are found on any of the files associated with the UPI parameter, the first one encountered is used.

2. If duplicate modules are found on any of the library files, the first one encountered is used.

3. If duplicate module names are found on a library file and a file associated with the UPI parameter, the module from the Users Private or pool Input file is used.

OMITTING MODULES FROM A LIBRARY FILE

A user's private input file ABC is added to an existing library file LIBA. The module named MOD1, which resides on file ABC, is not added to the new library file LIBA. The following control statement performs this operation:

```
OLE(LIBA,UPI=ABC,LIBRARY=LIBA,OMIT=ABC,MOD1)
```

The default method of omitting object modules is the duplicate module name heirarchy described under CREATING OR EXTENDING A LIBRARY FILE.

A library file named SYSLIB is created from input library files LIB1 and LIB2. Modules named SYS3 and SYS4 which exist on library file LIB2 are omitted from the new library file. This is performed by the following statement:

```
    OLE(SYSLIB,LIBRARY=LIB1,LIB2,OMIT=LIB2,SYS3,SYS4)
```

OBJECT LIBRARY FILE

LIBRARY HEADER TABLE

The header table is a fixed length table consisting of six 64-bit words in the following format:

<table>
<tr><td>0</td><td>31</td><td>63</td></tr>
<tr><td colspan="3">LIBRARY</td></tr>
<tr><td colspan="3">libname</td></tr>
<tr><td colspan="3">ASCII clock value</td></tr>
<tr><td colspan="3">calendar value</td></tr>
<tr><td colspan="2">dir len</td><td>lib len</td></tr>
<tr><td colspan="2">mod#</td><td>entry#</td></tr>
</table>

libname — Name given to a new or newly modified library; name must be different from any of the file names associated with the user's private or pool input file parameter. Name must be 1-8 alphanumeric characters, left justified with blank fill.

ASCII clock value — Hours, minutes and seconds expressed as hh:mm:ss

calendar value — Month, day, and year expressed as mm/dd/yy

dir len — Word length in hexadecimal of directory table

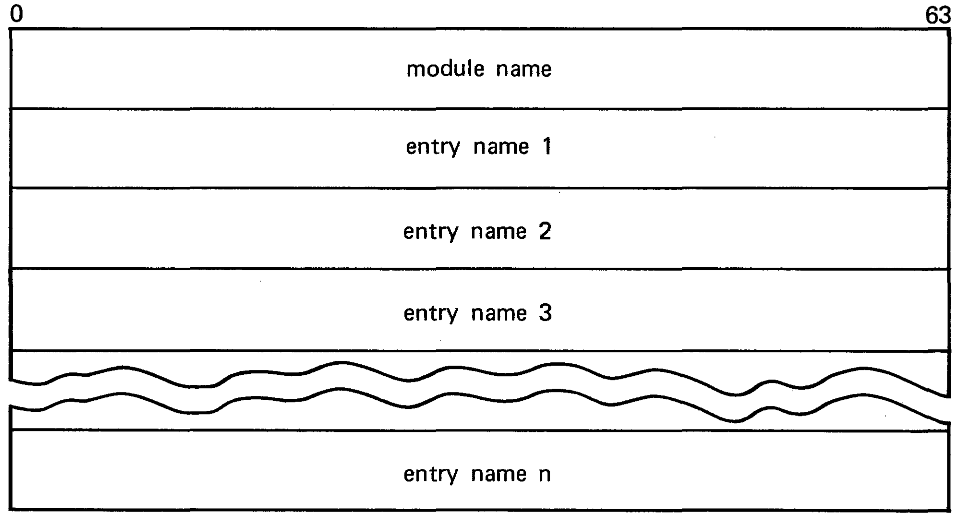
lib len — Total word length, in hexadecimal, of library

mod# — Hexadecimal number of modules in library

entry# — Hexadecimal number of entry points in directory

LIBRARY DIRECTORY

The directory is a variable length table consisting of 64-bit words in the following format:



module name — ASCII name of module in 1-8 alphanumeric characters, left justified with blank fill

entry name — External entry names associated with module name

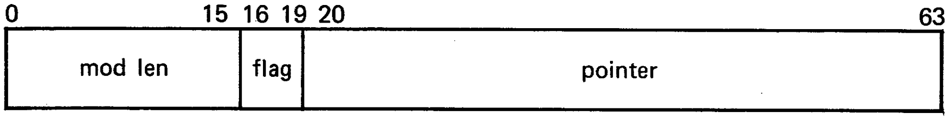
OBJECT LIBRARY FILE FORMAT

0 15 19 31 63

<table>
<tr><td colspan="4">L I B R A R Y</td></tr>
<tr><td colspan="4">libname</td></tr>
<tr><td colspan="4">ASCII clock value</td></tr>
<tr><td colspan="4">calendar value</td></tr>
<tr><td colspan="3">dir len</td><td>lib len</td></tr>
<tr><td colspan="3"># mods</td><td># entries</td></tr>
<tr><td colspan="4">module name n</td></tr>
<tr><td colspan="4">entry point name</td></tr>
<tr><td colspan="4">entry point name</td></tr>
<tr><td colspan="4">entry point name</td></tr>
<tr><td colspan="4">entry point name</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>mod len</td><td>flag</td><td colspan="2">pointer</td></tr>
<tr><td>mod len</td><td>flag</td><td colspan="2">pointer</td></tr>
<tr><td>mod len</td><td>flag</td><td colspan="2">pointer</td></tr>
<tr><td>mod len</td><td>flag</td><td colspan="2">pointer</td></tr>
<tr><td>mod len</td><td>flag</td><td colspan="2">pointer</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="4">M O D U L E</td></tr>
<tr><td>len</td><td colspan="3">0 —————————————— 0</td></tr>
<tr><td colspan="4">module name</td></tr>
<tr><td colspan="4">date & time created</td></tr>
<tr><td>T len</td><td colspan="3">processor</td></tr>
<tr><td>C len</td><td colspan="3">data base len</td></tr>
<tr><td>Type</td><td colspan="3">pointer</td></tr>
<tr><td colspan="4">Tables</td></tr>
</table>

INDEX TABLE

The index table is variable in length and contains 64-bit word pointers to the location of modules in the library. The word format is:



mod len — Hexadecimal word length of object module

flag — If flag is zero, directory entry corresponds to an object module name; if one, director entry corresponds to an entry point name in the module.

pointer — The pointer contains the word address, relative to the first word of the library header table, of the the object module.

MODULE TABLE

The module table contains the object modules referenced in the directory. The module table contains the module header table and all other tables associated with the module named in the directory.

EDITT

The user can edit source files interactively from a display terminal. The editing program is called into execution from a terminal with the command:

EDITT(olda,oldb,newa,newb)

olda — Required input file name, 1 to 8 characters, denotes source file to be edited.

oldb — Optional input file name. If only one input file name is given in command, a comma must appear in lieu of this file name parameter.

newa — Required output file name, 1 to 8 characters.

newb — Optional output file name.

The following forms of the EDITT command may be used:

EDITT(olda, , newa)	One input and one output file
EDITT(olda,oldb,newa)	Two input and one output files

EDITT(olda, , newa,newb)	One input and two output files
EDITT(olda,oldb,newa,newb)	Two input and two output files

All files names in the EDITT command must have been created or established by the user before EDITT is called. Each designated output file must be large enough to hold the edited version of the file. A maximum of two input and two output files provides the capability to combine two input files into one output file, expand one input file into two output files, rearrange sections of data, exchanging them with selected sections on a companion input file, or edit two files at once, making insertions and deletions in the process.

FILE PAGES

The user can divide each file into entities called pages. This term is not to be confused with the term page which has a definite meaning in both the STAR hardware as well as the operating system and other software. When a file is divided into pages, it is easier for the user to position a file to a particular location through skip or copy operations. A command provided in EDITT allows the user to insert the ASCII form feed character in the output file, thus denoting the end of a page. The number of pages in a file is not limited, provided the file is large enough. A file is considered to be contained in one page if no page-end marks are inserted. Also, no maximum is set on the number of lines that may be contained in a page.

EDITT keeps track of its position in any file by counting pages and lines. The line count can start at the beginning of the file or at the beginning of a page. EDITT assigns chronological page and line numbers to the input file; when the form feed character is encountered EDITT advances the page index and re-establishes the line count. The current position of the input file is displayed on the screen as: page-number/line-number.

MODES OF OPERATION

EDITT operates in two modes: insert and command. Insert mode allows lines of character information to be inserted at designated locations in the files. Command mode deals with the selection of files, positioning files, and other operations such as copying and preserving the contents of files.

INSERT MODE

EDITT is an interactive editor; any line entered interactively from the user terminal that does not begin with the @ character (at-the-rate-of) is interpreted as an insertion line and placed into the currently active new file. The manner in which a file is made currently active is discussed under COMMAND MODE, which follows. If a line to be inserted begins with the @ character, it must be preceded by an additional @ character. For example, @@ will insert @, @@@ will insert @@, and so forth. Although @ precedes all EDITT commands, the editor interprets the appearance of two or more @@ characters in succession to be an insertion line from which it strips off the first @ and inserts the remainder into the new file.

Up to 63 characters may be inserted or replaced at any one time. When an insertion line contains more than 63 characters, the ASCII null character, hexadecimal 00, must be the 63rd character in the line. Such a line, then, is considered to be part of a character string that is continued on the next insertion line. The next line entered is concatenated to the preceding line, and the null character is deleted in the process. In this manner, long lines of information may be inserted into the new file.

COMMAND MODE

Commands entered by the user from the interactive terminal have the following form:

```
@command value1 value2
```

command — A single character identifying the command.

value1
value2 — Use of these decimal values is determined by the individual commands. A comma or blank separates value1 from value2; however, only a blank may be used to separate value1 from the command.

FILE SELECTION COMMANDS

When EDITT is entered, files designated for olda and newa are selected for processing. When the first file selection (toggle) command is given, the alternate file, oldb or newb, is activated. The next toggle command will reactivate olda or newa, as determined by the specific command.

@T1 — Toggle the selection of old file.

@T2 — Toggle the selection of new file.

@T3 — Toggle the selections of both old and new files.

REPOSITION COMMANDS

These commands affect the line count.

@R1 — Move contents of new file to old file and reset old file line count to point to the first line in old file.

@R2 — Reset old file line count to point to the first line in old file.

@S1 — Reset new file line count to first line in new file.

BACKUP COMMANDS

The line count is backed up the number of lines specified in the commands.

@< — Back up old file line count by one line. (Default value)

@<1 n — Back up old file by n lines.

@<2 n — Back up new file by n lines.

@<3 n — Back up both old and new file line count by n lines.

BEGIN NEW PAGE COMMAND

@\ — Insert a line containing the ASCII form feed character into the currently active new file. This hexadecimal character OC denotes the end of a page and causes the new file line count to be set to the first line of the next page. The printer recognizes the form feed character as a page eject command.

DELETE COMMANDS

These commands cause the deletion of one or more lines from the new file. They cause the old file line count to be repositioned, effectively skipping over the lines and thereby omitting them from the new file.

@– n
@D n — Omit n lines of the old file from the new file. If n is not present in the command, only one line is skipped in the old file and thereby omitted from the new file.

@– pn,ln — Skip all lines from the current position in the old file up to the designated page number, pn, and line number, ln, thereby omitting these lines from the new file.

COPY COMMANDS

These commands always copy from the old file to the new file.

@+ — Copy from current location of old file to new file until an ASCII file separator, hexadecimal 1C, is encountered.

@+ pn,ln — Copy from old file to new file up to (but not including) the designated page and line number. If ln is omitted, copy up to (but not including) line 1 of the designated page.

@C n — Copy n lines from old file to new file. If n is omitted, copy one line only.

PRESERVE COMMAND

With this command, the user can save periodically the corrections made during a long edit session. These corrections are contained in the currently active new file being manipulated in core, and the current new file in core is written out to the disk to which the file is assigned.

@P — Write the currently active new file to disk.

EXIT COMMAND

@E — Exit from EDITT program. At this time, the new files are written from core to the assigned file device.

LIBRARY PREPARATION AND MAINTENANCE 10

UPDATE is a system utility program for creating, maintaining, and manipulating a user's library of program and data files. It also is used to maintain a program library containing source decks for the operating system. The STAR-100 UPDATE is a subset of the UPDATE used on the CDC 6000 and 7000 computer systems.

In using UPDATE, the user initially transfers a collection of source decks to a file known as a program library. Each card of each deck is assigned a unique identifier when it is placed on the library, so the card can be referenced during an UPDATE correction run. During correction runs, cards are inserted into or deleted from the program library according to sequence identification. The card image, even though deleted, is maintained permanently on the program library with its current status (active or inactive) and a chronological history of modifications to the status. A card listed as currently inactive has been deleted and is, in effect, removed from the deck. A card with active status is in the deck; either it has never been deleted or it has been restored after a deletion. During a single UPDATE correction run, a card may undergo one or more modifications, or no modification.

Up to six types of physical files are processed in an UPDATE run.

Input file containing UPDATE directives and corrective text (INPUT)

Old program library to be updated (OLDPL)

New program library to be created by this UPDATE (NEWPL)

Listable output files (OUTPUT)

Output file created for a compiler or assembler (COMPILE)

Output file containing all active cards on the program library (SOURCE)

UPDATE requires that all files exist on disk and, if specified by the user, creates files NEWPL, OUTPUT, COMPILE and SOURCE. For users requesting tape maintenance, UPDATE requires the user to perform utilities involving tape to disk and disk to tape operations. For example, the INPUT and OLDPL files must have been transferred to disk with a tape to disk utility prior to the execution of UPDATE. Subsequently, when UPDATE completes its function, the user must transfer the UPDATE output files NEWPL, OUTPUT, COMPILE, and SOURCE, if specified, from disk to tape.

The compile file is the primary output of an UPDATE run and contains only the active cards requested by the user. In a typical application, a user calls UPDATE to modify a FORTRAN source language deck maintained on an UPDATE program library, requests that the modified decks be written in source language format to the compile file, and then calls the FORTRAN compiler to read source input from the compile file.

A second type of output is a new program library. It contains updated decks requested by the user in program library format for use as an old program library in subsequent UPDATE runs. Normally, a new program library is output during an UPDATE creation run. It may become an old program library on subsequent correction runs. OLDPL is the logical file name used most often to refer to a standing program library. Whenever a NEWPL is required, UPDATE creates file TEMNEWPL, to be used for intermediate processing. If TEMNEWPL exists, UPDATE will utilize the existing file; otherwise, UPDATE will create TEMNEWPL with a length of 100 (hexadecimal) pages.

INPUT FILE

The input file is required and contains UPDATE directives, source decks, and/or corrective text (INPUT). UPDATE assumes that the input file contains 80-column card images in either compressed or uncompressed format. UPDATE determines the specific operations it is to perform from two sources: parameters on the UPDATE control card, and directives in the input file. In a directive card image, column 1 contains the special character specified as the master control character for this UPDATE run; the default character * is used most frequently. The control character is followed by a string of characters terminated by a blank or comma. This character string must be recognized by UPDATE as a valid directive.

As each card image is read from an UPDATE input stream, it is assigned a unique card identifier consisting of two parts. The first is an ident, established for the correction set with an *IDENT directive. (Under certain conditions, it may be established by a *DECK or *COMDECK directive.) The second part is a sequence number within that ident. The card identifier, then, is composed of ident.seqnum. An established identifier is retained along with the card image on the old program library (OLDPL). UPDATE directives may reference the card image by its identifier; also by referencing one ident, all sequence numbers under that ident may be referenced.

Some UPDATE directives exist only in the input stream; other directives are assigned identifiers and are placed on the OLDPL. Since these directives possess identifiers, they may be referenced thereafter in the same manner as text card images. Card images exist on the OLDPL in an active or inactive state. For text card images, the state is important only for COMPILE file output. Only currently active card images are written to the COMPILE file. For UPDATE directives on the library, the state has more meaning. Normally, these directives are processed when the COMPILE file is generated; they are not written to it, but specify certain operations that determine file contents. Inactive directives are ignored.

UPDATE CONTROL PARAMETERS

With UPDATE control parameters, the user specifies the correction mode of UPDATE, the format of the UPDATE directives, the existence and names of files involved, the types and formats of output from the UPDATE process.

Three UPDATE correction modes may be applied to a program library:

In FULL mode, all decks on the program library are processed.

In SELECTIVE mode, only decks specified or modified are processed.

In QUICK mode, only specified decks are processed.

The user can specify the character that begins an UPDATE directive card and the character that begins a comment.

The user may specify the names of each of the six types of files and whether the NEWPL, COMPILE, or SOURCE files are to be created.

For the OUTPUT file, the user may suppress default output. On the COMPILE file, he may select 80 columns of data to be preserved instead of 72, and he may select an 80-column format instead of 90.

UPDATE CONTROL CARD

This control card causes the UPDATE program to be loaded from the system library and executed. Parameters on the card specify modes and files for the run. The format is:

UPDATE(p1,p2, . . . ,pn)

A right parenthesis or a period terminates the control card. The parameters are optional and can be in any order. Generally, a parameter can be in one of the following forms or it can be omitted.

option

option=filename

option=filename,xxx (valid for C, N, O, S, T options, xxx= length of file in number of small pages. Default is 256)

option=0 (valid for modes C and L only)

The functions of the UPDATE control card parameters are outlined below.

UPDATE Parameter	Function
C	Compile file in deck list order
D	80 columns of data on compile file for compiler processing
F	Full UPDATE mode
I	Input file
L	Type of listing to be output
N	New program library
O	Output file
P	Old program library
Q	Quick UPDATE mode
S	Source file
T	Source file without common decks
8	80-column card images on compile file
*	Master control character
/	Comment character

Parameter Description

C Compile file output

omitted or C — Compile file output decks are written on file named COMPILE; contents are determined by the kind of update (F, Q, or normal).

C=filename — UPDATE writes compile file output decks on named file; contents are determined by kind of update (F, Q, or normal).

C=PUNCH — Compile file output decks are written on file named PUNCH; contents are determined by the kind of update (F, Q, or normal). This option also causes the D and 8 parameters to be selected.

C=0 — No compile file output

D Data width on compile file

omitted — Compile file output format is 72 columns of data for compiler processing

D — Compile file output format is 80 columns of data for compiler processing

F Full update

omitted — If the Q parameter is not specified, the omission of F designates the normal (selective) operation mode of UPDATE. All regular decks and common decks are processed. The new program library, if specified, contains all regular and common decks, after any corrections have been made in the sequence in which they occur on the old program library. The source file, if specified, contains all active cards with decks in deck list sequence. The compile file contains all decks corrected during this UPDATE run and all decks specified on COMPILE directives. Any deck that calls a corrected common deck is also considered to be corrected unless the common deck is a NOPROP deck (see COMDECK directive).

F — Source and compile files, if specified, contain all active decks in old program library sequence. The contents of the new program library are the same as if F were not specified.

I Input

omitted or I — Input is on job INPUT file

I=filename — Input comprises next record on named file

L List options

omitted List options A, 1, and 2 are automatically selected on a creation run. Options A, 1, 2, 3, and 4 are automatically selected on a correction run.

$L=c_1,c_2, . . ,c_n$ Each character in the string selects an option. The digit 0 in the string negates any other selections. For example, L=A12340 is equivalent to L=0. The options are as follows:

A Listing of error decks, correction set idnames, common decks, decks written on the compile file.

F All selections other than 0.

0 No listing.

1 Errors are listed if 1 is selected by default. If L=1 is specified, the deck name list, identifier list, and continuous commentary are suppressed.

2 Directives

Each IDENT directive produces a page eject. Each card recognized by UPDATE as a valid control card is marked by five asterisks to the left of the card.

3 Commentary on changes to cards processed, consisting of: name of deck, card image, card identifier with sequence number, and a key and action are shown below:

Key	Action
I	Card introduced during run
A	Inactive card reactivated
D	Active card deactivated
P	Card was purged during run

When currently active cards are purged, the word ACTIVE occurs in addition to the P action.

4 Input stream

Cards in error are marked by *ERROR* to the left and to the right. Cards resulting from a READ directive are marked to the right with the name of the file from which they were read.

Decks inserted by ADDFILE are not listed if list option 4 is selected by default; they are listed if option 4 is explicitly selected.

9 Listing of all active and inactive cards. List option 3 takes precedence over List option 9. Active (A) or Inactive (I) status is indicated to the right of each card image on the listing.

N New program library output

N New program library to be written on file NEWPL

N=filename New program library to be written on named file.

omitted UPDATE does not generate a new program library

O List output file

omitted or O List output is written on a file named OUTPUT.

O=filename List output is written on named file.

P Old program library; ignored on creation run.

omitted or P Old program library on file OLDPL

P=filename Old program library on named file

Q Quick update (takes precedence over F)

omitted If F is not specified, this is the normal (selective) mode. See F omitted.

Q Only those decks specified on COMPILE directives are processed. Corrections other than ADDFILE, that reference cards in decks not specified on COMPILE cards, are not processed. The compile file contains decks specified on COMPILE directives and any common decks called from decks. The new program library contains all decks mentioned on COMPILE directives as well as all common decks they call and common decks encountered on the old program library prior to processing of all of the specified decks. The source file contains the same active cards that are written on the new program library if a new program library is selected.

S Source output; the contents of the source file are determined by the mode in which UPDATE is operating and the decks named on the COMPILE directives.

If Q is not selected, regardless of F, the source file contains all cards required to recreate the library. This recreated library is resequenced because sequencing information is not included on the source file. This file contains all currently active DECK, COMDECK, and CALL directives in addition to all active text information. Decks are not necessarily in a sequence accepted by UPDATE for creating a new program library.

If Q is selected, decks written on the source file are those named on COMPILE directives and common decks they call. All common decks encountered on the old program library before all explicitly specified decks are included on the source file.

omitted UPDATE does not generate a source output file unless the source output is specified by the T option.

S Source output written on file SOURCE.

S=filename Source output written on named file.

T Source output excluding common decks (takes precedence over S)

omitted No source output unless source output specified by S.

T Source output excluding common decks on file SOURCE.

T=filename Source output excluding common decks is on named file.

8 80-column output on compile file

omitted Compile file output is composed of 90-column card images.

8 Compile file output is composed of 80-column card images.

* Master control character

omitted The master control character is *

*=character The master control character, which is the first character of each directive, for this UPDATE run is the character following the equals sign. The character may be A through Z 0 through 9 + - * / $ or =

On a correction run, if the master control character is not the same as the character used when the old program library was created, UPDATE uses the character indicated on the old program library.

Comment control character

omitted	The comment control character is / (slash)
/=character	The comment control character for this UPDATE run is the character following the equals sign. The character may be either: A through Z 0 through 9 or + - * / $ or =

UPDATE DIRECTIVES

UPDATE directives are cards in the input stream containing a master control character in column 1, a valid UPDATE directive beginning in column 2, and a blank or comma immediately following the directive. The default master control character * is used in subsequent examples; however, the master control character may be redefined through an UPDATE control parameter. The master control character is stored in the key word that prefixes the library file created; therefore some consistency should apply to redefining this parameter, especially with regard to directives that UPDATE places into the library file. When placed on the library, directives CALL, DECK, COMDECK, YANKDECK, and YANK carry with them the master control character with which they were introduced. UPDATE can recognize them on subsequent runs only when the same master control character is given. If file modification is attempted with a master control character other than that specified on OLDPL, a non-fatal error will occur.

UPDATE directives are written in the form:

```
*control word
```

or

```
*control word   parameter list
```

* represents the master control character, and the control word follows without intervening spaces. The control word is terminated by a comma or blanks; any number of blanks may appear between the control word and the parameter list. Parameters are separated from each other by commas; embedded blanks are not permitted. Control words may be written in an abbreviated form, as shown in the following table. All numeric values are to contain decimal digits, except where noted.

Abbreviation	UPDATE Directive Format
*AF	*ADDFILE fname,ident.seqnum
*C	*COMPILE dname1,dname2, . . . dnamen *COMPILE dname1.dname2
*CA	*CALL dname
*CD	*COMDECK dname,p1
*D	*DELETE ident1.seqnum,ident2.seqnum *DELETE ident.seqnum
*DK	*DECK dname
*I	*INSERT ident.seqnum
*ID	*IDENT idname,p1,p2, . . . ,pn
*P	*PURGE idname1,idname2, . . . ,idnamen *PURGE idname1.idname2 *PURGE idname,*
*PD	*PURDECK dname1,dname2, . . . dnamen *PURDECK dname1.dname2
*RD	*READ fname
*Y	*YANK idname1,idname2, . . . ,idnamen *YANK idname1.idname2
*YD	*YANKDECK dname1,dname2, . . . dnamen

CARD IDENTIFICATION

UPDATE recognizes one full form and two short forms of card identifiers. The full form card identifier is:

ident.seqnum

ident. 1 to 8 character name of a correction set or deck. A period terminates the ident name.

seqnum Decimal ordinal (1 to 65535) representing the sequence number of the card within the correction set or deck. Any character other than 0 to 9 terminates the sequence number.

The two short forms of card identifiers can be used on INSERT or DELETE directives. The short forms are expanded as follows:

seqnum   Expands to idname.seqnum where idname is a correction set identifier, whether or not it is also a deck name.

.seqnum   Expands to dname.seqnum where dname is a deckname.

In the short form, idname is assumed to be the last explicitly named ident given on an INSERT or DELETE directive, whether or not it is a deck name. The dname is assumed to be the last explicitly named ident given on an INSERT or DELETE directive that is known to be a deck name. Both of these default idents are originally set to YANK$$$ so the first directive using a card identifier must use the full form to reset the default.

All deck names are also idents. Thus, if EXAMPLE is the deck name last used, and there is no subsequent explicit reference to a correction set identifier, then both .281 and 281 expand to EXAMPLE.281 as the identifier. If there is an explicit reference to a correction set identifier after the explicit reference to the deck name, then 281 would expand to the correction set ident while .281 would expand to EXAMPLE.281 as the identifier.

For example; A is a deck name and B is a correction set on an UPDATE old program library.

```
*ID C
*INSERT A.2
< data card >
*INSERT B.1
<data card>
*D 2, 3         expands to *DELETE B.2, B.3
*D 4, .5        expands to *DELETE B.4 A.5
*D .7, 5        expands to *DELETE A.7, B.5
*D .9, .10      expands to *DELETE A.9, A.10
```

whereas:

```
*ID D
*INSERT B.1
< data card>
*INSERT A.2
<data card>
*D 2, 3         expands to *DELETE A.2, A.3
*D 4, .5        expands to *DELETE A.4, A.5
*D .7, 5        expands to *DELETE A.7, A.5
*D .9, .10      expands to *DELETE A.9, A.10
```

PROGRAM LIBRARY FILES

Any OLDPL will have first been a NEWPL; it can become an old program library only after its creation run. UPDATE creates new library files in sequential format. The first word in the file is a key word; the second contains two counters, one the number of deck names in the deck list, and the other a count of the card image identifiers in the directory.

The deck list starts in word three of the library record; it is a table containing one entry for each program deck in the file. Each entry is a word containing a coded deck name. The deck names are listed in the same sequential order as they appear in the library file.

The directory follows the deck list and contains the library identifier for every program deck that has ever appeared in the file. Each entry contains the deck identifier name, left justified with blank fill. Program identifiers differ from deck names in that the deck name is taken from the *DECK or *COMDECK directive which precedes the source deck in the input stream. Program identifiers are taken from the *IDENT directive which precedes correction sets in the input stream. Identifiers are listed in the directory in the order in which they were introduced into the library.

The remainder of the sequential format library file contains the compressed symbolic card images and their correction history bytes.

Card images may be deleted from an old program library in one of two ways. UPDATE's purge features allow card images to be removed permanently and irrevocably. Once a purge has been performed on a program library, it cannot be restored to an earlier level. However, since each set of corrections is associated with a unique identifier, any cards affected by the correction set can be referenced relative to the correction set. Thus, in a correction run, all or part of a correction set can be removed (yanked), and the effect of such an operation can be reversed, if necessary.

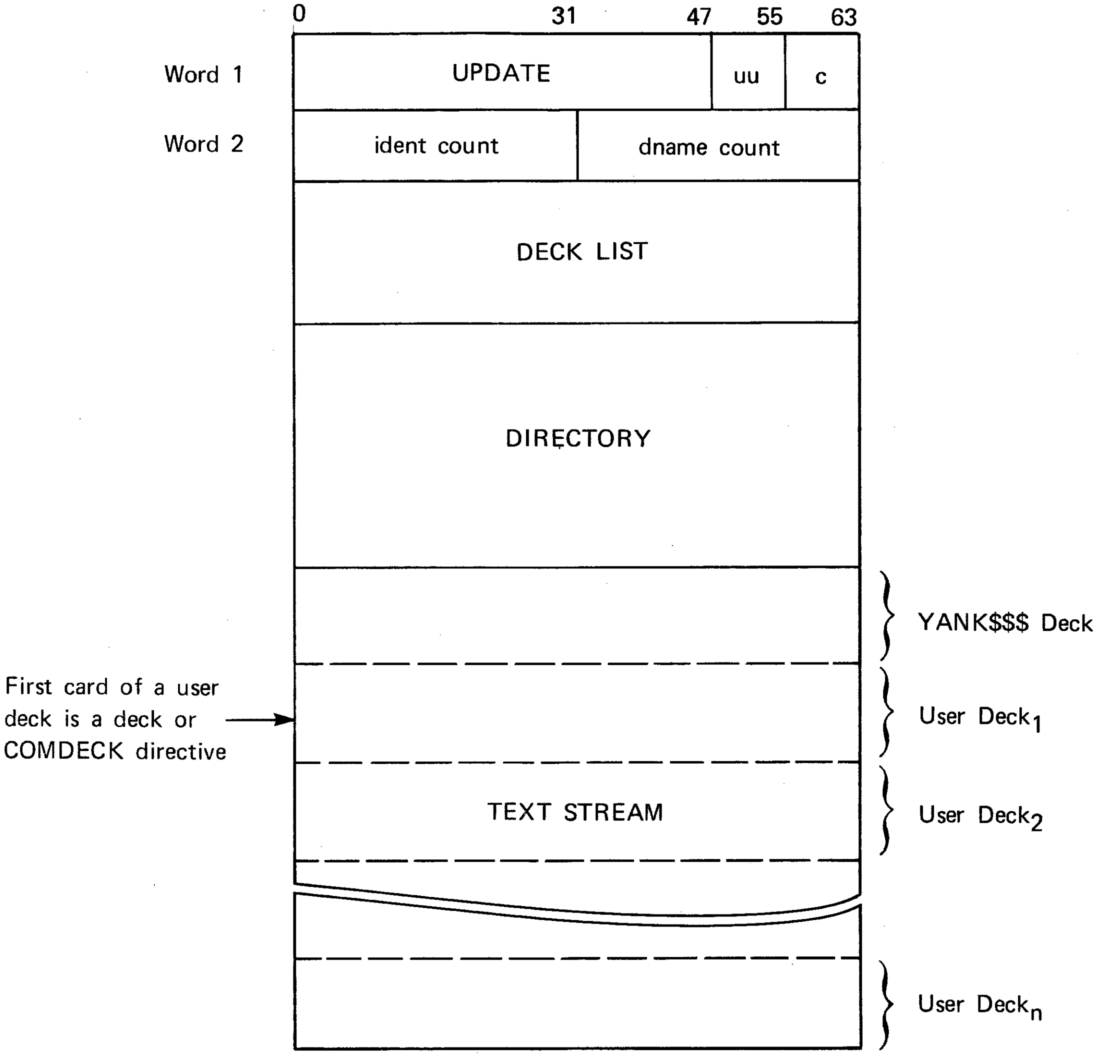
Each card image on the OLDPL also belongs to a deck. A deck is defined as an active *DECK or *COMDECK directive along with all card images that follow on the OLDPL (active or inactive) up to the next active *DECK or *COMDECK directive. The deck to which a card image belongs is determined by the card's position and by the status (active or inactive) or the *DECK or *COMDECK directives. Since the *DECK and *COMDECK directives can be deactivated (by *DELETE and *YANK), directives belonging to one deck at the beginning of an UPDATE run may belong to a different deck at the end of the run. When a *DECK directive is deactivated, all card images in that deck become members of the preceding deck on the OLDPL; thereafter, they are controlled by directives that affect the previous deck as a unit, such as *PURDECK.

A source deck can be assigned common status when it is first incorporated into a program library file. Common decks can be called from within other decks, as they are being written on the compile file. On the compile file, when UPDATE encounters a card calling a common deck, the text of that deck is used to replace the card issuing the call.

PROGRAM LIBRARY FORMAT

UPDATE creates new library files in sequential format. The first word in the file is an ASCII keyword; the second is a counter word containing the number of deck names in the deck list and a count of correction set identifiers in the directory.

The library format is:



Word one:

0 47	55	63
UPDATE	uu	c

Bits	Field	Description	
0-47	UPDATE	Identifies the field as being an UPDATE program library file. Field contains the word UPDATE in ASCII code.	
48-55	uu	Modulo #100 count representing number of times this program library has been accessed.	
56-63	c	Indicates master control character used when this library file was created.	
		*	First character of directives is an asterisk, the conventional master control character.
		other	First character of directives is the character indicated. On a correction run, if the master control character specified on the UPDATE control card does not match this character, UPDATE changes the character to match the one in this field.

Word two:

0 31	63
ident count	dname count

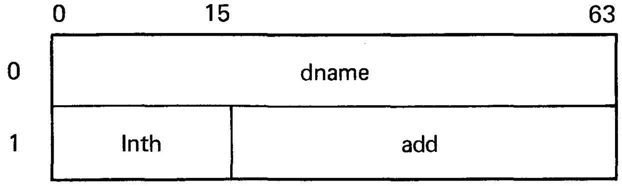
The second word of the program library file is composed of two binary count fields:

ident count	Count of all identifiers in the directory
dname count	Count of all deck names in the deck list

Deck List Format:

The deck list contains a two-word entry for each deck on the library. The first entry points to the YANK$$$ deck.

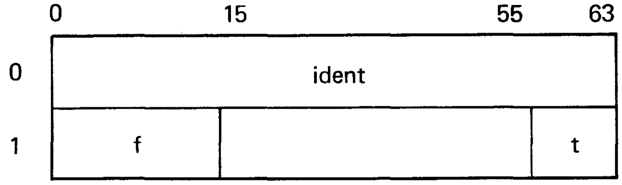
Each entry has the following format:



Word	Bits	Field	Description
0	0-63	dname	1 to 8 alphanumeric characters representing deck name obtained from DECK or COMDECK directive when deck was placed on library. The first dname is YANK$$$.
1	0-15	lnth	Length of deck in number of small pages.
	16-63	add	Address of first character for the deck in the old program library.

Directory Format:

The directory is a table that contains one entry for each DECK, COMDECK, and IDENT that has ever been used for this library. Each directory entry consists of two words containing the 8-character identifier in ASCII, left justified. Correction set identifiers and deck names are listed chronologically as they are introduced into the library. An identifier that has been purged is not printed on the listable output file although table space is allocated to it.



Word	Bits	Field	Description
0	0-63	ident	1 to 8 alphanumeric characters representing the name of a correction set or deck
1	0-15	f	Flags for this ident
	56-63	t	Type (0=deck; 1=comdeck; 7=ident)

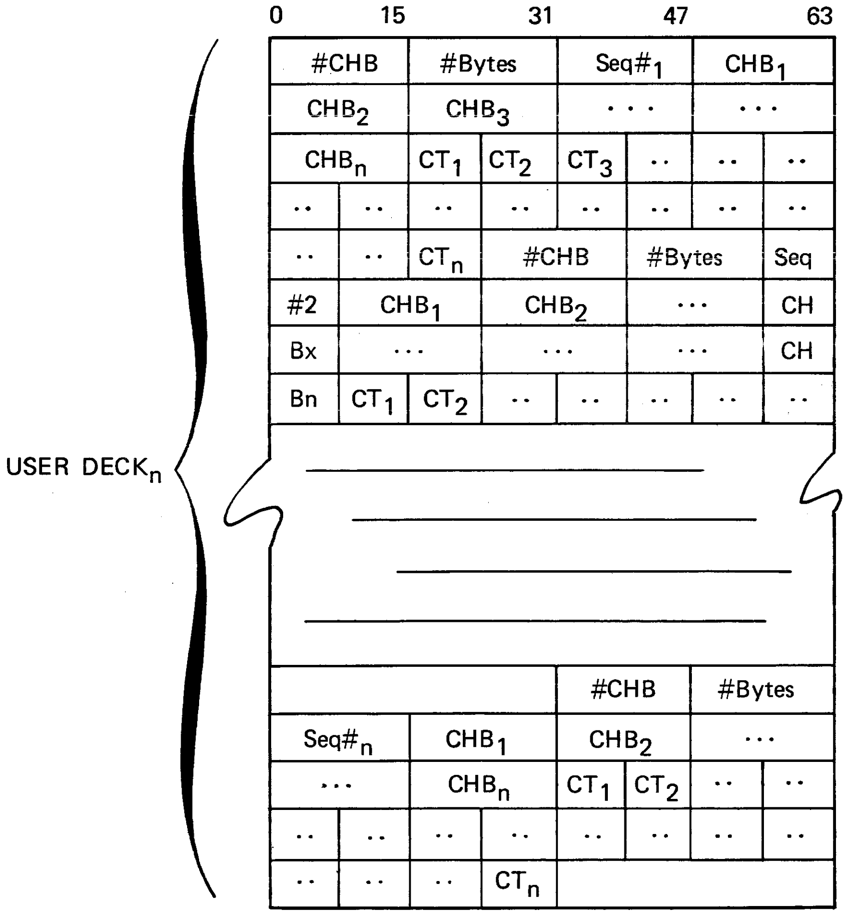
YANK$$$ Deck:

The YANK$$$ deck is automatically created on a creation run as the first deck on the program library. On correction runs, UPDATE inserts into the YANK$$$ deck any YANK or YANKDECK directives that it encounters during the read directive phase. These directives acquire identification and sequence information from the correction set from which they originate.

Although the YANK$$$ deck, as a whole, cannot be yanked or purged, cards in the deck can be deleted, yanked or purged from it. This deck does not have a DECK card as its first card image.

User Deck:

Each deck contains an indefinite number of words that maintain a correction history and the compressed image of each card in the deck. Its format is:

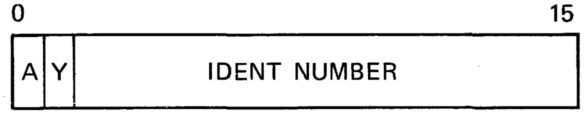


#CHB Number of correction history bytes (CHB) for this card

#Bytes Number of bytes of compressed text

Seq# Sequence number of the card in this deck

CHB Correction history byte



A – Activity bit – 0 Correction set deactivated card

1 Correction set activated card

Y – Yank bit – 0 Card not yanked

1 Card has been yanked

IDENT NUMBER – Index to Directory entry that contains the name of the deck or correction set that introduced the card or changed the card status.

CT Compressed text in ASCII characters

OUTPUT FILES

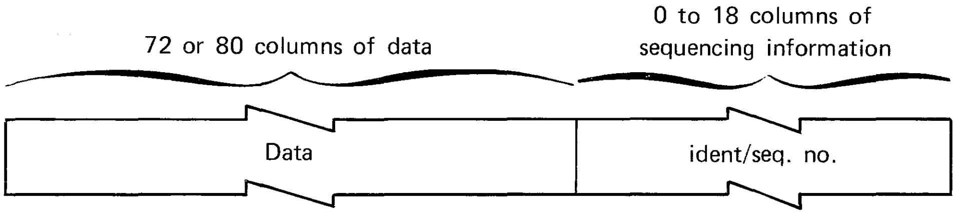
The listing of the output file is optional; its contents vary, depending on the nature of the run. For creation runs, UPDATE produces a list of known DECK and IDENT names, COMDECKs processed, DECKs written to the COMPILE file, all error messages, and all active control cards. For correction runs, the preceding are augmented to include all cards for which status changed during this UPDATE, and all cards encountered in the input stream.

COMPILE FILE

The compile file is a primary form of output from UPDATE. It contains updated source card images to be assembled or compiled. The default name of the compile file is COMPILE, but the name of the file can be changed through the C option on the UPDATE control card.

During a full update run or during a creation run, the compile file contains all decks that are on the new program library. In normal mode (F and Q not selected), the compile file contains only decks updated during the current UPDATE run or decks specified on COMPILE directives. The sequence of the decks on the compile file is determined by their order in the deck list.

Through control card options, a user can specify whether the text on the file is to be compressed or expanded. The expanded compile file format for each card consists of 72 or 80 columns of data followed by 0 to 18 columns of sequence information. The maximum size of a card image is 90 columns. The expanded card image is:



UPDATE attempts to place sequence information in the columns remaining in the card image after the data columns have been allocated. When the data field is 72 columns and the card image is 90 columns, column 73 is left blank making 17 columns available for sequence information. In this case, the 1 to 8 character ident is left adjusted in column 74 and the sequence number is right adjusted in column 86.

When the data field is 72 columns and the card image is 80 columns, 8 columns are available for sequencing information. If the data field is 80 columns and the card image is 90, then 10 columns are available for sequencing information. In either of these cases, if the ident and sequence number exceed the field, UPDATE truncates the least significant characters of the ident, leaving the sequence number intact.

If the data field and card image are both 80 columns, the compile file output cannot have sequence information appended.

In the following example, ident is SEVENCH and the sequence number is 1144:

Column	73	74						80						86				90
Normal compile output		S	E	V	E	N	C	H			1	1	4	4				
With D option									S	E	V	E	N	C	1	1	4	4
With 8 option	S	E	V	E	1	1	4	4										

SOURCE FILE

The source file contains a copy of all active DECK and COMDECK directives and all active cards within each deck. The source file is optional output from UPDATE through the use of the S and T options on the UPDATE control card; once created, the file can be used as source input on subsequent UPDATE runs. The source file is a coded file that contains 80-column images.

When Q is not selected on the UPDATE control card, the source file generated contains all cards needed to create a program library. The source file can be used as a back-up copy of the library or, providing that common decks occur first, can be used as input for an UPDATE run that produces a resequenced program library with all the inactive cards purged.

When Q is selected on the UPDATE control card, the source file contains all decks requested on the COMPILE directives, all common decks that they call, and any common decks encountered prior to processing all of the specified decks. The only directives that can legally appear in the source file are DECK, COMDECK, and compile file directives.

LIBRARY FILE CREATION AND MAINTENANCE

LIBRARY FILE CREATION

UPDATE operates in creation or correction mode. UPDATE considers any run to be a creation run if the first directive it encounters in the input file (except for *comment or *READ) is a *DECK or *COMDECK directive. Even if an old program library file is assigned to the job, UPDATE will ignore its existence and process the run in creation mode. Similarly, if UPDATE encounters a *DECK or *COMDECK directive while inserting text during a correction run, it will add the deck to the old program library following the card specified by *INSERT.

In a creation run, each *DECK or *COMDECK in the input stream defines a deck to be incorporated into the library file. The directives also cause entries to be made in the directory and deck list. UPDATE identifies and sequentially numbers each card image written to the library file. Identifiers take the form, dname.seqnum; dname is the name associated with the deck, and seqnum is the sequence number. Numbers are assigned, starting with 1 for the *DECK or *COMDECK card, and including all text cards and those introduced by a *READ directive.

Acceptable formats:

*DECK dname *DK	dname	1-8 character deck name for this deck that is different from any other deck name in the deck list; legal characters are: A through Z 0 through 9 + - * / ( ) $ = _

```
*COMDECK dname,NOPROP
*CD
```

dname 1-8 character name of deck being introduced; this name must differ from any names already in the deck list. Legal characters are: A to Z 0 to 9 + - * / ( ) $ = _

NOPROP Inclusion of this parameter specifies that if this deck is modified, decks calling this common deck are not to be considered as modified. The effects of the changes are not propagated during UPDATE mode (F option not specified).

The primary difference between *DECK and *COMDECK is that the latter introduces a deck that may be called from other decks as they are being written on the compile file; however, common decks must precede other decks that may call it during a creation run.

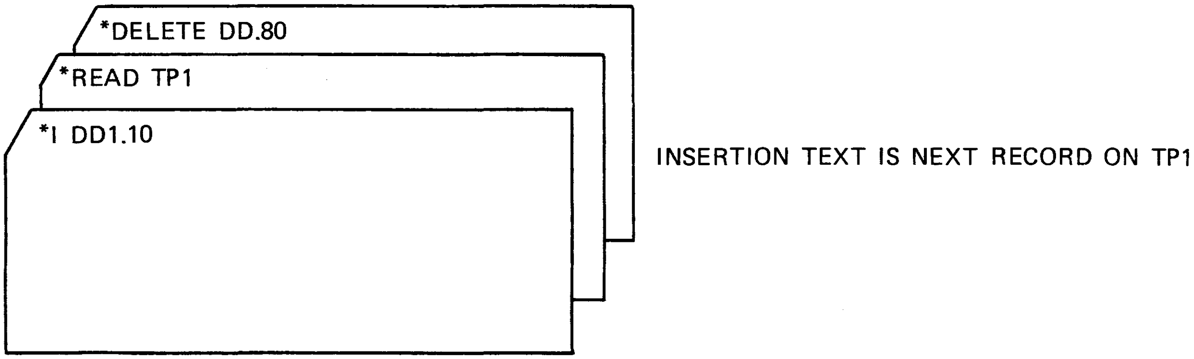
When text and directives are to be read from files other than the input stream, file manipulation directives are used. The directives may appear anywhere in the input stream; they may appear only on the main input file and may not reference files specified for internal use by UPDATE, such as those identified by file parameters. To read from a file other than the input stream, the user program should include the control statement:

```
*READ fname
*RD
```

fname Name of file containing insertion text and/or directives; READ and ADDFILE are illegal on the file.

The READ directive causes UPDATE to temporarily stop reading the input file and begin reading directives and insertion text from the named file at its current position. UPDATE reads from this alternate directives file until it encounters an end-of-file mark and then resumes reading the next card from the primary input file.

Example:



```
*DELETE DD.80
*READ TP1
*I DD1.10
```

INSERTION TEXT IS NEXT RECORD ON TP1

LIBRARY CORRECTION DIRECTIVES

The library correction process is the most common use of UPDATE. UPDATE directives and data card images comprise the correction set. The directives provide UPDATE with instructions concerning the modification of the program library text stream. The old library is not destroyed; the corrections are permanent only in the new program library being created.

A correction run consists of a read input stream phase and a correction phase. During the first phase, UPDATE reads directives and text, adds new decks, and constructs a dictionary of requested correction operations. During the second phase, UPDATE performs the requested modifications on a deck-by-deck basis.

Correction directives cause card images to be inserted or deleted from program library decks according to card sequence number. A card can be identified by deck name and sequence number. Each new card is assigned a correction set identifier specified by the user. UPDATE sequences the new cards. All cards having the same correction identifier comprise a correction set. UPDATE permits a user to remove (yank) the effects of a correction set (or deck) and later restore the set (or deck). This feature is convenient for testing new code. Requests for yanking are maintained in the YANK$$$ deck. Before obeying a correction, UPDATE checks the correction identifier against the YANK$$$ deck to see if the correction has been yanked.

UPDATE also allows a complete and irreversible purging of correction sets (through the PURGE directive).

The following directives are used for correcting and updating a program library.

IDENT

```
*IDENT idname,p1,p2, . . . ,pn
*ID
```

idname 1-8 character identifier to be assigned to this correction set. Legal characters are:

A-Z 0-9 + - * / ( ) $ = _

This name causes a new entry in the directory. Each card inserted by the correction set and each card for which the status is changed receives a correction history byte that indexes this idname. Sequencing of new cards begins with one for this idname

Omitting idname causes a format error. If idname duplicates a name previously used, UPDATE issues an error message. Both errors are non-fatal.

This idname remains in effect until UPDATE encounters another IDENT directive or encounters PURGE, PURDECK, or ADDFILE directive.

pi Any number or one of the following parameters:

B=num Bias of num is to be added to sequence numbers. If more than one B parameter is specified, UPDATE uses the last one encountered.

K=ident The specified ident must be already in the directory in order for this correction set to be incorporated. If ident is unknown, UPDATE skips the correction set and resumes processing with the next IDENT, PURGE, PURDECK, or ADDFILE directive. If more than one K parameter is specified, all the idents must be known or the correction set is skipped.

U=ident The specified ident must not be known for this correction set to be processed. If ident is known, UPDATE skips the correction set and resumes processing with the next IDENT, PURGE, PURDECK, or ADDFILE directive. If more than one U parameter is specified, all the idents must be unknown or the correction set is skipped.

Example:

```
*IDENT ZAP,B=100,K=ACE,U=NON,U=ARF
```

The decimal bias of 100 is added to all ZAP correction set card sequence numbers. That is, the first card in correction set ZAP becomes 101 instead of 1. UPDATE skips the correction set if ACE is unknown or either NON or ARF is known.

INSERT

```
*INSERT ident.seqnum
*I
```

ident = identifier name

seqnum = decimal sequence number

Cards may be inserted in a deck with the *INSERT directive. ident.seqnum specifies the card after which the insertion is to be made.

ADDFILE

Directive format:

```
*ADDFILE fname,ident.seqnum
*AF
```

fname — Name of file from which information is to be read. This text cannot contain correction directives. If file is omitted, it is assumed to be the UPDATE input file named INPUT or its equivalent, as specified by the I parameter.

ident.seqnum — Identifier and sequence number of card after which decks are to be placed on program library.

When *ADDFILE is encountered, UPDATE reads creation directives and text data from the named file (fname) and inserts this information after the card (ident.seqnum) on the new program library. The first card of fname must be *DECK or *COMDECK. UPDATE reads from this file until an end-of-file, which returns UPDATE to the main file. If the fname on *ADDFILE is the main input file, cards are added until an end-of-file or the next UPDATE directive, whichever occurs first. UPDATE directives placed on the library (*DECK, *COMDECK, *CALL) do not terminate the *ADDFILE function. *ADDFILE is illegal on an alternate input file. A READ directive is illegal during processing of the added file, unless the added file is the main input file.

Either or both fname and ident.seqnum may be omitted from *ADDFILE. If ident.seqnum is absent, the *ADDFILE function occurs at the end of the library. If both parameters are missing, the *ADDFILE function occurs at the end of the library, and it is taken from the main input file (INPUT by default or as specified by the I parameter). If only one parameter is present, it is assumed to be a file name.

YANK

*YANK $idname_1$,$idname_2$, . . . ,$idname_n$
*Y

*YANK $idname_1$.$idname_2$
*Y

$idname_i$ — Name of correction set previously applied to the program library. If UPDATE fails to find $idname_i$, it issues an error message.

The correction set idname$_1$ and all sets, up to and including idname$_2$ on the directory are yanked. If idname$_1$ and idname$_2$ cannot be located or are in reverse order, UPDATE issues an error message.

UPDATE places the YANK directive in the YANK$$$ deck. During the modification phase, UPDATE checks each correction to see if it has been yanked. All yanked corrections are ignored. If the card was deactivated by the yanked correction set, UPDATE reactivates it. If the card was activated by the yanked correction set, UPDATE deactivates it. Thus, UPDATE changes the correction history byte for every card that has changed status.

A YANK must be part of a correction set.

A YANK directive does not terminate insertion.

Example:

```
*YANK OLDMOD
```

This directive causes all effects of the correction set OLDMOD on the entire library to be nullified. Cards introduced by OLDMOD are deactivated; cards deactivated by OLDMOD are reactivated.

YANKDECK

The YANKDECK directive deactivates all cards within the decks specified. The directive format is:

*YANKDECK $dname_1,dname_2,\ldots,dname_n$
*YD

dname$_i$ Name of deck to be deactivated. All cards in the deck are deactivated regardless of the correction set to which they belong. If UPDATE is unable to find dname$_i$, it issues an error message.

The YANK$$$ deck cannot be deactivated by the yank operation.

The YANKDECK directive must be part of a correction set.

YANKDECK does not terminate insertion.

Example:

```
*YANKDECK OLDDECK
```

This directive affects all cards in OLDDECK, regardless of the correction set to which they belong.

PURGE

The PURGE directive causes the permanent and non-reversible removal of a correction set or group of correction sets. A PURGE directive can appear anywhere in directive input. Because it terminates a previous correction set, an INSERT or DELETE cannot follow a PURGE directive; they must follow an IDENT directive. The YANK$$$ deck cannot be purged. Purging cannot be rescinded.

The PURGE directive has the following three basic formats:

*PURGE idname_1,idname_2, . . . , idname_n
*P

idname_i Identifiers for correction sets to be purged.

*PURGE idname_1.idname_2
*P

Correction set idname_1 and all sets, up to and including idname_2 on the directory, are purged. If idname_1 and idname_2 cannot be located or are in reverse order, UPDATE issues an error message.

PURGE idname,
*P

Correction set idname and all correction sets, that have been introduced after idname, are purged. This returns the library to an earlier level only if no PURGE or PURDECK directive has been issued previously.

If UPDATE cannot locate a specified correction set, it issues an error message. Purged idnames can be reused on subsequent correction sets providing they do not appear in the YANK$$$ deck.

PURDECK

A PURDECK directive causes the permanent and non-reversible removal of a deck or group of decks from the program library. Every card in a deck is purged, regardless of the ident to which it belongs. PURDECK does not purge idnames. Thus, a deck name purged as a result of PURDECK can be reused as a dname. It can be used as a new idname only if it is not already in the directory list.

A PURDECK directive can be any place in the directives input. It terminates any previous correction set. Therefore, INSERT or DELETE cannot follow a PURDECK directive but must come after an IDENT directive. The YANK$$$ deck cannot be purged. Purging cannot be rescinded.

The PURDECK directive has two basic formats:

*PURDECK dname$_1$,dname$_2$, . . . ,dname$_n$
*PD

dname$_i$ Deck names for decks to be purged.

*PURDECK dname$_1$.dname$_2$
*PD

The deck named dname$_1$ and all decks, up to and including dname$_2$ listed in the deck list, are purged. If dname$_1$ and dname$_2$ cannot be located or are in reverse order, UPDATE issues an error message.

DELETE

The DELETE directive has two formats:

*DELETE ident.seqnum
*D

*DELETE ident$_1$.seqnum,ident$_2$.seqnum
*D

ident.seqnum Card identifier for single card to be deleted.

ident$_1$.seqnum, ident$_2$.seqnum Card identifiers of first and last cards in sequence of cards to be deleted. ident$_1$.seqnum must occur before ident$_2$.seqnum on the library. The range can include cards already deleted which are affected by the DELETE.

With the DELETE or D directive, the user deactivates a card or block of cards and optionally replaces it with insertion cards which follow the DELETE directive.

A deactivated card remains on the library and retains its sequencing. It can be referred to in the same way as an active card. A deactivated card is not included in compile decks or source decks.

COMMENT

* / comment

Comments to be listed with a correction set may be included with a comment directive. It has a master control character * in column 1, a comment control character / in column 2, and a comma or blank in column 3. This card is ignored by UPDATE except that it is copied onto the listable output file. A comment may appear at any place in the input stream.

CALL

*CALL dname
*CA

dname Name of common deck to be written on compile file.

UPDATE writes the text of a previously encountered common deck, dname, onto the compile file. Common code, such as system symbol definitions, must be declared in the common deck and used in subsequent decks or assemblies without repeating the data cards.

The CALL card does not appear on the compile file. The contents of the common deck, excluding the COMDECK card, follow immediately. Common decks can call other common decks. However, to avoid circularity of calls, a common deck must not call itself or call decks that contain calls to the common deck. A CALL directive is effective only when it is within a deck.

COMPILE

The COMPILE directive has two formats:

*COMPILE dname$_1$,dname$_2$, . . . ,dname$_n$
*C

*COMPILE dname$_1$.dname$_2$
*C

dname$_i$ Name of deck to be written on the compile file, new program library, and source file.

The first form of the directive requests one or more decks that may be in any sequence on the library. The second form requests all decks in the deck list starting with dname$_1$ through dname$_2$. If UPDATE fails to find the named deck or if the deck names are reversed, it issues a diagnostic message.

COMPILE directives can be anywhere in the input stream. Calling a common deck from within a deck being updated results in the common deck being updated.

NORMAL SELECTIVE MODE

During a normal update run (F and Q are not selected on the UPDATE control card), UPDATE writes on the compile file all decks specified on COMPILE directives as well as all decks corrected during the run. COMPILE causes a deck to be written on the compile file in the sequence encountered on the old program library.

FULL UPDATE MODE

During a full update run (F selected on UPDATE control card), UPDATE ignores COMPILE directives. It updates all decks in the sequence encountered on the library.

QUICK MODE

During a quick update run (Q selected), only decks specified on COMPILE directives and called common decks are written on the compile file. These decks are written in the sequence encountered on the program library.

ERROR CONDITIONS

UPDATE can detect four overlapping correction situations:

Type 1 Two or more modifications are made to one card by a single correction set.

Type 2 A modification attempts to activate an already active card.

Type 3 A modification attempts to deactivate an already inactive card.

Type 4 A card is inserted after a card which was inactive on the OLDPL.

When any of these types is detected, UPDATE prints the offending line with the words TP.n OVLP appended on the far right. Detection of an overlap does not necessarily indicate a user error. Overlap messages are advisory, and point to conditions in which the probability of error is greater than normal.

Type TP.2 and TP.3 are detected by comparing existing correction history bytes with those to be added. Complex operations involving YANK and PURGE may generate these overlap messages even though no overlap occurs.

Modifications for each correction set are performed by UPDATE in the order in which sets are introduced. The order is irrelevant if no correction is dependent on another. If a dependent relationship exists, however, order is of paramount importance.

CHECKPOINT/RESTART 11

A task may be ended as the result of machine malfunction, or program error. Abnormal termination may occur at any time during task execution. The checkpoint/restart facility is available to avoid loss of machine time when an abnormal end occurs during a long task. A user can capture the status of his executing task and its controllees at specified points so the task and controllee chain may be restarted later. The checkpoint/restart package is called as a subroutine; it is loaded into the user's virtual space and creates and checkpoints the tasks to disk files.

Checkpoint/restart cannot handle tapes. It restores message control only for the checkpointing program and programs at lower levels; however, if a checkpoint occurs while a system message is outstanding, the message is not preserved for the restart procedure. The user is responsible for ensuring that the checkpoint occurs at a logical breaking point during task execution. Any restarted task with a tape attached is responsible for re-establishing connection with the tape as well as repositioning, if desired. ,For a controllee with a message bypass pointing to a controller of higher level than the checkpointing task, the bypass is set by restart to point to the checkpointing task.

Checkpoint/restart verifies that private, public, and scratch files open at checkpoint time still exist and occupy the same file space.

CALLING SEQUENCE

CALL CHKPNT(filename,n,m,restart-flag,error-flag,ername)

filename — Name used for restart files. The filename must be left justified with blank fill. The task level will be added as the last character to the filename (replacing the eighth character if it is specified).

n, m — Available IOC numbers

restart-flag — Returned to user:

= 0 return was from checkpoint

≠ 0 return was from restart

error-flag — Returned to user:

= 0 no error

≠ 0 error response

ername — Filename returned by checkpoint/restart when an error occurs specific to a file; error flag field indicates errors that cause ername to be set.

Subroutine CHKPNT uses the labeled common block CKPT for an input/output buffer.

Checkpoint error responses:

1 Unable to create checkpoint file because of duplicate name (see ername)

2 Input/output connector is already in use or IOC is invalid

3 System table space is not adequate for file creation or no mass storage is available for a file

4 Invalid filename (see ername)

5 System error; checkpoint failed

Restart error responses:

10 Full system tables inhibit initialization of controllee program

11 Five levels of controllee exceeded

13 Controllee program file was not found (see ername)

14 Insufficient time to run controllee program (see ername)

15 System error; restart failed

17 Controllee program file is not executable (see ername)

18 Mass storage error (see ername)

19 Abnormality in controllee program file or dropfile I/O connector entry (see ername)

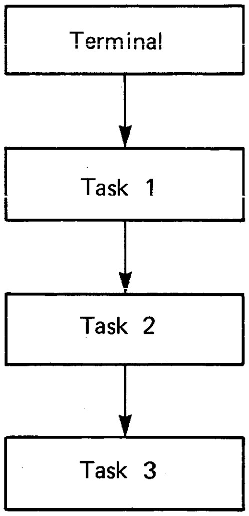
CHECKPOINT FILES

Checkpoint files are copies of the drop files for the checkpointed task and its controllees. Each controllee drop file is copied to a separate file, and the filename for the checkpointing controllee is the name provided in the checkpoint call. The last character of the filename for the lower level controllees designates the task's level relative to the checkpointing controllee. The checkpointing task file contains a list of controllees to be restarted.

RESTART

The checkpointing task restart file is executed when an execute line is issued for the restart filename. When the controllee or controllee chain (if lower level controllees exist for the checkpointing task) has been re-initialized, the restart flag is set non-zero and control returns to the program that called checkpoint. A drop file with no lower level controllees also may be restarted by issuing an execute line for the drop filename. To restart either a checkpoint file or a drop file, the respective files for the task and its controllees must have been preserved. Like drop files, checkpoint files (which have been restarted) are preserved upon abnormal termination and may be preserved or destroyed at the option of the program upon normal termination.

EXAMPLE



Task 2 issues:

CALL CHKPNT (8HRESTART , 2, 3, IRSFL, IERR, IERN)

Two files would be generated:

RESTART, RESTART1

Input/output connectors 2 and 3 would be used, then released by checkpoint

IRSFL = 0 if return from checkpoint

= 1 if the file RESTART had been executed

STAR RECORD MANAGER 12

SRM OVERVIEW

The STAR Record Manager is designed to handle the logical input/output interface between the STAR Operating System and the language processor users.

STAR Record Manager (SRM) is a collection of modules implemented in META language. All of the modules are callable as macros. Parameters for SRM macros designate a 64-bit register which contains the value to be used in the macro expansion. SRM routine parameters contain actual values rather than value addresses. All the SRM modules are part of the STAR system library; they are automatically loaded and linked to the user program at the point of reference.

SRM FUNCTIONS

The five functional tasks performed by SRM are listed below together with the related SRM modules.

FILE DEFINITION AND MAINTENANCE FUNCTIONS

FILE,GENFIT Generates file information table (FIT) for implicit input/output. FILE generates the FIT at the place of macro reference; GENFIT generates the FIT at a specified location.

FILEX, GENFITX Generates file information table (FIT) for explicit input/output. FILEX generates the FIT at the place of macro reference; GENFITX generates the FIT at a specified location.

GETFIT Retrieves values of specified fields in the FIT.

SETFIT Sets values in specified fields of the FIT.

CREATE Makes the file known to the system and connects it to the user program. Issues CREATE system message.

DESTROY Issues system DESTROY FILE message to sever connection between the user program and a tape drive or a mass storage file and releases the tape drive or mass storage space for re-assignment.

REDUCE Issues the system REDUCE FILE LENGTH message to reduce the length of an existing private mass storage file. (This feature is not currently implemented.)

CHANGE Issues the system CHANGE FILE NAME OR ACCOUNT message to change the name or account number of an existing private file.

FILE INITIALIZATION AND TERMINATION FUNCTIONS

OPEN Connects the file to the user program; generates the OPEN FILE system message.

CLOSE Disconnects the file from the user program; generates the CLOSE FILE system message.

DATA TRANSFER AND STRUCTURING FUNCTIONS

BUFOP Defines or releases a buffer for explicit input/output.

READ Transfers data to buffer 1 or 2 from the file using explicit input/output. Issues the system message EXPLICIT I/O.

WRITE Transfers data from buffer 1 or 2 to the file using explicit input/output. Issues the system message EXPLICIT I/O.

GET Transfers data from next record to a specified user space.

GETM Transfers more data from the same record to a specified user space.

PUT Transfers data from a specified user space to the file as the next record or part of it.

PUTM Transfers more data from a specified user space to the file as part of the current record.

GETL Distributes data to areas specified in the parameter list from the next record of the file.

GETML Distributes data to areas specified in the parameter list from remainder of the current record of the file.

PUTL Gathers data from areas specified in the parameter list to form the next record of the file.

PUTML Gathers data from areas specified in the parameter list to extend the current record of the file.

GETBCD Transfers next record (must be ASCII) from the file to a specified user space.

PUTBCD Transfers a specified number of ASCII characters from user space to the file.

WEOR	Writes end-of-record mark.
WEOS	Writes end-of-section mark.
WEOI	Writes end-of-information mark.

FILE POSITIONING FUNCTIONS

REWIND	Positions the file to beginning-of-information.
SKIP	Skips forward or backward specified number of records.
SKIPS	Skips forward or backward specified number of sections.
BKSPC	Backspaces one coded record.

MISCELLANEOUS SYSTEM INTERFACE FUNCTIONS

GIVE	Gives the file to a specified user. Generates the system message GIVE FILES.
TERM	Terminates program execution. Generates the system message TERMINATE.
STATUS	Interrogates input/output completion status, gives up central processor if so specified, and returns the system error response code. Generates the system message GIVE UP CPU UNTIL I/O COMPLETES.
TPFCN	Performs miscellaneous tape functions.

FILE INFORMATION TABLE

To facilitate input/output processing, STAR Record Manager maintains certain information for each file in a file information table (FIT), as shown in figure 12-1. A file is identified to SRM by the location of its FIT. Each FIT requires 20 words and resides in the user's virtual space. The user is responsible for reserving virtual space for the table. In FORTRAN, the DIMENSION or similar statement can be used; in META, the FIT can be assembled in the data section established by the MSEC 1 statement.

SRM FUNCTION DESCRIPTIONS

In the descriptions which follow, SRM functions are described by a generic function name followed by a set of parameters. For example:

```
    CREATE fit,packid
```

where CREATE is the function name and fit and packid are parameters.

~~Parameters~~ are ~~described in terms of values.~~ ~~In a macro call~~, a ~~parameter designates the register~~ containing the value; in a call to a routine, a parameter contains the actual value. Table 12-1 indicates the names by which SRM functions can be called in various languages, the form of which differs from language to language.

Word	0	8	16	24	32	40	48	56 63
0	LFN(Logical Filename in ASCII, Left Justified, Blank Filled)							
	FA (File Attributes)							
1	IOC	MCAT	TYPE	LOK	ACS	MODE	CLAS	UNIT
2	FLEN (File Length)		BVA (File Base Virtual Bit Address)					
			FP (File Position, Logical Mass Storage File Address)					
	FS (File Status)							
3	FO	LOP	BDRY	OCS	Unused	PTL (Partial Transfer Length)		
4	0	TPM	EEA (Error Exit Address)					
5	Unused		EDX (End-Of-Data Exit Address)					
	BUF1 (Buffer 1 Descriptor)							
6	BLEN	Unused	BADD (Buffer Bit Address)					
	BUF2 (Buffer 2 Descriptor)							
7	CI (Current Index)							
8	IL (Index Limit)							
	RL (Record Length)							
9	SBC (Segment Byte Count)							
10	EDP (End-Of-Data Pointer) PRP (Preceding Record Pointer)							
11	RT[†]	DPC[†]	DRB (Directory Buffer Address)					
12	Reserved							
13 19	System Message Area (Alpha 1 to Beta 5)							

[†] RT (Record Type) and DPC (Directory Page Count) are used internally by SRM only.

Figure 12-1. File Information Table (FIT)

TABLE 12-1. STAR RECORD MANAGER MODULE REFERENCE TABLE

Module Class	Generic name used in this document	Called from FORTRAN as	Called from META as	Macro name in META	System message issued
File definition and maintenance	FILE			FILE	
	FILEX			FILEX	
	GENFIT	Q7GENFIT	GENFIT–	GENFIT	
	GENFITX	Q7GNFITX	GENFITX–	GENFITX	
	GETFIT	Q7GETFIT	GETFIT–	GETFIT	
	SETFIT	Q7SETFIT	SETFIT–	SETFIT	
	CREATE	Q7CREATE	CREATE–	CREATE	CREATE FILE
	DESTROY	Q7DESTR	DESTROY–	DESTROY	DESTROY FILE
	REDUCE	Q7REDUCE	REDUCE–	REDUCE	REDUCE FILE LENGTH
	CHANGE	Q7CHANGE	CHANGE–	CHANGE	CHANGE FILE NAME OR ACCOUNT
File initialization and termination	OPEN	Q7OPEN	OPEN–	OPEN	OPEN FILE
	CLOSE	Q7CLOSE	CLOSE–	CLOSE	CLOSE FILE
Data transfer and structuring	BUFOP	Q7BUFOP	BUFOP–	BUFOP	
	READ	Q7READ	READ–	READ	EXPLICIT I/O
	WRITE	Q7WRITE	WRITE–	WRITE	EXPLICIT I/O
	GET	Q7GET	GET–	GET	
	GETM	Q7GETM	GETM–	GETM	
	PUT	Q7PUT	PUT–	PUT	
	PUTM	Q7PUTM	PUTM–	PUTM	
	GETL	Q7GETL	GETL–	GETL	
	GETML	Q7GETML	GETML–	GETML	
	PUTL	Q7PUTL	PUTL–	PUTL	
	PUTML	Q7PUTML	PUTML–	PUTML	
	GETBCD	Q7GETBCD	GETBCD–	GETBCD	
	PUTBCD	Q7PUTBCD	PUTBCD–	PUTBCD	
	WEOR	Q7WEOR	WEOR–	WEOR	
	WEOS	Q7WEOS	WEOS–	WEOS	
	WEOI	Q7WEOI	WEOI–	WEOI	
File Positioning	REWIND	Q7REWIND	REWIND–	REWIND	
	SKIP	Q7SKIP	SKIP–	SKIP	
	SKIPS	Q7SKIPS	SKIPS–	SKIPS	
	BKSPC	Q7BKSPC	BKSPC–	BKSPC	
Miscellaneous	GIVE	Q7GIVE	GIVE–	GIVE	GIVE FILES
	TERM	Q7TERM	TERM–	TERM	TERMINATE
	STATUS	Q7STATUS	STATUS–	STATUS	GIVE UP CPU UNTIL I/O COMPLETES
	TPFCN	Q7TPFCN	TPFCN–	TPFCN	

To call the CREATE function in FORTRAN:

```
CALL Q7CREATE(FIT,PACKID)
```

The same function call, restated in META, would be:

```
CREATE       FIT,PACKID        * MACRO FORM
```

Programmers who wish to use META but not the SRM macros can follow the convention with regard to registers and parameters:

Register FF	First parameter
Register FE	Second parameter
.	.
.	.
.	.
.	.
etc.	Nth parameter

Therefore, the META user might have written:

```
RTOR     FIT,REGFF             *PARAMETER 1
RTOR     PACKID,REGFE          *PARAMETER 2
RTOR     CREATE_DB,REG1E       *ESTABLISH DATA BASE
BSAVE    CREATE_,REG1A         *CALL SUBROUTINE CREATE_
```

FILE DEFINITION AND MAINTENANCE

FILE and FILEX are available only as macros and are used for assembling the FIT in a META data section at the place of macro reference. GENFIT and GENFITX are used to generate the FIT at some specified location. GETFIT and SETFIT are used for retrieving and setting values of specified FIT fields.

FILE MACRO

The FILE macro is referenced in a META data section where the FIT is to be assembled; it is initialized according to specified parameters and is to be used for implicit input/output processing. None of the FILE parameters are register designators.

Macro Format:

```
FILE lfn,len,bva,fa,eea,fo
```

lfn — Logical file name, up to eight characters, starting with a letter

fa — File attributes, a set name for eight elements specifying the value for fields in the BETA(2) word of the CREATE FILE message. The fields are: IOC, MCAT, TYPE, LOK, ACS, MODE, CLAS, and UNIT. In this optional parameter, the user may specify all or only some of the fields by using null elements in the SET definition to indicate default values as follows:

IOC=n — Number of FIT generation macro references preceding this one in the same routine. An assembly error is forced if n is 16 or greater.

MCAT=0	Mass storage private file
TYPE=0	Physical data file.
LOK=0	No lockout protection.
ACS=3	Read/write access
MODE=1	Open for implicit input/output.
CLAS=0	Use security access code of this message caller
UNIT=0	

eea Optional address expression representing error exit virtual address. Default value is 0 and a fatal error.

len Optional parameter representing length of file in small pages. Default value is 40.

bva Base virtual address for mapping in the file. The parameter is optional and the default hexadecimal address is #100000000+n*(#10000000) where n is the number of FIT generation macro references preceding this one in the same routine.

fo File organization.

0	SRM-structured sequential (SS)
1	ASCII only

This parameter is optional; default value is 0. These two types of file organization are described in Appendix C of this manual.

FILEX MACRO

This macro is referenced in a META data section where the FIT is to be assembed; it is initialized according to the specified parameters and is to be used for explicit input/output processing. None of the FILEX parameters are register designators.

Except for those described below, parameters are the same as those for FILE macro.

```
FILEX lfn,len,bfl,bfa,fa,eea,fo,drb,tpm
```

bfl Optional parameter designating buffer length in number of small pages. Default value is 1.

bfa Optional, indicates starting buffer virtual page address. The default hexadecimal address is #100000000+n*(#10000000) where n is the number of FIT generation macro (GENFITX) references preceding this one in the same routine.

drb Optional parameter designating virtual bit address for directory buffer. The buffer size is preset to one page. The default bit address is determined to be the next page following the data buffer for the file.

tpm Optional parameter designating tape mode:

0	BCD (7-track tape)
1	Binary (7- or 9-track tape)
2	Binary ASCII (7-track tape)
4	Mass storage file

GENFIT MODULE

GENFIT initializes the FIT for implicit input/output.

GENFIT fit,len,bva,fa,eea,fo

The fit parameter is required; all others are optional.

fit Virtual address of the FIT for the file. The first word of the FIT contains the file name, left-justified with blank fill.

The optional parameters len, bva, fa, eea and fo are the values described under the FILE macro. However, partial default of the file attributes (fa) parameter is not possible in this case.

GENFITX MODULE

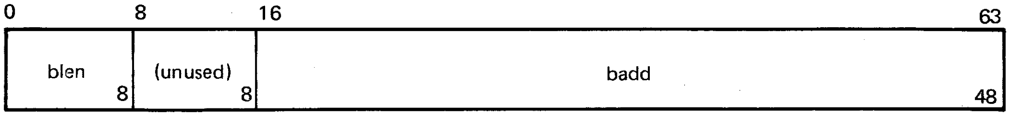
GENFITX initializes the FIT for explicit input/output.

GENFITX fit,len,bufd,fa,eea,fo,drb,tpm

The parameter fit is required; all others are optional.

fit Virtual address of the FIT for the file.

bufd Buffer descriptor word in the following format:



blen Length of virtual range, in small pages, of this buffer.

badd Starting virtual address of buffer where data transfer requests will deposit or obtain information drb Virtual address for directory buffer. Buffer size is fixed at one page. Default address is determined as the next page following the data buffer for the file.

tpm Optional parameter designating tape mode:

0	BCD (7-track tape)
1	Binary (7- or 9-track tape)
2	Binary ASCII (7-track tape)
4	Mass storage file

GETFIT MODULE

GETFIT is used to access the FIT and retrieve specified fields. Refer to the diagram of the FIT. The keywords available for use by both GETFIT and SETFIT are listed below. These keywords are used in the parameter list to designate the FIT fields to be accessed; GETFIT retrieves a designated field; SETFIT sets a value in the designated field.

Keyword	Description
LFN	Logical file name is ASCII
FA	File attributes (refers to entire word 1 containing the 8 keyword fields following)
IOC	Input/output connector number 0 to 15
MCAT	Management category for this file
TYPE	Type of. device or file
LOK	Lockout protection
ACS	File access
MODE	Mode in which file is to be used
CLAS	Security access code
UNIT	Logical unit on which this file resides
FO	File organization
LOP	Last operation
BDRY	Boundary condition
OCS	Open/close
PTL	Partial transfer length (for GETFIT only)

TPM	Tape mode
EEA	Error exit address
EDP	End-of-data pointer (for GETFIT only)
EDX	End-of-data exit address
BUF1	Buffer 1 descriptor word (refers to entire word 6 containing the 2 keyword fields following)
BUF2	Buffer 2 descriptor word
DRB	Directory buffer address

Macro Format:

GETFIT fit,keyword,reg

fit	Register containing the virtual address of the FIT for the file
keyword	The keyword which specifies the FIT field to be retrieved (refer to FIT keyword list)
reg	Specifies the register to receive the FIT field value. The value will be deposited in the register right justified with zero fill.

Routine Format:

GETFIT fit,keyword,loc

fit	Virtual address of the FIT for the file
keyword	Keyword (see list) in ASCII, left justified with blank fill
loc	Virtual address to receive FIT field, right justified with zero fill

SETFIT MODULE

SETFIT sets the specified FIT to the value given in the keyword parameter.

Macro Format:

SETFIT fit,keyword,reg

fit	Register containing virtual address of FIT for the file
keyword	Keyword (see list) specifying the FIT field to receive the value

reg Register containing keyword value right justified. Value is set in FIT field designated by keyword parameter.

Routine Format:

SETFIT fit,keyword,loc

fit Virtual address of FIT for the file

keyword Keyword (see list) in ASCII, left justified with blank fill

loc Virtual address of the new value for the specified FIT field. Value is right justified in the virtual space with zero fill.

CREATE MODULE

This module issues the CREATE FILE message to the STAR operating system. CREATE may be called as a routine.

CREATE fit,packid

fit In the macro call, this parameter designates the register containing the virtual address of the FIT; in the routine call, the parameter contains the actual FIT virtual address. Only fit is required; packid is optional.

packid In the macro, register designation containg six-character pack identifier where the file is to be created. In the routine, this parameter contains the actual value. The default value is zero, in which case, the system creates the file on a system pack having space available.

DESTROY MODULE

This module issues the system message DESTROY FILE to sever the connection between a user's program and a tape drive or mass storage device. The action releases the tape unit or mass storage space for reassignment.

DESTROY fit

fit Virtual address of FIT for the file

REDUCE MODULE

This module issues the system message REDUCE FILE LENGTH to reduce the length of an existing private mass storage file. A file can be reduced in length only if it is not active (open) for any program in the system. Reduction occurs at the largest absolute end of the file.

REDUCE fit,length

fit Virtual address of FIT for the file

length — New file length in number of words. STAR OS rounds length upward to the nearest 200 words.

CHANGE MODULE

This module issues the system message CHANGE FILENAME OR ACCOUNT to change the name or account number of an existing private file.

CHANGE fit,new,c

fit — Virtual address of the FIT for the file

new — For the macro call, parameter designates register which contains the new file name in ASCII, left-justified with blank fill. For the account number, it contains the new account number in ASCII, right-justified. In the routine call, the parameter contains the actual value.

c — Register contents or actual value must be zero to change the file name or 1 to change the account number. The parameter is optional and the default value is zero.

FILE INITIALIZATION AND TERMINATION

This category contains only the OPEN and CLOSE modules.

OPEN

OPEN issues the system message OPEN FILE, which connects the user to a pre-existing mass storage file for either explicit or implicit input/output functions.

OPEN fit,change

fit — Virtual address of FIT for the file

change — Optional parameter. If the value is zero or the parameter is omitted, the file is to be opened as specified in the MODE field in word 1 of the FIT and the system file index type remains unchanged. If the value is non-zero, the file is to be opened as specified in the MODE field of the FIT and the system file index type is to be changed as that specified in the TYPE field of word 1 of the FIT.

CLOSE

CLOSE issues the system message CLOSE FILE which severs the connection between the user program and the file. Virtual space associated with a file is no longer defined when the file is closed.

CLOSE fit,change

fit	Virtual address of FIT for the file
change	Optional parameter. If the value is zero or the parameter is omitted, no change is made in the system file index table. If the parameter value is non-zero, changes are made using the contents of the FIT. The parameter is used to signal that changes are (≠ 0) or are not (0) to be made to the file attributes (keyword FA) and base virtual address (keyword BVA) in the FIT when the file is closed.

DATA TRANSFER AND STRUCTURING

Five specific type of functions are provided by SRM through this group of modules.

Explicit input and output processing and user buffer definitions are accomplished by BUFOP, READ and WRITE.

Whole or partial sequential binary records can be read or written by specifying one user data area with GET, GETM, PUT, and PUTM. In initial implementation version of SRM, GET and PUT use implicit I/O.

Scatter/gather reading and writing of whole or partial binary records, by specifying a list of data block descriptors, is done by GETL, GETML, PUTL and PUTML.

ASCII records are transferred between the user's buffer and file by GETBCD and PUTBCD.

File demarcations are written by WEOR, WEOS, and WEOI.

BUFOP

BUFOP is used to open or close a buffer for a specified file.

BUFOP fit,bufopcode

fit — Virtual address of FIT for the file for which a buffer is to be opened or closed

bufopcode — Specifies one of the values:

1	Open buffer 1
2	Close buffer 1
3	Open buffer 2
4	Close buffer 2

READ

READ issues the system message EXPLICIT I/O and transfers data from file to user buffer.

Call Format:

READ fit,bufno,cpu

fit — Virtual address of FIT for the file to be read

bufno — Number of buffer to be opened, 1 or 2

cpu — Optional

- 0 — Do not give up central processor; return control immediately, this is the default value.
- 1 — Give up central processor until this request is complete.

WRITE

WRITE issues the system message EXPLICIT I/O and transfers data from the user buffer to the file. The designated buffer must be open before WRITE may be used.

Call Format:

WRITE fit,bufno,cpu

fit — Virtual address of FIT for file to be written

bufno — Number of buffer to be opened, 1 or 2

cpu — Optional

- 0 — Do not give up central processor return control immediately. This is the default value.
- 1 — Give up central processor until this request is complete.

GET

This module transfers all or part of the following record from the file to the user's virtual space. If the file is positioned in the middle of a record, SRM skips forward to the next record boundary before the data transfer.

GET fit,bc,da,edx

fit — Virtual address of the FIT for the file to be processed

bc — Byte count of data to be transferred

da — Data area; first virtual bit address of user area to receive data

edx — Exit virtual address to receive control when end-of-data condition occurs before the requested number of characters have been transferred. The condition may be end of record, end of section, or end of information.

Upon return from the GET routine, the file position is advanced by bc characters or to a file boundary, whichever occurs first. If the file is positioned in the middle of a record after a GET call, one or more GETM calls may be used to read more data from the same record; SKIP may be called to skip to a record boundary.

GETM

One or more GETM calls may be used to read data from the same record partially read by a GET or GETL call.

GETM fit,bc,da

The parameters have the same meanings as for the GET routine. If an end-of-data condition occurs before completion of the data transfer, control is returned to edx address specified in the previous GET or GETL call.

PUT

PUT writes all or part of the next record on the file from a specified user data area.

PUT fit,bc,da,rl

fit — Virtual address of FIT for the file to be processed

bc — Byte count of the amount of data to be transferred

da — Data area designated as the first virtual bit address of the user area holding the data to be transferred.

rl — Total number of bytes which are to form the next record. The value must be zero or it must be greater than or equal to the value given for bc.

If rl is zero, SRM automatically computes the record length when the WEOR routine is called. If rl is equal to the value for bc, the current call to PUT results in a complete record; subsequent calls to PUTM would not be permitted to write more data to the same record. A WEOR call, however, is always needed to terminate a record. If rl exceeds bc, the current call to PUT writes only part of a record; and the remainder of the record should be completed by subsequent PUTM calls. The value for rl then, sould reflect the total number of bytes written by PUT and subsequent PUTM calls that form the entire record.

PUTM

One or more calls to PUTM are used to complete the writing of a record partially written by a PUT or PUTL call.

```
PUTM fit,bc,da
```

The parameters have the same meaning as for the PUT call. If rl is non-zero in the preceding PUT or PUTL call, the number of bytes to be written by this PUTM call may not exceed the record size as defined by that rl value.

GETL

This module transfers all or part of the next succeeding record in the file to the user's virtual space. If the file is positioned in the middle of a record when GETL is called, SRM skips forward to the next record boundary before beginning the data transfer.

```
GETL fit,pl,edx
```

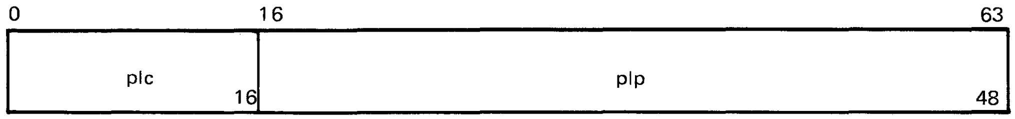
fit — Virtual address of the FIT for the file to be processed

pl — Virtual address of the parameter list descriptor word

edx — Exit address to receive control when an end-of-data condition occurs before the requested number of data blocks has been transferred.

This routine differs from GET in that a parameter list descriptor word is passed to the routine. The descriptor word points to the actual list of data descriptors and indicates the size of the list, permitting multiple data blocks to be transferred into separate and scattered areas of the user's virtual space.

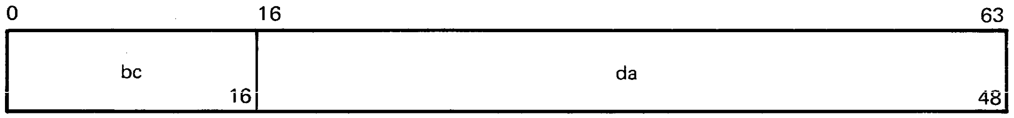
The parameter list descriptor word at address pl has the format:



plc Number of data descriptors in the parameter list

plp Address of first word in parameter list

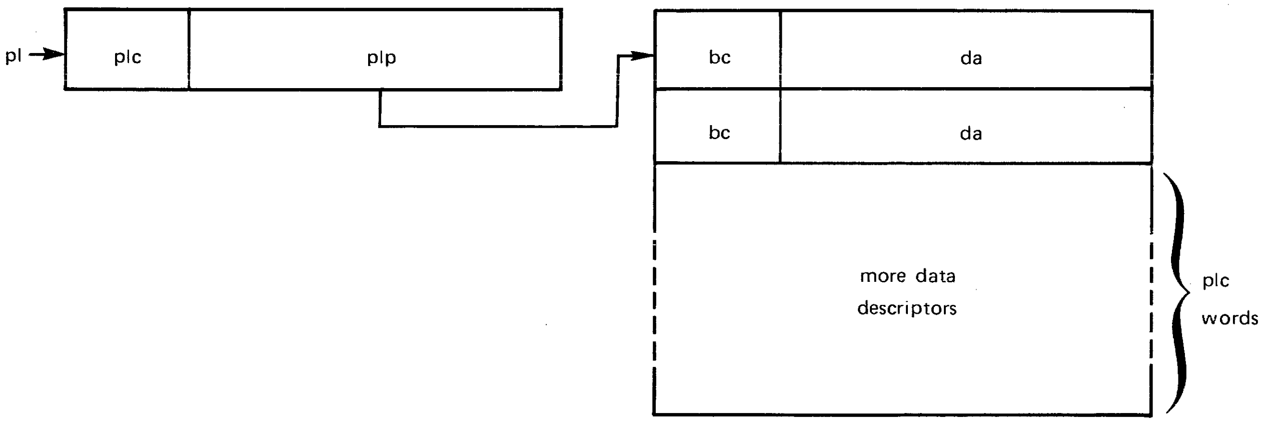
Each data descriptor defines the location and size of the data area. Thus GETL will input a stream of data from the file and scatter or distrubute the data to separate data areas in the user's virtual space. The format of each data descriptor word is:



bc Number of sequential bytes to be transferred from the file to the designated data area

da Virtual address of the data area to receive the data

The function of the parameter list descriptor word and the data descriptor list is illustrated in the following diagram:



GETML

One or more calls to GETML are used to read more data from a record partially read by a GET or GETL call.

GETML fit,pl

fit Virtual address of FIT for file to be processed

pl Virtual address of parameter list descriptor word. The format of pl and the parameter list is the same as for GETL.

If an end-of-data condition occurs before all requested data blocks have been transferred, control returns to the end-of-data exit virtual address specified for the edx parameter in the preceding GET or GETL call.

PUTL

PUTL is called to write all or part of a next record from the specified user data areas. PUT can write only from one data area; PUTL, however, can gather data from a number of separate data areas to write the record.

```
PUTL fit,pl,rl
```

fit Virtual address of the FIT for the file to be processed

pl Virtual address of the parameter list descriptor word. The format of pl and the parameter list is the same as for GETL.

rl Record length expressed as number of bytes. The value is the sum of all bc fields in the list of data descriptors referred to by the pl parameter list pointer.

PUTML

One or more calls to PUTML are used to write more data to a record partially written by a PUT or PUTL call.

```
PUTML fit,pl
```

fit Virtual address of the FIT for the file to be written

pl Virtual address of the parameter list descriptor word. The format of pl and the data descriptors forming the parameter list are the same as for GETL.

If a non-zero record length specified rl in a preceding PUT or PUTL call, the total number of bytes to be written on the record may not exceed rl.

GETBCD

The records processed by GETBCD and PUTBCD must be in ASCII code. GETBCD transfers the next ASCII record from the file to the user buffer.

GETBCD fit,da,cc

fit	Virtual address of the FIT for the file to be processed
da	Virtual address of user buffer to receive record
cc	Maximum number of characters to be transferred. This parameter is optional and has a default value of 151.

Operating within the limit set by cc, SRM transfers characters from the current file position up to and including the end-of-record marker (US, unit separator). SRM transfers control to the user's end-of-data exit address (edx) if the first character to be transferred is one of the following: FS (#IC); GS (#ID); RS (#IE).

PUTBCD

This module transfers ASCII characters.

PUTBCD fit,da,cc

fit	Virtual address of the FIT for the file to be processed
da	Virtual address of user buffer area holding ASCII character string to be transferred
cc	Number of characters to be transferred. SRM does not append ASCII control characters to the character string.

The following functions operate on the files whose addresses are specified by the fit parameter:

WEOR fit	Writes an end-of-record control word. It also computes and records the record length for the file if it was not previously specified.
WEOS fit	Writes an end-of-section control word.
WEOI fit	Writes an end-of-information control word.

The WEOR, WEOS, and WEOI routines are not applicable to ASCII files.

FILE POSITIONING

STAR Record Manager allows file positioning without doing any data transfer. Using REWIND, SKIP and SKIPS, the user can position a file to the beginning-of-information or skip forward or backward a specified number or records or sections.

REWIND

REWIND positions a tape file at the beginning of the current volume, or it positions a mass storage file at the beginning of information. If the last operation on the file is a write other than WEOI, an end of information is automatically recorded on the file before the rewind operation takes place.

REWIND fit

fit	Virtual address of FIT for the file to be processed

SKIP

SKIP positions a file to a record boundary by skipping forward or backward a specified number of full records.

Call Format:

SKIP fit,rc

fit	Virtual address of FIT for file to be processed
rc	Record count indicating the number of full records to be skipped. Value may be positive, negative, or zero.

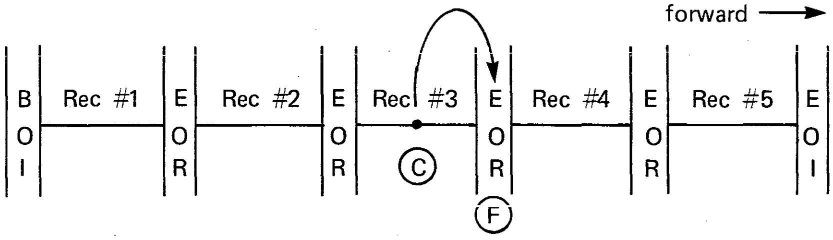
When rc is zero, the file is skipped to the next end-of-record control word if the file is positioned in the middle of a record when the call is made. No action is taken if the file is already positioned at a record boundary.

When rc is a positive non-zero value, the file is skipped forward rc full records. If the file is positioned at some point within a record when the call is made, the file is moved to the next end-of-record control word and then skipped forward rc records.

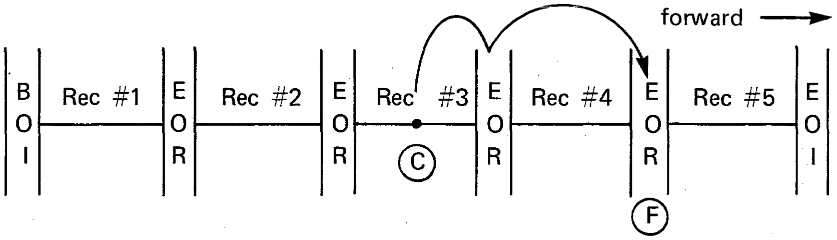
When rc is a negative value, the file is skipped backward. If the file is positioned within a record when the call is made, the file is moved backward until an end-of-control word is encountered, counting 1 for this action. If the file is at a record boundary, the file is skipped backward the number of full records specified by the negative rc.

The following examples illustrate skipping action for three values of rc when a file is positioned within a record. Ⓒ is current postion; Ⓕ is final position.

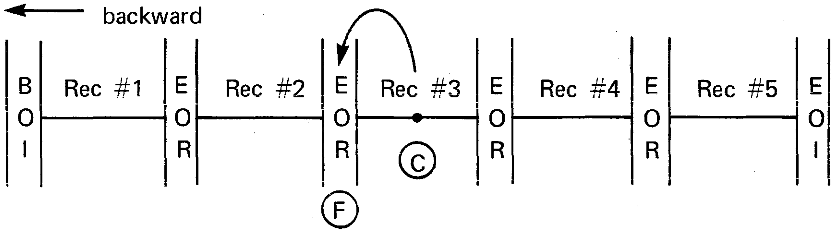
File is positioned within record 3 and SKIP call specifies rc=0:



SKIP call specifies RC=1:



SKIP call specifies RC=-1:



SKIPS

SKIPS positions a file to a section boundary by skipping forward or backward a specified number of full sections.

SKIP fit,sc

fit — Virtual address of FIT for file to processed

sc — Section count indicating number of sections to be skipped. Value may be positive, negative or zero.

When sc is zero, the file is skipped to the next end-of-section control word when the current position of the file is not at end of section. No action is taken if the file is already positioned at a section boundary.

When sc is a positive non-zero value, the file is skipped forward sc full sections. If the file is positioned at some point within a record when the call is made, the file is moved forward to the next end-of-section control word and then skipped forward sc sections.

When sc is a negative value, the file is skipped backward. If the file is positioned somewhere within the record when the call is made, the file is moved backward to the first end-of-section control word encountered, counting 1 for this action. If the file is at a section boundary, the file is skipped backward the number of full sections specified by the negative sc.

When the file is positioned within a record, SKIPS action is the same as illustrated under the SKIP call.

BKSPC

BKSPC backspaces one logical record of an ASCII file.

```
BKSPC fit
```

fit	Virtual address of the FIT for the file to be processed

If the file is positioned at beginning of information or at beginning of tape volume, SRM takes the end-of-data exit (edx) as specified in the FIT. If the file is positioned at the first character of a logical record, the file is skipped backward and positioned at the first character of the preceding logical record.

If the file is positioned somewhere other than the first character of a logical record, it is skipped backward to the first character of the same logical record where the file was initially positioned.

Each of the following ASCII characters is considered as a logical record in itself: FS (#1C); GS (#1D); RS (#1E)

MISCELLANEOUS SYSTEM INTERFACE

GIVE

GIVE presents a file to another user. The system message GIVE FILES is issued when the macro expansion code is executed.

```
GIVE fit,un
```

fit	Virtual address of the FIT for the file
un	ASCII number of user to receive the file. This parameter is optional with a default value of 999999.

TERM

TERM terminates program execution. The STAR system message TERMINATE is issued when the module is executed.

TERM msg,rest,c,rc

msg — Virtual address of a two-word space reserved for storing the Alpha words of the TERMINATE message. This parameter is required.

rest — Virtual address at which the program is to be resumed when it is restarted

c — Drop file disposition codes to be used when the program is removed from main memory. Drop file codes appear in section 5.

rc — Return code. Required parameter containing an integer value less than 256 to be passed back to the program controller.

The rest and c parameters are optional and needed only if the program is to be restarted. If rest is specified without the c parameter, the default value for c is zero, meaning that the drop file will be reserved so that the program may be restarted.

STATUS

STATUS is called to perform several functions as specified by the parameters in the call as follows:

Retrieves the system error response code for the last system message issued.

Issues the system message GIVE UP CPU UNTIL I/O COMPLETES.

Checks the system busy bit and returns the status after waiting for I/O completion.

STATUS fit,loc,cpu

fit — Virtual address of FIT for the file. If only the fit parameter is given in the call, SRM issues the system message GIVE UP CPU UNTIL I/O COMPLETES.

loc — Virtual address of word to which the system busy bit and error code is to be returned. This word has the format:

busy bit	unused 15	peripheral sys. error code 24	central sys. error code 8	response code 16

cpu — Optional parameter may have the following values:

0 — Give up central processor until all input/output is completed. This is the default value.

1 — Give up the central processor until the input/output specified in FIT is complete. If input/output is in progress, issue A GIVE UP call.

2 — Don't give up CPU but return control immediately .

TPFCN

TPFCN performs explicit tape functions provided by the file system.

```
TPFCN fit,op,fadd
```

fit — Virtual address of the FIT table for the file.

op — Keyword in ASCII, left justified with blank fill, indicating tape function to be performed. Keywords for this parameter are listed below.

fadd — Optional parameter, meaningful only for certain keywords given for the OP parameter. The values related to the keywords and the default values as shown below.

OP Keywords	Value Meaning for FADD	Default Value for FADD
REWIND	(N/A)	
UNLOAD	(N/A)	
WEOF	(N/A)	
SKIPR	Number of records to skip	1
REOF	(N/A)	
BKSPR	Number of records to backspace	1
BKSPF	Number of files to backspace	1
DENSITY	0 = 200BPI 1 = 556BPI 2 = 800BPI 3 = 1600BPI	
SEEK	(N/A)	
ERASE	Number of erasures	
STATUS	To receive status upon return	

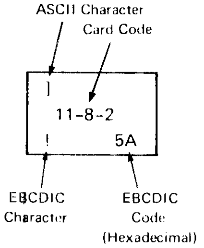
AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

	b8 b7 b6 b5	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b4 b3 b2 b1	COL ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	SP no-punch SP 40	0 0 0 F0	@ 8-4 @ 7C	P 11-7 P D7	` 8-1 ` 79	p 12-11-7 p 97	11-0-9-8-1 DS 20	12-11-0-9-8-1 30	12-0-9-1 41	12-11-9-8 58	12-11-0-9-6 76	12-11-8-7 9F	12-11-0-8 B8	12-11-9-8-4 DC
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	! 12-8-7 | 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8	a 12-0-1 a 81	q 12-11-8 q 98	0-9-1 SOS 21	9-1 31	12-0-9-2 42	11-8-1 59	12-11-0-9-7 77	11-0-8-1 A0	12-11-0-9 B9	12-11-9-8-5 DD
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	" 8-7 " 7F	2 2 2 F2	B 12-2 B C2	R 11-9 R D9	b 12-0-2 b 82	r 12-11-9 r 99	0-9-2 FS 22	11-9-8-2 CC 1A	12-0-9-3 43	11-0-9-2 62	12-11-0-9-8 78	11-0-8-2 AA	12-11-0-8-2 BA	12-11-9-8-6 DE
0 0 1 1	3	ETX 12-9-3 ETX 03	DC3 11-9-3 TM 13	# 8-3 # 7B	3 3 3 F3	C 12-3 C C3	S 0-2 S E2	c 12-0-3 c 83	s 11-0-2 s A2	0-9-3 23	9-3 33	12-0-9-4 44	11-0-9-3 63	12-0-8-1 80	11-0-8-3 AB	12-11-0-8-3 BB	12-11-9-8-7 DF
0 1 0 0	4	EOT 9-7 EOT 37	DC4 9-8-4 DC4 3C	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3	d 12-0-4 d 84	t 11-0-3 t A3	0-9-4 BYP 24	9-4 PN 34	12-0-9-5 45	11-0-9-4 64	12-0-8-2 8A	11-0-8-4 AC	12-11-0-8-4 BC	11-0-9-8-2 EA
0 1 0 1	5	ENQ 0-9-8-5 ENQ 2D	NAK 9-8-5 NAK 3D	% 0-8-4 % 6C	5 5 5 F5	E 12-5 E C5	U 0-4 U E4	e 12-0-5 e 85	u 11-0-4 u A4	11-9-5 NL 15	9-5 RS 35	12-0-9-6 46	11-0-9-5 65	12-0-8-3 8B	11-0-8-5 AD	12-11-0-8-5 BD	11-0-9-8-3 EB
0 1 1 0	6	ACK 0-9-8-6 ACK 2E	SYN 9-2 SYN 32	& 12 & 50	6 6 6 F6	F 12-6 F C6	V 0-5 V E5	f 12-0-6 f 86	v 11-0-5 v A5	12-9-6 LC 06	9-6 UC 36	12-0-9-7 47	11-0-9-6 66	12-0-8-4 8C	11-0-8-6 AE	12-11-0-8-6 BE	11-0-9-8-4 ⑁ EC
0 1 1 1	7	BEL 0-9-8-7 BEL 2F	ETB 0-9-6 ETB 26	' 8-5 ' 7D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6	g 12-0-7 g 87	w 11-0-6 w A6	11-9-7 IL 17	12-9-8 GE 08	12-0-9-8 48	11-0-9-7 67	12-0-8-5 8D	11-0-8-7 AF	12-11-0-8-7 BF	11-0-9-8-5 ED
1 0 0 0	8	BS 11-9-6 BS 16	CAN 11-9-8 CAN 18	(12-8-5 (4D	8 8 8 F8	H 12-8 H C8	X 0-7 X E7	h 12-0-8 h 88	x 11-0-7 x A7	0-9-8 28	9-8 38	12-8-1 49	11-0-9-8 68	12-0-8-6 8E	12-11-0-8-1 B0	12-0-9-8-2 CA	11-0-9-8-6 EE
1 0 0 1	9	HT 12-9-5 HT 05	EM 11-9-8-1 EM 19	) 11-8-5) 5D	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8	i 12-0-9 i 89	y 11-0-8 y A8	0-9-8-1 29	9-8-1 39	12-11-9-1 51	0-8-1 69	12-0-8-7 8F	12-11-0-1 B1	12-0-9-8-3 CB	11-0-9-8-7 EF
1 0 1 0	10 (A)	LF 0-9-5 LF 25	SUB 9-8-7 SUB 3F	* 11-8-4 * 5C	: 8-2 : 7A	J 11-1 J D1	Z 0-9 Z E9	j 12-11-1 j 91	z 11-0-9 z A9	0-9-8-2 SM 2A	9-8-2 3A	12-11-9-2 52	12-11-0 70	12-11-8-1 90	12-11-0-2 B2	12-0-9-8-4 ⎍ CC	12-11-0-9-8-2 |(LVM) FA
1 0 1 1	11 (B)	VT 12-9-8-3 VT 0B	ESC 0-9-7 ESC 27	+ 12-8-6 + 4E	; 11-8-6 ; 5E	K 11-2 K D2	[12-8-2 ¢ 4A	k 12-11-2 k 92	{ 12-0 { C0	0-9-8-3 CU2 2B	9-8-3 CU3 3B	12-11-9-3 53	12-11-0-9-1 71	12-11-8-2 9A	12-11-0-3 B3	12-0-9-8-5 CD	12-11-0-9-8-3 FB
1 1 0 0	12 (C)	FF 12-9-8-4 FF 0C	FS 11-9-8-4 IFS 1C	, 0-8-3 , 6B	< 12-8-4 < 4C	L 11-3 L D3	\ 0-8-2 \ E0	l 12-11-3 l 93	¦ 12-11 ¦ 6A	0-9-8-4 2C	12-9-4 PF 04	12-11-9-4 54	12-11-0-9-2 72	12-11-8-3 9B	12-11-0-4 B4	12-0-9-8-6 ⑂ CE	12-11-0-9-8-4 FC
1 1 0 1	13 (D)	CR 12-9-8-5 CR 0D	GS 11-9-8-5 IGS 1D	11 60	8-6 7E	M 11-4 M D4	] 11-8-2 ! 5A	m 12-11-4 m 94	} 11-0 } D0	12-9-8-1 RLF 09	11-9-4 RES 14	12-11-9-5 55	12-11-0-9-3 73	12-11-8-4 9C	12-11-0-5 B5	12-0-9-8-7 CF	12-11-0-9-8-5 FD
1 1 1 0	14 (E)	SO 12-9-8-6 SO 0E	RS 11-9-8-6 IRS 1E	12-8-3 . 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ¬ 5F	n 12-11-5 n 95	~ 11-0-1 ~ A1	12-9-8-2 SMM 0A	9-8-6 3E	12-11-9-6 56	12-11-0-9-4 74	12-11-8-5 9D	12-11-0-6 B6	12-11-9-8-2 DA	12-11-0-9-8-6 FE
1 1 1 1	15 (F)	SI 12-9-8-7 SI 0F	US 11-9-8-7 IUS 1F	/ 0-1 / 61	? 0-8-7 ? 6F	O 11-6 O D6	0-8-5 6D	o 12-11-6 o 96	DEL 12-9-7 DEL 07	11-9-8-3 CU1 1B	11-0-9-1 E1	12-11-9-7 57	12-11-0-9-5 75	12-11-8-6 9E	12-11-0-7 B7	12-11-9-8-3 DB	EO 12-11-0-9-8-7 FF

64-Character ASCII Subset

96-Character ASCII Subset

LEGEND



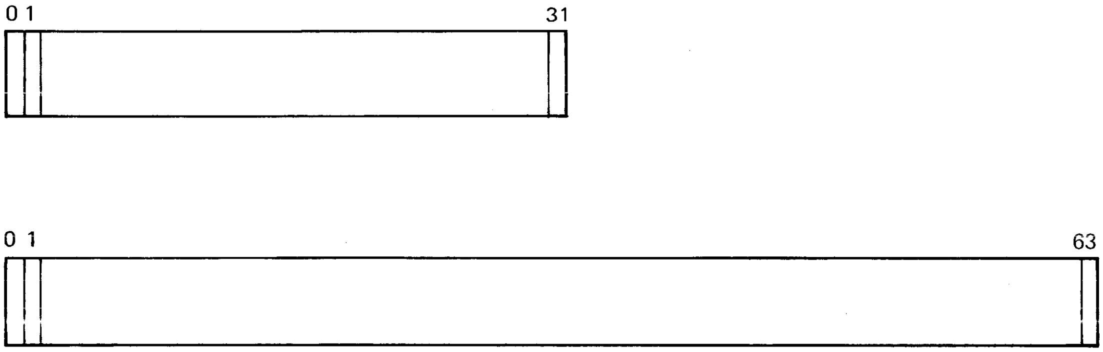
HEXADECIMAL–OCTAL CONVERSION TABLE

		First Hexadecimal Digit (Leftmost of a 2-digit number)															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Second Hexadecimal Digit (Right-most of a 2-digit number)	0	000	020	040	060	100	120	140	160	200	220	240	260	300	320	340	360
	1	001	021	041	061	101	121	141	161	201	221	241	261	301	321	341	361
	2	002	022	042	062	102	122	142	162	202	222	242	262	302	322	342	362
	3	003	023	043	063	103	123	143	163	203	223	243	263	303	323	343	363
	4	004	024	044	064	104	124	144	164	204	224	244	264	304	324	344	364
	5	005	025	045	065	105	125	145	165	205	225	245	265	305	325	345	365
	6	006	026	046	066	106	126	146	166	206	226	246	266	306	326	346	366
	7	007	027	047	067	107	127	147	167	207	227	247	267	307	327	347	367
	8	010	030	050	070	110	130	150	170	210	230	250	270	310	330	350	370
	9	011	031	051	071	111	131	151	171	211	231	251	271	311	331	351	371
	A	012	032	052	072	112	132	152	172	212	232	252	272	312	332	352	372
	B	013	033	053	073	113	133	153	173	213	233	253	273	313	333	353	373
	C	014	034	054	074	114	134	154	174	214	234	254	274	314	334	354	374
	D	015	035	055	075	115	135	155	175	215	235	255	275	315	335	355	375
	E	016	036	056	076	116	136	156	176	216	236	256	276	316	336	356	376
	F	017	037	057	077	117	137	157	177	217	237	257	277	317	337	357	377
Octal		000 – 037		040 – 077		100 – 137		140 – 177		200 – 237		240 – 277		300 – 337		340 – 377	

STAR INSTRUCTION SET (MACHINE LANGUAGE) B

The computer system uses two basic sizes of instruction words and operands: 32-bit and 64-bit words. In instruction words, the type of instruction specifies the length of the word. For most instructions that use operands, the programmer may select either 32-bit or 64-bit operands by setting or clearing a control bit in the corresponding instruction word. In either case, the bits in the operands or instruction words are numbered 0-31/63 from left to right as shown in the following diagrams.

Operands contain the least significant bits in the higher order bit positions. For example, a two's complement, 32-bit operand would contain the least significant bit in bit 31 in the figure.



Basic Instruction Word and Operand Bit Numbering

When designated portions of instruction words are undefined, the bits must be cleared (0) or the instruction will produce unpredictable results. Portions of instruction words or register contents not used are so indicated; they need not be cleared.

In the instruction descriptions, all references to a register refer to one of the 256 registers in the register file. These registers can contain operands, address modifiers, short program loops, etc. as described in appendix E, Register Conventions.

0	8	16	24 31
F	R	S	T
		undefined (must be 0's)	

INSTRUCTION WORD FORMATS

Instruction formats are divided into 12 categories. A particular format type is usually common to a group (or groups) of instructions.

FORMAT TYPES

Each format type is divided into the corresponding instruction designator portions. Most instruction formats are divided into 8-bit, designator portions which correspond to the 8-bit, character length of the computer. The format types shown below are numbered in hexadecimal.

INSTRUCTION DESIGNATORS

Following the illustrations of format types, a table lists the designators in alphabetic order. General definitions are given for the designator; definitions of corresponding designators may vary slightly with individual instructions. If the C + 1 designator is used in an instruction, the C-designator must specify an even numbered register. If the C-designator specifies an odd-numbered register, the results of the instruction become undefined.

Format 6 Used for 3E, 3F, 4D, and 4E index instructions and 2A register instruction.

0	8	16 31
F	R	I
(function)	(destination)	(16 bits)

Format 1 Used for vector, vector macro, and some non-typical instructions.

0	8	16	24	32	40	48	56 63
F (function)	G (sub-function)	X (offset for A)	A (length & base address)	Y (offset for B)	B (length & base address)	Z (Control* Vector base address)	C (length & base address)
							C + 1 (offset for C & Z)

Format 2 Used for sparse vector and some non-typical instructions.

0	8	16	24	32	40	48	56 63
F (function)	G (sub-function)	X (Order Vector length & base address)	A (base address)	Y (Order Vector length & base address)	B (base address)	Z (Order Vector length & base address)	C (result, length & base address)

Format 3 Used for logical string and string instructions.

0	8	16	24	32	40	48	56 63
F (function)	G (sub-function)	X (index for A)	A (length & base address)	Y (index for B)	B (length & base address)	Z (index for C)	C (length & base address)

Format 4 Used for some register, all monitor, 3D, and 04 non-typical instructions.

0	8	16	24 31
F (function)	R (source 1)	S (source 2)	T (destination)

Format 5 Used for BE, BF, CD,and CE index instructions and for the B6 branch instruction.

0	8	16 63
F (function)	R (destination)	I (48 bits)

Format 6 Used for 3E, 3F, 4D, and 4E index instructions and 2A register instruction.

0	8	16 31
F **(function)**	**R** **(destination)**	**I** **(16 bits)**

Format 7 Used for some branch and non-typical instructions.

0 8 16 24 31

F (function)	R	S	T

[described where used]

Format 8 Used for some branch instructions.

0 8 16 24 31

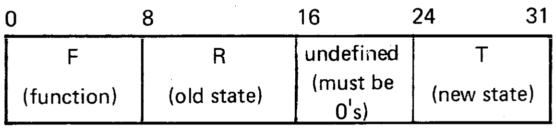
F (function)	R (register)	S (register)	T (base address)

Format 9 Used for 32-branch instruction.

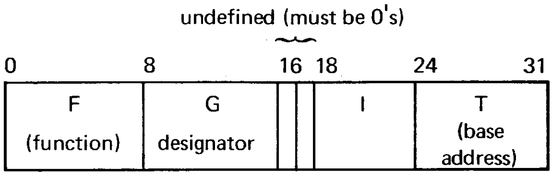
0 8 16 24 31

F (function)	G designator	S (bit test address)	T

Format A Used for some index, branch, and register instructions.



Format B Used for 33-branch instruction.



Format C Used for B0 through B5-branch instructions.

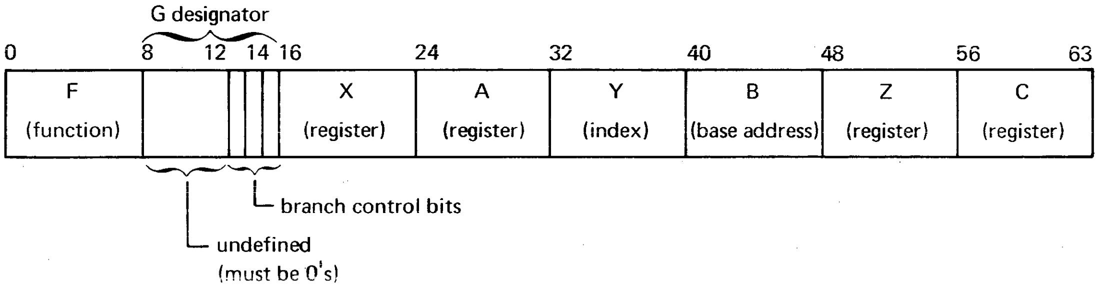


Table B-1. Instruction Designators

Designator	Format Type	Definition
A		Specifies a register that contains:
	1 & 3	Field length and base address for corresponding source vector or string field
	2	Base address for a source sparse vector field
	C	Two's complement integer in rightmost 48 bits
B		Specifies a register that contains:
	1 & 3	Field length and base address for corresponding source vector or string field
	2	Base address for a source sparse vector field
	C	Branch base address in rightmost 48 bits
C		Specifies a register that contains:
	1, 2, & 3	Field length and base address for storing result vector, sparse vector, or string field
	C	Two's complement sum of (A) + (X) in rightmost 48 bits. Leftmost 16 bits are cleared
C + 1	1	Specifies a register that contains the offset for C and Z vector fields
d	9 & B	2-bit designator specifies branch conditions for corresponding branch instructions
e	9 & B	2-bit designator specifies object bit altering conditions for corresponding branch instructions
F	1 - C	8-bit designator used in all instruction format types to specify instruction function code. This designator is always contained in the leftmost 8-bits and is expressed in hexadecimal for all instruction descriptions. The function code range is 00-FF; however, not all possible function codes are used
G	1, 2, 3, 9 B & C	8-bit designator specifies certain sub-function conditions for the corresponding instruction, including length of operands (32- or 64-bit), normal or broadcast source vectors, etc. The number of bits used in G designator varies with instructions (Tables B-3 through B-8 list bit usages within G designator field)

Table B-1. Instruction Designators (continued)

Designator	Format Type	Definition
I	5	48-bit I field functions as an index used in forming the branch address in a B6 branch instruction. In the BE and BF index instructions, I is a 48-bit operand
	6	In the 3E and 3F index instructions, I is a 16-bit operand
	B	In the 33 branch instruction, the 6-bit I-designator specifies the number of the DFB object bit used in the branching operation
R	4	In the register and 3D instructions, R specifies the register containing an operand used in an arithmetic operation
	5 & 6	In the 3E, 3F, BE, and BF index instructions, R functions as a destination register for the transfer of an operand or operand sum. In the B6 branch instruction, R specifies a register containing an item count used in forming the branch address
	7, 8, & A	R specifies registers and branching conditions described with individual instructions
S	4	In the register and 3D instructions, S specifies a register containing an operand used in an arithmetic operation
	7, 8, & 9	S specifies registers and branching conditions described with individual instructions
T		T specifies:
	4	A destination register for transfer of arithmetic results
	7, 8, 9, & B	A register containing base address and, in some cases, field length of corresponding result field or branch address
	A	A register containing old state of a register, DFB register, etc., in index, branch, or inter-register transfer operation
X	1 & 3	Specifies a register containing offset or index for vector or string source field A
	2	X specifies a register containing length and base address for order vector corresponding to source sparse vector field A
	C	In the B0-B5 branch instructions, X specifies a register containing a signed, two's-complement integer in rightmost 48 bits used as operand in branching operation

Table B-1. Instruction Designators (continued)

Designator	Format Type	Definition
Y	1 & 3	Specifies a register containing offset or index for vector or string field B
	2	Y specifies a register containing length and base address for order vector corresponding to source sparse vector field B
	C	In the B0-B5 branch instructions, Y specifies a register containing index used to form branch address
Z	1	Specifies a register containing base address for order vector, used to control result vector in field C
	2	Z specifies a register containing length and base address for order vector corresponding to result sparse vector field C
	3	Z specifies a register containing index for result field C
	C	In the B0-B5 branch instructions, Z specifies a register containing a signed, two's-complement integer in rightmost 48 bits. This integer is used as comparison operand in determining whether branch condition is met

INSTRUCTION TYPES

The instructions in this table are grouped according to ten general types:

Index (IN)	Sparse Vector (SV)	Logical String (LS)
Register (RG)	Vector Macro (VM)	Non-Typical (NT)
Branch (BR)	String (ST)	Monitor (MN)
Vector (VT)		

Unused function codes are omitted from the list. An attempt to execute an unused function code is treated as an undefined operation. More detailed information about the instructions are included in the STAR Hardware Reference Manual, No. 60256000-04.

The lists are set up in the following format:

Inst. Code	Format Type	No. of Bits in Operand		Instruction Title
(Function Code: 00 - FF_{16})	(1-C)	1	Single bit	For RG, VT, and SV instructions, U/L apply to double precision operands: U (upper or leftmost) L (lower or rightmost) 64 bits; N indicates normalized, S un-normalized.
		8	Bytes	
		32	Half words	
		64	Full words	
		E	Either 32- or 64-bit	
		B	Both 32- and 64-bit	
		NA	Not applicable	

Table B-2. Instructions Listed by Instruction Type

Index Instructions (IN)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
3E	6	64	Enter (R) with I (16)
3F	6	64	Increase (R) by I (16)
4D	6	32	Half-word Enter (R) with I (16)
4E	6	32	Half-word Increase (R) by I (16)
CD	5	32	Half-word Enter (R) with I (24)
CE	5	32	Half-word Increase (R) by I (24)
BE	5	64	Enter (R) with I (48)
BF	5	64	Increase (R) by I (48)
38	A	64	Transmit R (00-15) to T (00-15)

Table B-2. Instructions Listed by Instruction Type (continued)

Register Instructions (RG)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
2C	4	64	Logical Exclusive or R, S, to T
2D	4	64	Logical and R, S, to T
2E	4	64	Logical Inclusive or R, S, to T
30	7	64	Shift R per S to T
34	4	64	Shift R per (S) to T
6D	4	64	Insert Bits from R to T per S
6E	4	64	Extract Bits from R to T per S
40/60	4	32/64	Add U; (R) + (S) to (T)
41/61	4	32/64	Add L; (R) + (S) to (T)
42/62	4	32/64	Add N; (R) + (S) to (T)
44/64	4	32/64	Subtract U; (R) - (S) to (T)
45/65	4	32/64	Subtract L; (R) - (S) to (T)
46/66	4	32/64	Subtract N; (R) - (S) to (T)
48/68	4	32/64	Multiply U;(R) * (S) to (T)
49/69	4	32/64	Multiply L; (R) * (S) to (T)
4B/6B	4	32/64	Multiply S; (R) * (S) to (T)
4C/6C	4	32/64	Divide U; (R) / (S) to (T)
4F/6F	4	32/64	Divide S; (R) / (S) to (T)
63	4	64	Add Address R + S to T
67	4	64	Subtract Address R - S to T
58/78	A	32/64	Transmit (R) to (T)
59/79	A	32/64	Absolute (R) to (T)
51/71	A	32/64	Floor (R) to (T)
52/72	A	32/64	Ceiling (R) to (T)
5A/7A	A	32/64	Exponent of (R) to (T)
50/70	A	32/64	Truncate (R) to (T)
5B/7B	4	32/64	Pack R, S to T
5C	A	B	Extend R (32) to T (64)
5D	A	B	Index Extend R (32) to T (64)
76	A	B	Contract R (64) to T (32)
77	A	B	Rounded Contract R (64) to T (32)
7C	A	64	Length of R to T
53/73	A	32/64	Significant Square Root of (R) to (T)
10	A	64	Convert BCD to Binary, fixed length
11	A	64	Convert Binary to BCD, fixed length
54/74	4	32/64	Adjust Significance of (R) per (S) to (T)
55/75	4	32/64	Adjust Exponent of (R) per (S) to (T)
2A	6	64	Enter Length of (R) with I (16)
2B	4	64	Add to Length Field

Table B-2. Instructions Listed by Instruction Type (continued)

Branch Instructions (BR)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
20/24	8	32/64	Branch IF R = S (32/64 bit floating point)
21/25	8	32/64	Branch IF R ≠ S (32/64 bit floating point)
22/26	8	32/64	Branch IF R ⩾ S (32/64 bit floating point)
23/27	8	32/64	Branch IF R < S (32/64 bit floating point)
33	B	1	Data Flag Register Bit Branch and Alter
3B	A	64	Data Flag Register Load/Store
32	9	1	Bit Branch and Alter
36	7	64	Branch and Set (R) to Next Instruction
31	7	64	Increase (R) and Branch IF (R) ≠ 0
35	7	64	Decrease (R) and Branch IF (R) ≠ 0
09	4	64	Exit Force
B0	C	64	Index, Branch IF (A) + (X) = (Z)
B1	C	64	Index, Branch IF (A) + (X) ≠ (Z)
B2	C	64	Index, Branch IF (A) + (X) ⩾ (Z)
B3	C	64	Index, Branch IF (A) + (X) < (Z)
B4	C	64	Index, Branch IF (A) + (X) ⩽ (Z)
B5	C	64	Index, Branch IF (A) + (X) > (Z)
B6	5	NA	Branch to Immediate Address [(R) + I (48)]
2F	9	NA	Branch to [S] on condition of bit 63 in register T

Table B-2. Instructions Listed by Instruction Type (continued)

Vector Instructions (VT)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
80†	1	E	Add U; A + B → C
81†	1	E	Add L; A + B → C
82†	1	E	Add N; A + B → C
84†	1	E	Subtract U; A - B → C
85†	1	E	Subtract L; A - B → C
86†	1	E	Subtract N; A - B → C
88†	1	E	Multiply U; A * B → C
89†	1	E	Multiply L; A * B → C
8B†	1	E	Multiply S; A * B → C
8C†	1	E	Divide U; A / B → C
8F†	1	E	Divide S; A / B → C
83	1	64	Add A; A + B → C
87	1	64	Subtract A; A - B → C
98	1	E	Transmit A → C
99	1	E	Absolute A → C
91	1	E	Floor A → C
92	1	E	Ceiling A → C
9A	1	E	Exponent of A → C
90	1	E	Truncate A → C
9B	1	E	Pack A, B → C
9C	1	B	Extend A (32) → C (64)
96	1	B	Contract A (64) → C (32)
97	1	B	Rounded Contract A (64) → C (32)
93†	1	E	Significant Square Root of A → C
94	1	E	Adjust Significant of A per B → C
95	1	E	Adjust Exponent of A per B → C

†These instructions have sign control capability.

Table B-2. Instructions Listed by Instruction Type (continued)

Sparse Vector Instructions (SV)†

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
A0	2	E	Add U; A + B → C
A1	2	E	Add L; A + B → C
A2	2	E	Add N; A + B → C
A4	2	E	Subtract U; A - B → C
A5	2	E	Subtract L; A - B → C
A6	2	E	Subtract N; A - B → C
A8	2	E	Multiply U; A * B → C
A9	2	E	Multiply L; A * B → C
AB	2	E	Multiply S; A * B → C
AC	2	E	Divide U; A / B → C
AF	2	E	Divide S; A / B → C

†These instructions have sign control capability.

Vector Macro Instructions (VM)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
C0	1	E	Select EQ; A = B, item count to C
C1	1	E	Select NE; $A \neq B$, item count to C
C2	1	E	Select GE; $A \geqslant B$, item count to C
C3	1	E	Select LT; $A < B$, item count to C
DA	1	E	Sum (A0 + A1 + A2 + . . . An) to C and C + 1
DB	1	E	Product (A0, A1, A2, . . . An) to C
D5	1	E	Delta $\{A(N+1) - A(N)\}$ → C(N) X
D1	1	E	Adjust Mean $\{A(N+1) + A(N)\} /2$ → C(N)
D0	1	E	Average $\{A(N) + B(N)\} /2$ → C(N)
D4	1	E	Average Difference $\{A(N) - B(N)\} /2$ → C(N)
B8	1	E	Transmit Reverse A → C
DE	1	E	Poly Evaluation $\{A(N)\}$ per B → C(N)
DF	1	E	Interval A per B → C
BA	1	E	Transmit Indexed List → C
B7	1	E	Transmit List → Indexed C
DC	1	E	Vector Dot Product to C and C + 1

Table B-2. Instructions Listed by Instruction Type (continued)

String Instructions (ST)

Inst. Code	Format Code	No. of Bits in Operand	Instruction Title
E0	3	8	Binary Add; A + B → C
E1	3	8	Binary Subtract; A - B → C
E2	3	8	Binary Multiply; A * B → C
E3	3	8	Binary Divide; A / B → C
EC	3	8	Modulo Add A + B → C
ED	3	8	Modulo Subtract A - B → C
FB	3	8	Pack Zoned to BCD; A → C
FC	3	8	Unpack BCD to Zoned; A → C
E4	3	8	Decimal Add; A + B → C
E5	3	8	Decimal Subtract; A - B → C
E6	3	8	Decimal Multiply; A * B → C
E7	3	8	Decimal Divide; A / B → C
FA	3	8	Move and Scale; A → C
F8†	3	8	Move Bytes Left; A → C
F9†	3	8	Move Bytes Left, Ones Complement
EA	3	8	Merge per Byte Mask A, B per G → C
FD†	3	8	Compare Bytes A, B per Mask Field C
FE††	3	8	Search for Masked Key Byte; A, B per C, G
FF††	3	64	Search for Masked Key Word; A, B per C, G
D6	3	1	Search for Masked Key Bit; A, B per C, G
EE†	3	8	Translate A per B → C
EF†	3	8	Translate and Test per B → C
D7†	3	8	Translate and Mark A per B → C
EB	3	8	Edit/Mark A per B → C
E8	3	8	Compare Binary A, B
E9	3	8	Compare Decimal A, B

†Delimiters may be used with this instruction. Automatic index incrementing also takes place (see individual instruction description).

††Automatic index incrementing takes place (see individual instruction description).

Logical String Instructions (LS)

Inst. Code	Format Code	No. of Bits in Operand	Instruction Title
F0	3	1	Logical Exclusive OR A, B → C
F1	3	1	Logical AND A, B → C
F2	3	1	Logical Inclusive OR A, B → C
F3	3	1	Logical Stroke, A, B → C
F4	3	1	Logical Pierce A, B → C
F5	3	1	Logical Implication A, B → C
F6	3	1	Logical Inhibit A, B → C
F7	3	1	Logical Equivalence A, B → C

Table B-2. Instructions Listed by Instruction Type (continued)

Non-Typical Instructions (NT)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
3D	4	64	Index Multiply R * S to T
3C	4	32	Half Word Index Multiply R * S to T
5E	7	32	Load T per S, R
5F	7	32	Store T per S, R
7E	7	64	Load T per S, R
7F	7	64	Store T per S, R
12	7	64	Load Byte T per S, R
13	7	64	Store Byte T per S, R
37	A	64	Transmit Job Interval Timer to T
39	A	64	Transmit Real-time Clock to T
3A	A	64	Transmit R to Job Interval Timer
BB	2	E	Mask A, B → C per Z
BC	2	E	Compress A → C; per Z
CF†	1	E	Arithmetic Compress A → C per B
BD	2	E	Merge A, B → C; per Z
14	7	1	Bit Compress
15	7	1	Bit Merge
16	7	1	Bit Mask
17	7	8	Character String Merge
DD	2	E	Sparse Dot Product to C and C + 1
C4	1	E	Compare EQ; A = B, Order Vector → Z
C5	1	E	Compare NE; A ≠ B, Order Vector → Z
C6	1	E	Compare GE; A ⩾ B, Order Vector → Z
C7	1	E	Compare LT; A < B, Order Vector → Z
C8	1	E	Search EQ; A = B, Index List → C
C9	1	E	Search NE; A ≠ B, Index List → C
CA	1	E	Search GE; A ⩾ B, Index List → C
CB	1	E	Search LT; A < B, Index List → C
D8†	1	E	Maximum of A to C; Item Count → B
D9†	1	E	Minimum of A to C; Item Count → B
B9	1	E	Transpose/Move
18	7	8	Move Bytes Right (R) + (T) → (R) + (S) + (T)
19	7	8	Scan Right
28	7	8	Scan Equal
29	7	8	Scan Unequal
1A	7	8	Fill Field T with Byte R
1B	7	8	Fill Field T with Byte (R)
1C	7	1	Form Repeated Bit Mask with Leading Zeros
1D	7	1	Form Repeated Bit Mask with Leading Ones
1E	7	1	Count Leading Equals R
1F	7	1	Count Ones in Field R, Count to T
04	4	64	Breakpoint (maintenance)
06	7	NA	Fault Test (maintenance)

†These instructions have sign control capability.

Table B-2. Instructions Listed by Instruction Type (continued)

Monitor Instructions (MN)

Inst. Code	Format Type	No. of Bits in Operand	Instruction Title
00	4	NA	Idle
08	4	64	Input/Output per R
0C	4	64	Store Associative Registers
0D	4	64	Load Associative Registers
0E	4	64	Translate External Interrupt
0F	4	64	Load KEYS from R, Translate Address S to T
0A	4	64	Transmit (R) to Monitor Interval Timer

G DESIGNATOR BIT USAGES

The following tables show the instruction G designator bit usages in a condensed form; they provide quick look-up charts for determining the G-bit control configuration for a particular instruction. G-bit usage tables are arranged according to instruction type [vector (VT), sparse vector (SV), etc.] and according to function code within that instruction type.

Keys to abbreviations designating G-bit usage conditions are given below:

Bit	Abbreviation	Meaning
8	E	Either 32- or 64-bit operands
9	C	Control vector
10	O	Offset
11, 12	B	Broadcast
13, 14, 15	S	Sign control
13, 14, 15	I	Optional index increment
8, 9, 10, 11	D	Delimiter control
Any	X	Defined in individual instruction description

Table B-3. G-Bit Usage for Vector (VT) Instructions

Blank spaces in the tables indicate the corresponding G-bits do not apply for the particular instructions and must be 0.

	G-Bit Usage							
Function Code	8	9	10	11	12	13	14	15
80	E	C	0	B	B	S	S	S
81	E	C	0	B	B	S	S	S
82	E	C	0	B	B	S	S	S
83		C	0	B	B			
84	E	C	0	B	B	S	S	S
85	E	C	0	B	B	S	S	S
86	E	C	0	B	B	S	S	S
87		C	0	B	B			
88	E	C	0	B	B	S	S	S
89	E	C	0	B	B	S	S	S
8B	E	C	0	B	B	S	S	S
8C	E	C	0	B	B	S	S	S
8F	E	C	0	B	B	S	S	S
90	E	C	0	B				
91	E	C	0	B				
92	E	C	0	B				
93	E	C	0	B		S	S	
94	E	C	0	B	B			
95	E	C	0	B	B			
96		C	0	B				
97		C	0	B				
98	E	C	0	B				
99	E	C	0	B				
9A	E	C	0	B				
9B	E	C	0	B	B			
9C		C	0	B				

Table B-4. G-Bit Usage for Sparse Vector (SV) Instructions

	G-Bit Usage							
Function Code	8	9	10	11	12	13	14	15
A0	E					S	S	S
A1	E					S	S	S
A2	E					S	S	S
A4	E					S	S	S
A5	E					S	S	S
A6	E					S	S	S
A8	E					S	S	S
A9	E					S	S	S
AB	E					S	S	S
AC	E					S	S	S
AF	E					S	S	S

Table B-5. G-Bit Usage for Branch (BR) Instructions

Function	G-Bit Usage							
Code	8	9	10	11	12	13	14	15
B0						I	I	X
B1						I	I	X
B2						I	I	X
B3						I	I	X
B4						I	I	X
B5						I	I	X

Table B-6. G-Bit Usage for Vector Macro (VM) Instructions

Function	G-Bit Usage							
Code	8	9	10	11	12	13	14	15
B7	E				B			
B8	E	C	0					
BA	E	C	0					
C0	E	C		B	B			
C1	E	C		B	B			
C2	E	C		B	B			
C3	E	C		B	B			
D0	E	C	0	B	B			
D1	E	C	0					
D4	E	C	0	B	B			
D5	E	C	0					
DA	E	C						
DB	E	C						
DC	E	C						
DE	E	C	0	B				
DF	E	C	0					

Table B-7. G-Bit Usage for Non-Typical (NT) Instructions

Function	G-Bit Usage								Function	G-Bit Usage							
Code	8	9	10	11	12	13	14	15	Code	8	9	10	11	12	13	14	15
B9	E		X	X	X				C8	E	C	X					
BB	E			B	B				C9	E	C	X					
BC	E	C							CA	E	C	X					
BD	E			B	B				CB	E	C	X					
C4	E			B	B				CF	E				B	S	S	S
C5	E			B	B				D8	E	C				S	S	
C6	E			B	B				D9	E	C				S	S	
C7	E			B	B				DD	E			B	B			

Table B-8. G-Bit Usage for String (ST) Instructions

Function	G-Bit Usage								Function	G-Bit Usage							
Code	8	9	10	11	12	13	14	15	Code	8	9	10	11	12	13	14	15
D6	◄——— DESIGNATOR ———►								F8	D	D	D	D		I		I
D7	D	D			X				F9	D	D	D	D		I		I
EA	◄——— MASK ———►								FB	X	X						
EB	◄——— DESIGNATOR ———►								FC	X	X						
EC	◄——— MODULOS ———►								FD	D	D	D	D		I	I	
ED	◄——— MODULOS ———►								FE	◄——— DESIGNATOR ———►							
EE	D	D	D	D		I		I	FF	◄——— DESIGNATOR ———►							
EF	D	D				I											

RECORD AND FILE STRUCTURES C

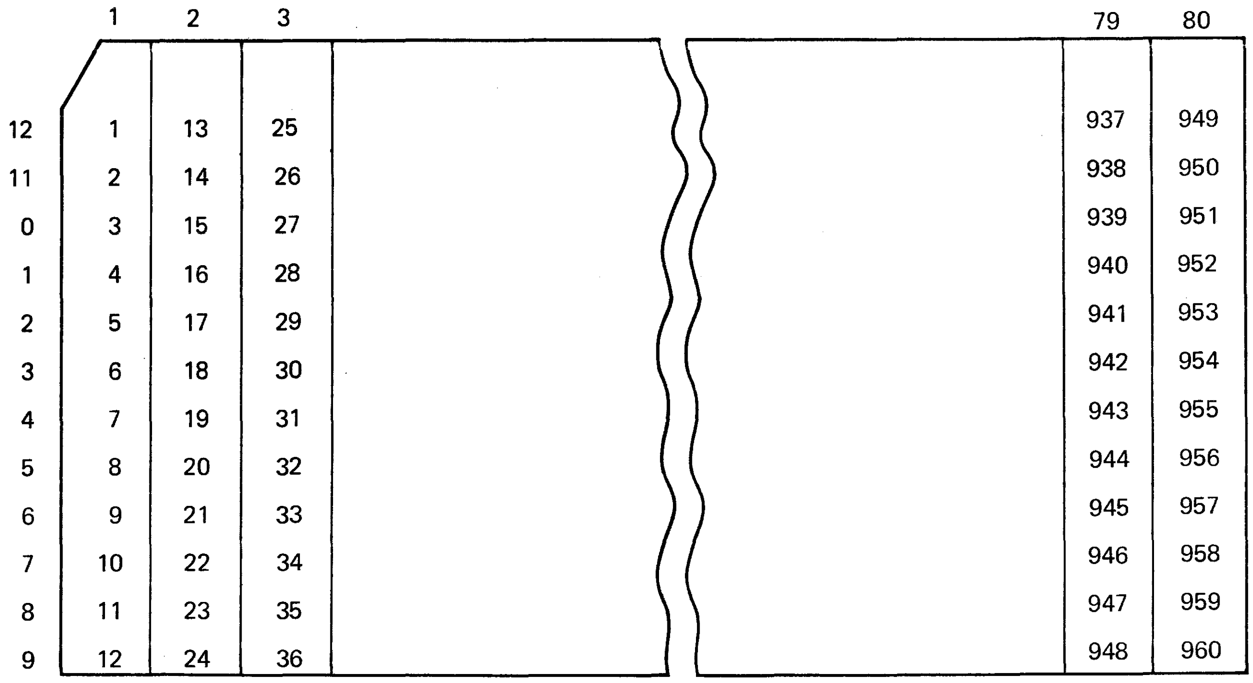
CARD FILES

Whenever a deck of cards is read into the card reader, the STAR Operating System transfers information from the deck to a disk file, and creates a file index entry that contains the file name, user number, and location of the file on disk.

The first card of each deck must be a card reader ID card. One parameter on the Card Reader ID card specifies whether the card deck is to be treated as absolute binary data or as mixed mode data. This card tells the system how to process the data and contains the information used to create the file index entry. The data of each card file starts with the card immediately following the Card Reader ID card.

ABSOLUTE BINARY DATA CARDS

The first card of an absolute deck contains the ASCII character string UNFORMAT in columns 1 to 8. Each card in the remainder of the deck is treated as a string of 960 bits of data (fifteen 64-bit words). The bits are ordered as follows:



	1	2	3	79	80
12	1	13	25	937	949
11	2	14	26	938	950
0	3	15	27	939	951
1	4	16	28	940	952
2	5	17	29	941	953
3	6	18	30	942	954
4	7	19	31	943	955
5	8	20	32	944	956
6	9	21	33	945	957
7	10	22	34	946	958
8	11	23	35	947	959
9	12	24	36	948	960

All 80 columns in an absolute binary card are treated as data. The 960 bits of data from each card are copied directly to the disk file. The end of a deck of absolute binary cards is detected when the card reader reads a second UNFORMAT card.

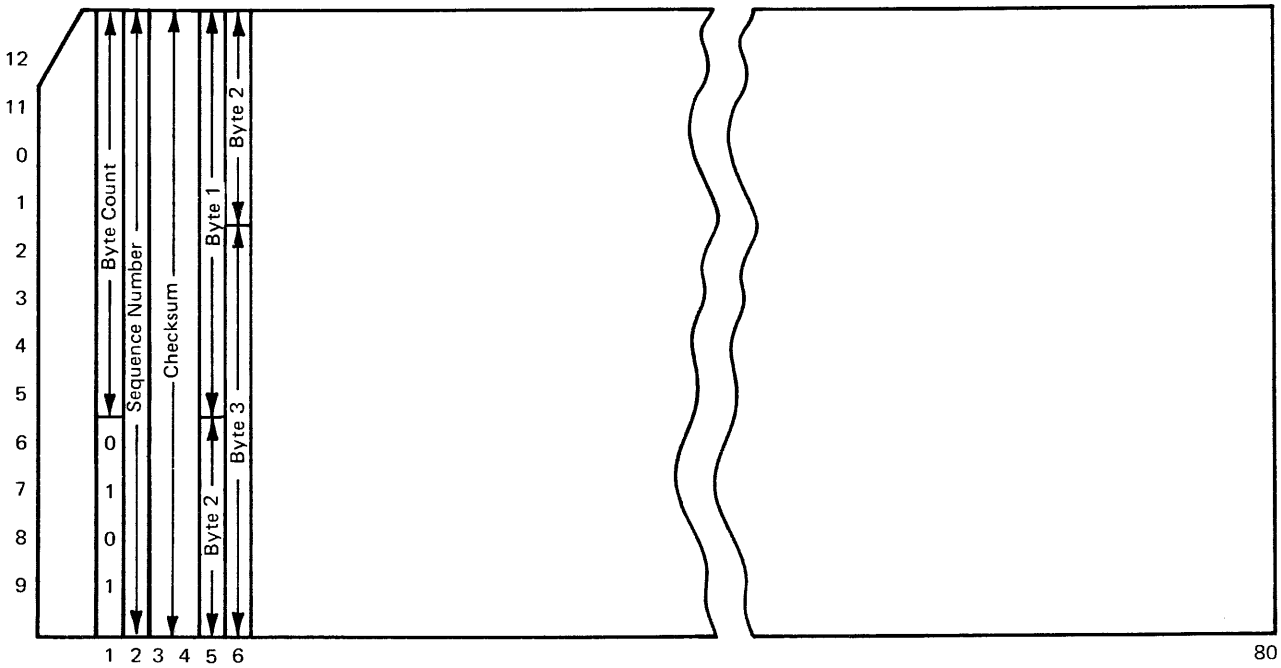
MIXED MODE CARD DECKS

A card deck processed as a mixed mode deck may contain data cards and separator cards. Separator cards are identified by combination punches in column one, as follows:

7/8/9 punch	Record separator
6/7/9 punch	Group separator
6/7/8/9 punch	File separator

Data cards in a mixed mode deck must either be in the format of STAR binary data cards or must contain valid ASCII data.

The format of a STAR binary data card is as follows:



Byte Count — Number of 8-bit bytes on this card, starting with bytes in column 5. Only the number of data bytes specified by this count are actually written as data to a disk file.

Sequence Number — Sequence number of this card

Checksum — 24-bit arithmetic sum of 8-bit data bytes on this card

Byte 1, byte 2, . . . — 8-bit data bytes

Columns 5 through 80 of a STAR binary data card may contain data. A STAR binary card contains up to 912 bits (76 columns) of data; an absolute binary card always contains 960 bits (80 columns) of data.

Every STAR binary data card has a 7/9 punch in column one to identify it and to distinguish it from ASCII data cards or separator cards in a mixed mode deck.

A deck of mixed mode cards is ended by a file separator card, or when the card reader hopper becomes empty.

RECORD STRUCTURED FILES

When a data deck is read through the card reader and processed as a mixed mode deck, the data undergoes some processing before being written to a disk file. This processing generates a record structured file on disk.

This file is processed as follows:

1. Blank fields containing two or more characters in ASCII data cards are compressed by replacing them with the ASCII control character, ESC, followed by a count of the number of blanks. The ASCII character, 0, (hexadecimal 30) is added to the count of the number of blanks to ensure that the result is beyond the range of ASCII control characters.

2. The control character, US, (unit separator) is inserted following the last character of ASCII data on each ASCII data card.

3. The control characters, GS and RS, are inserted where group separators or record separators occur. These control characters must appear in the first byte of a machine word and are followed by any ASCII data on the separator card.

4. An FS character is inserted as the last data character of the file.

5. A directory is inserted in the file after the last data word. The directory contains pointers to the start of each section of data specifying whether the data is in ASCII mode or binary mode. One entry is made in the directory for each record or group separator.

   The format of each directory entry is as follows:

nextp	unused	mode	address
16	16	8	24

nextp Ordinal of next directory entry. This number, added to the base address of the directory, points to the next directory entry.

mode Indicates the mode of the specified block.

01	Record separator
02	Group separator
03	Compressed ASCII line data
04	STAR binary byte string data
05	Unformatted data
FF	Last pointer entry

address Address of specified block or separator.

6. The last six words of the file contain a trailer. The first word of this trailer is a pointer to the directory. The last five words contain file identification information, which includes:

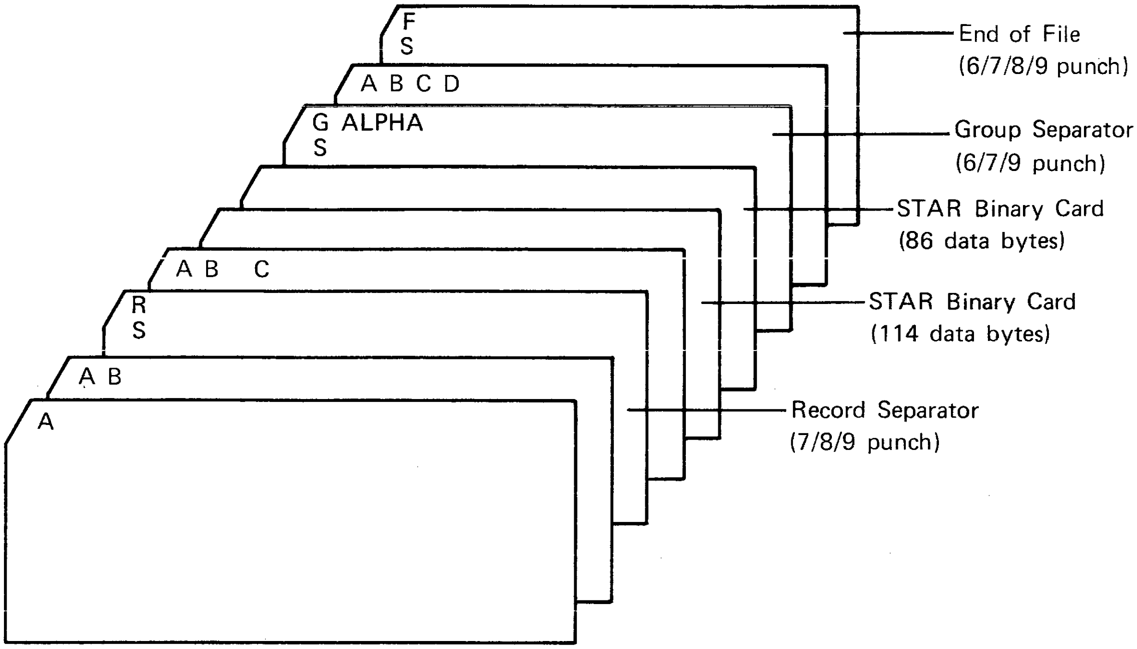
END-4	TYPE (8), LEVEL (8), USERNO (48)
END-3	File name
END-2	Unused (16), Account Number (48)
END-1	ID word 1
END-0	ID word 2.

where

TYPE is as follows:

0	Record format sequential
1	Record format virtual
2	Absolute virtual code
4	Record format sequential, batch
5	Record format virtual, batch
LEVEL	Security level
ID word 1	User identification data (from cols. 61-68 of card reader identification card)
ID word 2	User identification data (from cols. 69-76 of card reader identification card)

Page C-6 contains an example of the creation of a file from the mixed mode card deck shown on the next page.



Mixed Mode Card Deck

Word									
0	A	ESC	7F	US	A	△	B	ESC	Compressed ASCII
1	7D	US	FS	FS	FS	FS	FS	FS	
2	RS	ESC	7F	US	FS	FS	FS	FS	Record Separator
3	A	△	B	ESC	33	C	ESC	79	Compressed ASCII
4	US	FS	FS	FS	FS	FS	FS	FS	
5	byte 1	byte 2	byte 3	...	...	...	...	...	STAR Binary Data
13	byte 113	byte 114	byte 1	byte 2	...	...	...	...	
1D	...	...	...	...	...	...	byte 85	byte 86	
1E	GS	A	L	P	H	A	ESC	7A	Group Separator
1F	US	FS	FS	FS	FS	FS	FS	FS	
20	A	△	B	△	C	△	D	ESC	Compressed ASCII
21	79	US	FS	FS	FS	FS	FS	FS	
22	FS	NUL	NUL	NUL	NUL	NUL	NUL	NUL	File Separator (End of Media)
1000	0001				03	000000			Pointers
1001	0002				01	000002			
1002	0003				03	000003			
1003	0004				04	000005			
1004	0005				02	00001E			
1005	0006				03	000020			
1006	0000				FF	000022			
11FA						001000			Pointer Field Location
11FB	40-Character ASCII ID Field								ID
11FF									

PRINT FILES

When a file is given to the PRINTOUT routine, the data undergoes some processing before being printed. This processing generates a compressed ASCII file with ANSI carriage control characters. This file is processed as follows:

1. Two or more blanks (hexadecimal 20) are compressed by replacing them with the ASCII escape control character, ESC (hexadecimal 1B), followed by a count of the number of blanks. The ASCII character 0 (hexadecimal 30) is added to the count of blanks to ensure that the result is beyond the range of ASCII control characters.

2. If the internal characteristic, from a list file index, of the file is ASCII with ANSI carriage control, then the carriage control characters are assumed to be correct and are not looked at by PRINTOUT.

3. If the internal characteristic of the file is ASCII with ASCII carriage control, then the carriage control characters are changed to ANSI by PRINTOUT. The ASCII form feed control character, FF (hexadecimal 0C), as the first character of the file and after a unit separator, US (hexadecimal 1F), is replaced with the ANSI page eject control character 1 (hexadecimal 31). The ASCII single space, which is a line without the FF after the US, is changed by inserting the ANSI space one line control character, blank (hexadecimal 20), after the US.

4. The maximum length that will be printed at one time is #1000 blocks. Any file that goes over this limit will be printed in parts. If a family of files goes over this limit, the family will be divided at the end of a family member. If one file goes over the limit, it will be divided at a unit separator (end of a line) that is in the last 25 words of the #1000 blocks. If a file does not have a US in the last 25 words, the file will be divided at a random point.

5. Family files will be concatenated into one file. The file separator, FS (hexadecimal 1C), at the end of all but the last file will be replaced with a blank (hexadecimal 20). The start of all but the first file will be changed to a unit separator followed by a page eject (hexadecimal 1F31).

SEQUENTIAL RECORD FILES

STAR Record Manager (SRM) supports three sequential file organizations. The FO (file organization) field in the FIT is used to specify the type of file organization.

FO=0	SRM-structured file
FO=1	Pure ASCII file

A pure ASCII file is a string of ASCII characters terminated by an FS character (#1C). Its structure is completely specified by the appearance of the ASCII control characters US, RS, GS, and FS within the file.

SRM SEQUENTIAL STRUCTURE

SRM supports a sequential record file structure that is consistent with the concept of the ANSI FORTRAN files. Logically, a sequential record file consists of one or more sections terminated by an end-of-information mark; a section consists of none or a number of records terminated by an end-of-section mark; a record consists of none or a number of segments terminated by an end-of-record mark; a segment consists of a header followed by one or more data words.

The SRM representation of a sequential record file actually consists of data segments along with control words that provide structural information. A control word may signify the beginning-of-information, segment header, end-of-record, end-of-section, or end-of-information. An end-of-record control word may serve also as the segment-head for the first data segment of the next record.

Each control word starts on a 64-bit word boundary. The format of a control word is:

0	8	32	63
BDRY	LINK	SBC	

BDRY Boundary condition:

0 Intra-record, segment-header only

1 End-of-record (EOR)

3 End-of-section (EOS)

7 End-of-information (EOI)

#FF Beginning-of-information (BOI)

LINK Pointer to preceding BOI, EOR, or EOS control word.

SBC Segment byte count indicating number of data bytes following this control word.

The LINK field of a BOI control word is set to #FFFFFF to facilitate the detection of BOI during a backspace operation.

The following diagram illustrates the SRM sequential file structure:

Word Address	0	8	32 63	
0	FF	FFFFFF	SBC_1	BOI
	Data Segment SBC_1 Bytes			
	00	0	SBC_2	
	Data Segment SBC_2 Bytes			
	More Segments			
XX	01	0	SBC_{12}	EOR_1
	Data Segment SBC_{12} Bytes			
YY	01	XX	0	EOR_2
YY+1	03	YY	SBC_{111}	EOS_1
	Data Segment SBC_{111} Bytes			
	01	YY+1	SBC_{121}	EOR_{11}
	Data Segment SBC_{121} Bytes			
	More Segments, Records			
	03	ZZ	SBC_{112}	EOS_2
	More Sections			
	07	LINK	0	EOI

LOADER CONVENTIONS D

GENERAL TABLE STRUCTURE

The loader works with files that are composed of one or more object modules. Each object module consists of a number of standard tables; each table begins with a standard two-word header:

1	ASCII Table Name 64	
2	Length 16	Address 48

Word 1 Name of the table in ASCII

Word 2 Length Length of the table in full words

Address Bit difference between first word of the respective table and word 1 of module header table; i.e.:

Back pointer (bits) + address of first word of respective table (bits) = address of word 1 of header table (bits)

MODULE HEADER TABLE

The module header table contains general information concerning the object module and provides a linkage to all the other tables in the module.

Word		
1	△MODULE△ 64	
2	Length 16	0 48
3	Module Name 64	
4	Date + Time Created 64	
5	T Length 16	Processor 48
6	C Length 16	Data Base Length 48
7	Type 16	Pointer 48
8	Type 16	Pointer 48

Word 3 Name of module in ASCII, expressed as 8 characters, left justified and blank filled

Word 4 Date and time module was created, in packed decimal with a positive sign. Date and time format is: year, year, month, month, day, day, hour, hour, minute, minute, second, second, millisecond, millisecond, millisecond

Word 5 Word length of tables, excluding code, followed by ASCII name of processor that created module

Word 6 Word length of code, followed by bit length of data base area. The maximum size of the data base is one large page.

Word 7 & on Each word contains a table type and an address pointer to a table of that type. The pointer contains a bit address relative to the first word address of the header. By convention, the first table described is the code, and the second is the external/entry table. If HEX type is 0004, the pointer contains the bit address of the next module header table. Each table type is described in detail in this appendix

HEX Type	ASCII Name	Description
0001	CODE	Code Block Table
0002	EXT ENTR	External/Entry Table
0003	REL CODE	Code Relocation Table
0005	XFER SYM	Transfer Symbol Table
0006	SYMB TAB	Debug Symbol Table
0101	INT DATA	Interpretive Data/Initialization Table
0201	INT RELO	Interpretive Relocation/Initialization Table

CODE BLOCK TABLE

The code block table contains the executable code in the following format:

Word		
1	△△CODE△△ 64	
2	Length 16	Back Pointer 48
3 ⋮	Code	

Word 3 & on Executable code

CODE RELOCATION TABLE

This table describes relocation in the code itself.

Word		
1	REL△CODE 64	
2	Length 16	Back Pointer 48
3	nbi 16	ni 48
4	Current Base 64	
5 ⋮	I1, I2, I3, . . . In	

Word 3 nbi is number of bits per index in the bit string starting in word 5
ni is number of indexes in the string

Word 4 Current base: current bit address to which this module is relocated

Word 5 Bit string of indexes, each nbi bits long. Each index references a half word of code to be relocated relative to the base address of the code

When this table is processed, the bit base address of the code is added to the 48-bit fields pointed to by the indexes in the bit string.

EXTERNAL/ENTRY TABLE

The external/entry table contains definitions for all entry points, external symbols, and common blocks.

Word		
1	EXT△ENTR 64	
2	Length 16	Back Pointer 48
3	m 16	n 48
4	Entry Name 1 64	
5	Entry Name 2 64	
	Entry Name m 64	
	External Name 1 64	
	External Name 2 64	
	External Name (n-m) 64	
	Entry Descriptor 1 64	
	Entry Descriptor 2 64	
	Entry Descriptor n 64	
	External Descriptor 1 64	
	External Descriptor 2 64	
	External Descriptor (n-m) 64	

Word 3	m is number of entry point names in table n is number of names in table
Word 4 through 3+m	List of entry point names
Word 4+m through 3+n	List of external names
Word 4+n through 3+m+n	List of entry point descriptors
Word 4+m+n through 3+n+n	List of external descriptors

Each descriptor is of the following form:

Type	Value
16	48

Type Field	Symbol Type	Value Field
1	Entry point in code	Relative bit address in the code
2	Entry point in data	Relative bit address in the data section
3	Constant entry point	48-bit constant
14	External procedure	0
15	External datum	0
16	Common block	Bit length of the common block

ENTRY POINTS

An entry point is a named value defined in the procedure; it is to be referenced as an external by an external procedure. It may be an address in the code block, an address in the data base, or a constant value.

COMMON BLOCKS

A common block is a named alterable space referenced by one or more procedures. A common block can be initialized with relocatable data. Blank common is a common block with name of eight blanks.

EXTERNAL PROCEDURE

An external procedure reference is used in a call. Having a symbol doubly defined as a common block and external procedure is specifically allowed. All names are eight characters, left justified and blank filled.

EXTERNAL DATA

An external datum is an external that is referenced by a method other than a procedure call.

INTERPRETIVE DATA INITIALIZATION TABLE

When the loader processes information in this table, areas of static space are initialized.

Word		
1	INT△DATA 64	
2	Length 16	Back Pointer 48
3	Data Item Descriptor Data Item 64	
.	Data Item Descriptor Data Item 64	
.		
n	Data Item Descriptor Data Item 64	

Word 3 & on   Data item descriptor and item pairs, formatted as follows:

ord1	ord2	Type	Mode	Chain
16	16	8	8	16

ord1   Pseudo address vector ordinal of static space to be initialized

ord2   Pseudo address vector ordinal relative to which relocation is to be done (relocation base)

Type   Type of data item that follows

Mode 00 Values to destination
01 Values plus relocation base to destination
02 Destination plus relocation base to destination

When mode = 00, the values in the item are stored directly into the destination fields, and ord2 is ignored. When mode = 01, the relocation base is added to the values before they are stored in the destination fields. Halfword values are not defined for this case. When mode = 02, the relocation base is added to the destination fields. The value fields are absent in the various items in this case.

Chain Relative full-word count to next data item descriptor in table

Data items may be stored in one of the following formats, depending on the type:

Data Items

Item Format 1

<table>
<tr><td>Length 16</td><td>Relative Address 48</td></tr>
<tr><td colspan="2">Value 64</td></tr>
</table>

Item Format 2

<table>
<tr><td>Length 16</td><td>Relative Address 48</td></tr>
<tr><td colspan="2">Value 64</td></tr>
<tr><td>Length2 16</td><td>Bit String 48</td></tr>
</table>

Item Format 3

<table>
<tr><td>Length 16</td><td>Relative Address 48</td></tr>
<tr><td colspan="2">Value 64</td></tr>
<tr><td>nbi 16</td><td>ni 48</td></tr>
<tr><td colspan="2">Bit String 64</td></tr>
</table>

INITIALIZATION TYPES

Type	Description	Data Item Format
1	Full-Word Broadcast	1
2	Half-Word Broadcast	1
3	Full-Word Vector Transmit	1
4	Half-Word Vector Transmit	1
5	Full-Word Sparse Vector	2
6	Half-Word Sparse Vector	2
7	Full-Word Index List	3
8	Half-Word Index List	3
9	Byte String	1
A	Bit String	1
D	Nested List	Any

For each data item type, the appropriate format is applied as follows:

FULL WORD BROADCAST

Data Item Type	1
Item Format	1
Length	Full word vector length
Value	A full word to be stored in consecutive full words starting at the relative address in the section of static space

HALF WORD BROADCAST

Data Item Type	2
Item Format	1
Length	Half-word vector length
Value	A left justified half-word to be stored in consecutive half-word locations starting at the relative bit address

FULL WORD VECTOR TRANSMIT

Data Item Type	3
Item Format	1
Length	Full-word vector length
Value	Full-word vector to be transmitted to the relative address in control section

HALF WORD VECTOR TRANSMIT

Data Item Type	4
Item Format	1
Length	Half-word vector length
Value	Half-word vector to be transmitted to the relative address in control section

FULL WORD SPARSE VECTOR

Data Item Type	5
Item Format	2
Length	Number of values in item
Value	Full-word values
Length2	Length of control vector
Bit String	Control vector of length length2

HALF WORD SPARSE VECTOR

Data Item Type	6
Item Format	2
Length	Number of values in item
Value	Left justified half word vector
Length2	Length of control vector
Bit String	Left justified control vector

FULL WORD INDEX LIST

Data Item Type	7
Item Format	3
Length	Number of values in item
Value	Full word values
nbi	Number of bits per index
ni	Number of indexes
Bit String	A bit string of ni indexes; each index is nbi bits long and contains a full-word count

HALF WORD INDEX LIST

Data Item Type	8
Item Format	3
Length	Number of values in item
Value	A left justified half-word vector
nbi	Number of bits per index
ni	Number of indexes
Bit String	A bit string of indexes; each index is nbi bits long and contains a half-word count

BYTE STRING

Data Item Type	9
Item Format	1
Length	Number of bytes in value field
Value	A left justified byte string

BIT STRING

Data Item Type	A
Item Format	1
Length	Number of bits in value field
Value	A left justified bit string

NESTED LIST

<table>
<tr><td>ORD1
16</td><td>ORD2
16</td><td>TYPE1
8</td><td>MODE
8</td><td>CHAIN1
16</td></tr>
<tr><td>LENGTH1
16</td><td colspan="4">RBA
48</td></tr>
<tr><td>NI2
16</td><td colspan="4">NITER
48</td></tr>
<tr><td>NI1
16</td><td>16</td><td>TYPE2
8</td><td colspan="2">CHAIN2
24</td></tr>
<tr><td>LENGTH2
16</td><td colspan="4">48</td></tr>
<tr><td colspan="5">VALUE
64</td></tr>
<tr><td>NI3
16</td><td colspan="4">CHAIN3
48</td></tr>
</table>

ORD1 Pseudo address vector ordinal relative to the data area to be initialized

ORD2 Pseudo address vector ordinal relative to which relocation is to be done (relocation base)

TYPE1 D-nested list

MODE 00 Value to destination
01 Value plus relocation base to destination
02 Destination plus relocation base to destination

LENGTH1 Number of nested item types that follow

RBA Relative bit address

NI1 Nested data item

NI2 Nested iteration start item

NI3 Nested iteration end item

NITER Number of times data item/items associated with this iteration start item are to be repeated

TYPE2 Any initialization type. If more than one data item in an iteration, types may not be mixed

CHAIN1 Relative full word count to next data item in nested list

CHAIN2 Length of data item in number of words

CHAIN3 0 No nested item types follow
1 More nested item types follow

LENGTH2 Half word vector length

VALUE A left justified half word to be stored in consecutive half word locations starting at the relative bit address RBA

INTERPRETIVE RELOCATION INITIALIZATION TABLE

Word		
1	INT△RELO 64	
2	Length 16	Back Pointer 48
3	Relocation Item 1 64	
	Relocation Item 2 64	
	– – – – – – –	
	Relocation Item n 64	

Word 3 & on Relocation items; item formats are similar to data initialization table formats but do not contain values

TRANSFER SYMBOL TABLE

Word		
1	XFER△SYM 64	
2	Length 16	Back Pointer 48
3	Transfer Symbol 64	

Word 3 The symbol name of the entry point to which control is to be transferred at the start of execution. The name is left justified with blank fill.

DEBUG SYMBOL TABLE

The debug symbol table contains the ASCII representation of symbols which appear in a program. It can be useful in a debugging package to allow a symbol to be referenced by name rather than by address. These tables will appear in the Error Processing Information if the compiler/assembler used is capable of generating these tables and the appropriate option is selected and used during compilation/assembly.

Word		
1	SYMB△TAB 64	
2	Length 16	Back Pointer 48
3	Number of Symbols 16	0 48
4	Symbol 1 64	
	Symbol 2 64	
	⋮	
	Symbol N 64	

Word 2 Length: length of table including the symbol definition table
Back Pointer: bit difference between word 1 of this table and word 1 of the module header table

Word 3 Number of symbols in this table

Word 4 & on — Symbols can be any of the following:

Variable or array names in ASCII; must be left-justified and blank-filled.

Statement line numbers in ASCII; must be hexadecimal values, right-justified and ASCII zero-filled.

Statement labels in ASCII. Labels which are symbolic names are stored left-justified and blank-filled; labels which are statement numbers are stored right-justified and ASCII zero-filled.

SYMBOL DEFINITION TABLE

The symbol definition table is a sub-table to the debug symbol table. It provides further definition to the debug symbols, including the type of symbol, address, and mode.

Word				
1	SYMB△DEF 64			
2	Length 16	0 48		
3	Type 16	Location 48		Symbol 1 Definition
4	Mode 16	0 32	Ordinal 16	
	Type 16	Location 48		Symbol 2 Definition
	Mode 16	0 32	Ordinal 16	
	Type 16	Location 48		Symbol N Definition
	Mode 16	0 32	Ordinal 16	

Word 3 Symbol type:

0	Unknown
1	Half-word register variable name
2	Full-word register variable name
3	Variable or array name
4	Line number
5	Label

Location:

For symbol type	Location field
1	Half-word address within register file. Since half-word values can be stored in full-word registers, location value can range to hexadecimal 1FF.
2	Full-word register number
3	Bit address relative to the start of the data base.
4	Bit address relative to the start of the code base.
5	Bit address relative to the start of the code base.

Word 4 Mode: Symbol mode, which consists of three parts: precision; description; and data type. In the case of a descriptor, P and Dtype describe the contents of the referenced vector.

P	Desc	Dtype
1	3	12

P	= 0	Precision base is 32 bit; or, irrelevant
	= 1	Precision base is 64 bit
Desc	= 0	Not a descriptor
	= 1	Vector descriptor
	= 2	Vector descriptor array
	= 4	Sparse vector descriptor
	= 5	Sparse vector descriptor array

Dtype	= 0	Unknown
	= 1	Logical
	= 2	Integer
	= 3	Real
	= 4	Complex
	= 5	Double Precision
	= 6	Character
	= 7	Bit

Ordinal: The pseudo address vector ordinal of the data base or common block

PSEUDO ADDRESS VECTOR
(Ordinal Description)

Word		
0	Code Address	64
1	Data Base Address	64
2/3	External Address 1	64
4/5	External Address 2	64
6/7	External Address 3	64
8/9		64
2n+1, 2n+2	External Address n	64

For Common:

0 16	Address 48
0 16	Bit Length 48

For External Symbols:

0 16	Entry Address 48
Data Base Length 16	Data Base 48

REGISTER CONVENTIONS E

Like the minus page and other software conventions discussed earlier in this book, STAR-OS assumes some conventions regarding the handling of the register file, the area a user program assumes to be accessible to him between machine registers 00 to FF. This area is divided into 256 registers, some of which are available to the user as buffers, and some of which have other specific purposes, by convention.

The register file is subdivided into six major areas:

Machine registers

Temporary registers

Global registers

Environment registers } Register save area

Temporary and working registers }

Unused registers

MACHINE REGISTERS

These registers include only registers 0, 1, and 2. Register 0, by convention, contains the number 0. Register 1 contains the data flag branch exit address, and register 2 contains the data flag branch entry address.

TEMPORARY REGISTERS

A user program may utilize two areas for temporary storage, addresses, or data. The two areas are from registers 3 to 13_{16}, and from 20 to the end of the register save area.

The lower area is large enough for execution of short subroutines (such as SIN, COS, etc.) completely within the temporary space, obviating the need for saving and restoring any of the caller's permanent registers when short modules are needed by a program. However, some assemblers and compilers use this area; user caution is advised. The upper area, which is large enough to hold a variety of user procedures, is never saved by the callee. The user should not expect data in these registers to be preserved across external subroutine calls.

GLOBAL REGISTERS

The contents of the global registers are universal to all programs within a specific execution/language system. The contents can be assumed by all modules within a given system and are not usually loaded, saved, or restored by called modules. The values in these registers are unique to a given operating environment; thus, if a module from a different environment is to be called, it is the caller's responsibility to establish the correct values for the callee in the proper registers.

Register 14 contains the constant 20 in the lower portion of the register.

Register 15 contains the constant 1A in the lower portion of the register.

Register 16 contains the constant 1.

Register 17 contains the parameter descriptor. The number of parameters being passed during a call is contained in the first 16 bits; the address of the parameter list is stored in the remaining 48 bits. If the address portion contains a zero, the parameters, if any, are passed through the register file.

Registers 18 and 19 contain function results obtained from a called subroutine. For example, the result of a trigonometric or exponential function would be placed in register 18. Register 19 could be used when a result has two components: for example, the imaginary part of a complex number whose real part is returned to register 18.

ENVIRONMENT REGISTERS

The environment registers consist of the minimum set needed to support the sharing of code in a virtual system and the general requirements of recursive, re-entrant execution with dynamic linking. The environment registers include:

1A Return Register. Contains the bit address of the location in the caller to which the callee normally returns.

1B Dynamic Stack Pointer. Contains the bit base address of the next available free location in the dynamic stack. The dynamic stack pointer is always advanced prior to storing data into that region or before addresses pointing to that region are calculated.

1C Current Stack Pointer. Contains the length and bit base address of the region in the dynamic stack for storing the register file. The minimum length of that region will be the number of environment registers plus the number of registers needed for dynamic working storage for the program. During call sequences, the caller will set the length portion of the current stack pointer to the number of registers to be saved by the callee. The current stack pointer is set up by the caller, but the callee uses a swap to save the registers. A minimum of six registers can be saved (the number of environment registers).

1D Previous Stack Pointer. Contains the number of registers and bit base address where the caller's registers have been saved. The callee's previous stack pointer is an exact copy of the caller's current stack pointer.

1E Link Register. Contains the length and bit base address of the static space allocated to the module by the loader. The caller passes the callee the address of the callee's static space in the link register.

1F This register is reserved for future system use.

The environment registers are used in two areas of a code block module, called prologue and epilogue. Instructions in prologue and epilogue are inserted into the executable code by the assembler or compiler to ensure the caller's register file is saved when an external routine is called. They are discussed later in more detail.

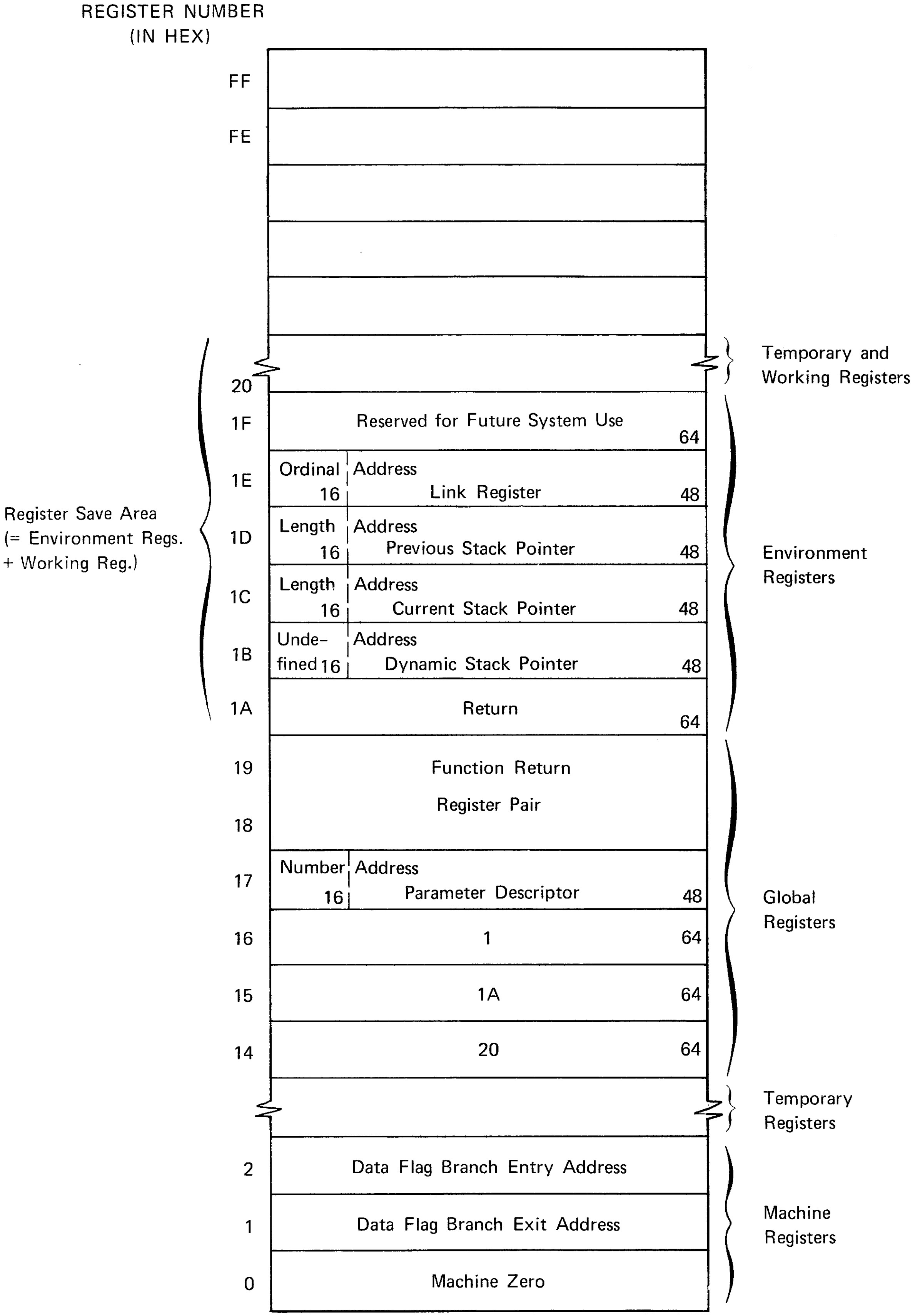
REGISTER SAVE AREA

These registers include the environment registers and the working registers. This space is saved and restored by called procedures; therefore it is the space where permanent variables and addresses should be stored. The length of this area depends on how much space has been allocated by the caller as working registers, which will contain other information the user deems necessary to save. Allocation of environment registers at the beginning of the space ensures that they will appear at the beginning of every stack, facilitating unstacking or stack searching procedures needed for block structured languages, as well as non-standard FORTRAN call/return usage. The working registers follow the environment registers.

The information in the register save area is used in the following manner: When a program in process calls an external program, the prologue of the called program executes code to save the callers's register file in dynamic space. It then places the current stack pointer in the previous stack pointer, and also places the dynamic stack pointer in the current stack pointer, and sets the dynamic stack pointer to the next free location. Finally, the prologue loads the called program's register file from static space.

When one program calls another, it uses some dynamic space to contain the status of the register file at the time the next program was called, together with linking information required to return to the calling program. In the normal sequence, dynamic space use increases until the lowest level called program has been executed; then, as the returns are encountered, the space is made available in reverse order to the calls.

Some programs can perform their tasks entirely within the temporary registers, and do not call other programs. Such routines need not contain a prologue and may be assembled or compiled to omit it.


REGISTER NUMBER
(IN HEX)
FF
FE
20
1F
Reserved for Future System Use
64
1E
Ordinal
16
Address
Link Register
48
1D
Length
16
Address
Previous Stack Pointer
48
1C
Length
16
Address
Current Stack Pointer
48
1B
Unde-
fined 16
Address
Dynamic Stack Pointer
48
1A
Return
64
19
Function Return
Register Pair
18
17
Number
16
Address
Parameter Descriptor
48
16
1
64
15
1A
64
14
20
64
2
Data Flag Branch Entry Address
1
Data Flag Branch Exit Address
0
Machine Zero
Register Save Area
(= Environment Regs.
+ Working Reg.)
Temporary and
Working Registers
Environment
Registers
Global
Registers
Temporary
Registers
Machine
Registers

SUBROUTINE LINKAGE CONVENTIONS

CALL SEQUENCE

The standard sequence of an external procedure call is as follows:

78xx001E Load the link register with the address of the callee data base. (xx = *ep*+1)

78yy0017 Load parameter descriptor register.

361A00zz Branch to the entry point of the called procedure and set a return location (contained in the return register).

(In the above instructions, 17, 1E, and 1A are the register numbers of the parameter descriptor register, the link register, and the return register, respectively; xx contains the callee data base address, yy contains a descriptor of the parameter list, and zz contains the procedure entry point address.)

SAMPLE PROLOGUE SEQUENCE

The prologue of the called procedure includes the following functions:

7D00151C Starting with the register number specified by register 15 (1A), save the registers specified in the stack of the caller.

781C001D Save the current stack pointer in the previous stack pointer.

781B001C Save the dynamic stack pointer in the current stack pointer.

2A1Cxxxx Set the number of registers to be saved by a called program (xxxx = number of working registers of callee plus six environment registers).

3F1Byyyy Increment the dynamic stack pointer by the number of callee registers to be saved (yyyy = xxxx*64).

2A1Ezzzz Enter for zzzz- number of registers to be loaded for callee execution.

7D1E1400 Load the register file with data from the callee data area, starting with the register number specified by register 14 (20).

EPILOGUE SEQUENCE

The epilogue of the called procedure should be as follows:

7D1D1500 Using the length and address of the previous stack pointer, restore the register file from the callee's current stack, starting with register 1A (the environment registers).

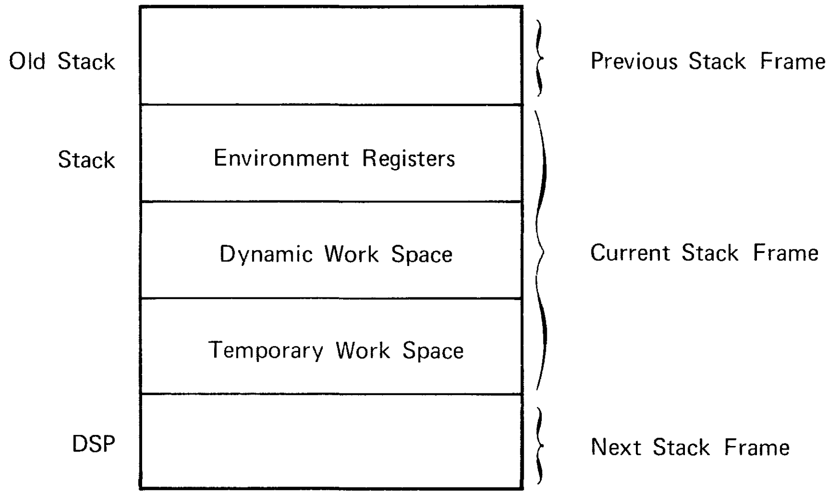
3340001A Jump to the return address specified in register 1A.

A non-normal return is carried out in similar fashion, except the values of the old stack pointer and the return register (1D and 1A respectively) will be obtained from known variables.

NOTES

1. Addresses established at execute time are stored in free space and the procedure, itself, is not modified; therefore, the procedure may be maintained in write protected storage.

2. The register file is stored and saved in a conventional chained stack, allowing creation of dynamic storage for block structured languages such as PL/1 and ALGOL. Environment registers are saved beginning at the top of a stack frame (prologue of caller); thus a stack frame appears as follows:



The initial size of a frame does not include temporary work space. Any time temporary work space is needed, the program can increment the DSP and obtain space. An entire frame disappears when a return is made to a calling program.

3. The number of registers to be saved is set by the caller. The save is performed in the prologue of the callee. If the caller can do all its work within temporary registers, the registers need not be saved. This is true only for modules that never call external modules.

4. The address contained in the DSP register (1B) must be an even 64-bit word address whenever a swap (7D) instruction is executed.

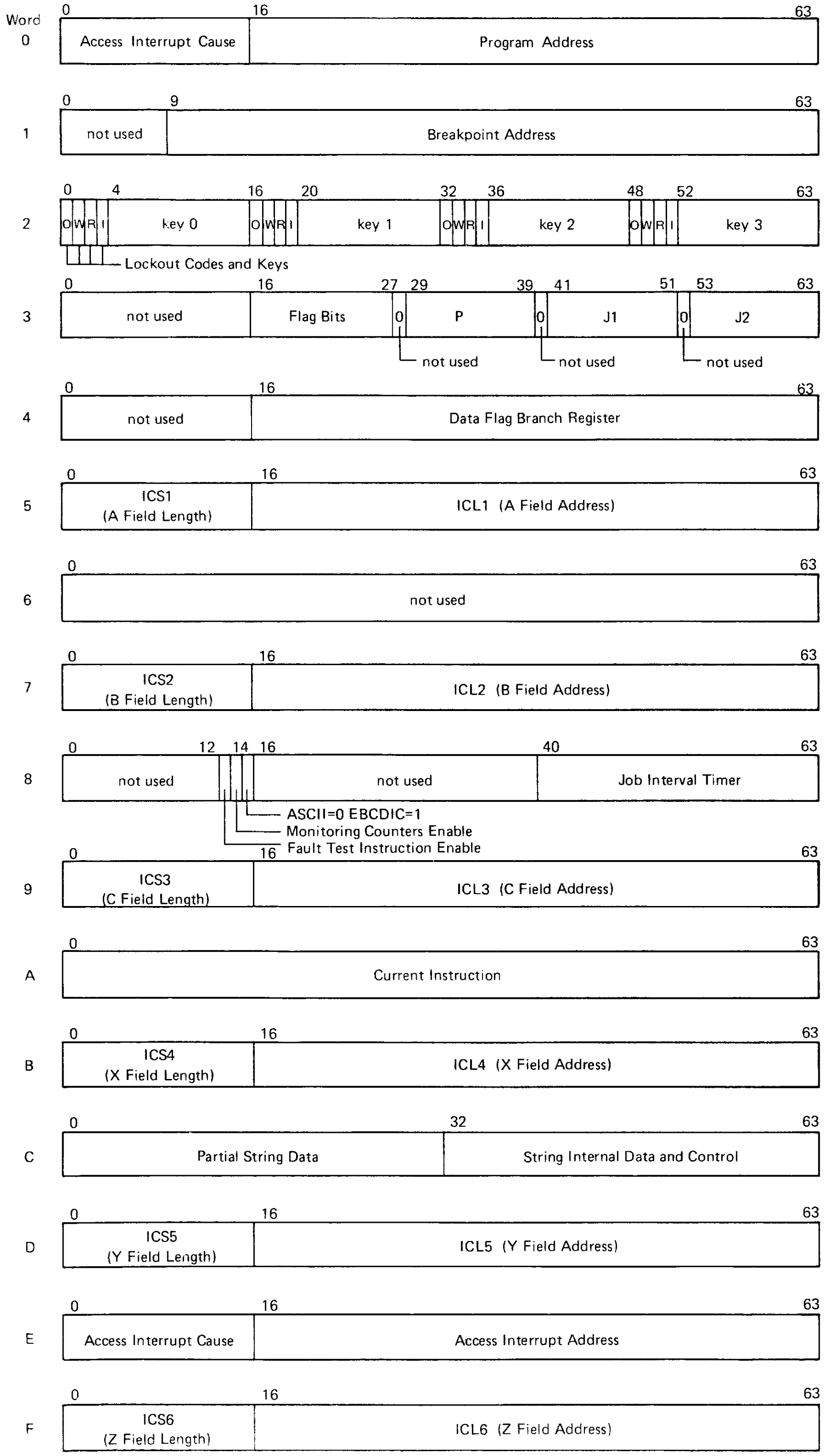
STAR-100 INVISIBLE PACKAGE F

The invisible package is a hardware convention that contains the address and control information for the corresponding job. Thus, each job is associated with an invisible package. When the CPU switches from monitor mode to job mode, the invisible package for the corresponding job is automatically loaded from main memory, beginning at the address assigned by the monitor. The invisible package data is loaded into the appropriate registers in the CPU. Thus, the appropriate invisible package provides the necessary control and address information to either begin a new job or continue a job that was interrupted during execution.

When the CPU switches from the job mode to the monitor mode, as in the case of an interrupt, the contents of the corresponding registers are automatically stored in main memory as the invisible package for that job.

X denotes variable address bit. Because the fixed portion of the absolute word address is divided within the hexadecimal character, bits 52-55 are shown as their binary equivalents.

INVISIBLE PACKAGE CONTENTS



Word 2 Lockout Codes and Keys

O =	Must be zero
W =	Lockout CPU write operations
R =	Lockout CPU read operations
I =	Lockout CPU instruction references

Word 3 Flag Bits

16	Flag 0
17	Flag 1
18	Flag 2
19	Flag 3
20	Interrupt flag
21	Not used
22	Load/store 1
23	Load/store 2
24	Subfunction bit 0
25	Subfunction bit 1
26	Subfunction bit 2
27	Subfunction bit 3

Word E InterruptCause

0-11	Zeros
12	Word not in page table
13	Write operand violation
14	Read operand violation
15	Read instruction violation

PROGRAM STATES G

Program states carried in a field in the descriptor block usually indicate the current disposition of the program.

Mnemonic	Program Is:
RUNNING	In the alternator loop.
WAIT ALT	Waiting for an alternator slot.
WAIT TPE	Waiting on tape assignment.
WRT CNTR	Waiting for controller to get on disk.
WRT CNTE	Waiting for controllee to get on disk.
RCV CNTR	Waiting for message from controller.
RCV CNTE	Waiting for message from controllee.
RCV PDP	Waiting for message from the PDP-10
SND CNTR	Waiting to send message to controller.
SND CNTE	Waiting to send message to controllee.
SND PDP	Waiting to send message to the PDP-10
SND OPR	Waiting to send message to the operator.
SND TTY	Waiting to send message to the teletype.
DUMPING	I/O being dumped to disk.
FINISH	Finished; clean-up is in progress.
SUSPEND	Suspended.
WAIT MP	Waiting for minus page to be assigned.

Range of Codes	
00-09	Program is in alternator loop
0A-0F	Program is not in alternator, but is partially in core/drum
10-1F	Miscellaneous program wait states
20-2F	Message program wait states
30-3F	System action for program in progress
40-4F	Other

Code	Meaning
00	DB free
01	PP put in alternator slot from DBLOD queue
02	PP alternator unblocked after direct page fault
03	PP alternator unblocked after non-terminal dump
04	
05	PP alternator unblocked after new slot time
06	
07	
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	

Code	Meaning
10	
11	Waiting for alternator slot
12	Waiting for I/O device assignment
13	
14	
15	
16	
17	
18	
19	
1A	
1B	
1C	Waiting for controllee A to get on disk
1D	Waiting for controllee B to get on disk
1E	Waiting for controllee C to get on disk
1F	Waiting for controllee D to get on disk
20	Waiting for message from controller
21	Waiting for message from controllee A
22	
23	
24	
25	Waiting for message from the PDP-6
26	Waiting to send a message to controller

Code	Meaning
27	Waiting to send a message to controllee
28	Waiting to send a message to PDP-6
29	Waiting to send a message to the operator
2A	Waiting to send a message to the teletype
2B	
2C	
2D	
2E	
2F	
30	XEQ line in, DB and keys assigned; message sent to load file management. (WAIT MP)
31	Drop file created, source and drop IOC's loaded, minus page in; message to scheduler
32	
33	
34	
35	
36	
37	
38	
39	Terminate and kill all pages
3A	Break while loading
3B	PP dump, accounting finished, clean up done and #80 accounting done, dump I/O
3C	Dump finished, clean up to go
3D	Terminal dump error

Code	Meaning
3E	Terminal dump scheduled, no error
3F	Non-terminal dump scheduled
40	Suspend for time period
41	
42	
43	
44	
45	
46	
47	
48	
49	
4A	
4B	
4C	
4D	
4E	
4F	

ERROR CODES AND MESSAGES H

ERROR MESSAGES

BATCH PROCESSOR ERROR MESSAGES

CONTROL CARD ERROR (AJOB)

Batch detected a control card syntax error.

JOB FILE STRUCTURE BAD (GJOB)

Record structure directory of batch input (card reader) file does not contain an ASCII mode first entry or a separator mode (RS, GS, or FS) second entry for a job.

JOB FILE VACUOUS (JOBC)

Job control card record contains no ASCII information.

JOB CARD ERROR (JOBC)

Job card syntax error detected.

SYSTEM MESSAGE ERROR

```
A(1) = xxxxxxxxxxxxxxxx        A(2) = xxxxxxxx  xxxxxxxx
B(1) =                         B(2) =
B(3) =                         B(4) =
```

A system message request error was detected within BATCHPRO.

ILLEGAL FIRST CARD

File size on the LOGON/STORE card exceeds 70 blocks. Enter n.ABORT, correct the card, then reread the deck.

LOADER ERROR MESSAGES

ILLEGAL CHAR NUMBER – BCDHEX

Illegal character number in BCD to HEX, control card has an illegal hex value. Fatal error.

CR STR TOO BIG – CRACK

Character string is bigger than space allotted – cannot CRACK all, too many characters in input symbol. Fatal error.

UNDEFINED NAME OR ALREADY IN GROUP #

Grouping not done because of an undefined name or the element to be grouped is already in another group. Fatal error.

BAD ORIGIN ADDRESS FOR GROUP

When an origin address is specified, it must be on a small or large page boundary. Fatal error.

BAD FILE

The name LIBRARY is not the first word of library file. Fatal error.

ER OPENING FILE

An error was encountered in trying to open specified library input file. Fatal error.

NO ENTRY

No entry point found. Fatal error.

ER OPENING FILE

The word MODULE was not the first word of input file. Fatal error.

COMMON LENGTHS DON'T MATCH

Two common blocks of same name are not of same length. Non-fatal error.

FILE ERROR – LODER

Error in opening input files. Fatal error.

FILE ERROR – LODTTY

Error in creating output or load map output file. Fatal error.

ILLEGAL COMMAND – LODTTY

Illegal input command to loader (illegal loader option). Fatal error.

CONTROLLEE FORMAT ERROR

Error in format of controllee option format. Fatal error.

FILE ERROR – PASS2

Error in closing controllee file. Fatal error.

INT DATA ** MODE = 1 TYPE ILLEGAL

Mode 1 of an interpretive data type is illegal – probably most common with type 9 mode 1. Non-fatal error.

INT DATA ** MODE = 2 TYPE DOES NOT EXIST

Mode 2 of an interpretive data type does not exist. Non-fatal error.

MESSAGE ERROR – PUTMES

Error in message transmitted. Control card or interactive input not properly formed. Fatal error.

DEBUG ERROR MESSAGES

filename DOES NOT EXIST

Attempt to OPEN a user specified filename, produced error code indicating filename did not exist.

OVERLAP IN BV MAP FOR filename

An attempt to OPEN a user specified filename, produced a bound virtual overlap error code.

IOC FOR filename ALREADY IN USE

Self-explanatory.

filename CANNOT BE OPENED

User specified filename could not be opened for reasons other than the foregoing.

FORMAT ERROR

User input command and/or parameters are not correct.

DEBUG ERROR. TRY AGAIN. or FATAL PROBLEM. or TRY AGAIN.

These messages come up if an error occurs (through no fault of the user) when DEBUG issues one of the following system messages:

Send message to controller. (fct #14)
List controllee chain. (fct #13)
GET message from controller. (fct #16)

or if DEBUG gets lost while processing a command.

UNABLE TO MAP-IN USER MINUS PAGE or
UNABLE TO MAP-IN USER FILE

Attempt to map-in the user's minus page or bound virtual map entries, produced an error.

UNABLE TO ENLARGE DROP FILE

An error occurred during attempt to CREATE a larger drop file.

BAD COMMAND

User specified an invalid command.

EDITT ERROR MESSAGES

NO INPUT WAS SPECIFIED

Call EDITT again, specifying at least one input file name as olda parameter.

NO OUTPUT WAS SPECIFIED

Call EDITT again, specifying at least one output file name for newa parameter.

ILLEGAL FILE NAME OR COMMA OMITTED

Call EDITT again, using correct format for parameter string (olda,oldb,newa,newb).

IMPROPER FORMAT

Call EDITT again; include parameter string in proper format (olda,oldb,newa,newb).

OPEN ERROR

Files specified in parameter string do not exist. Create files first then call EDITT again.

CLOSE ERROR

System error.

SEND ERROR

System error.

GET ERROR

System error.

OLE ERROR MESSAGES

NO MODULE HEADER FOUND filename. OLE TERMINATED.

filename HEADER LENGTH CAUSES A ZERO LIBRARY

OLE TERMINATED

UNABLE TO OMIT modulename FROM LIBRARY

UNABLE TO FIND modulename's EXTERNAL ENTRY TABLE

SPECIFIED LIBRARY LENGTH TOO SMALL. OLE WILL SET LENGTH.

PARAMETER OR FORMAT ERROR FOUND

SYSTEM DETECTED ERROR

RSP CODE ________

SS CODE _________

OPERATOR COMMUNICATION ERROR MESSAGES

Message	Response From Command
ABOVE USER ACTIVE, DID NOT DELETE DIVISION	DELETE
ACCOUNT NUMBER MISSING	DELETE
ACCOUNT OR USER NUMBER DOES NOT EXIST	DELETE
ACCOUNT TABLE FULL OR USER DOES NOT EXIST	CREATE
BAD TERMINATOR	DISPLAY MEMORY
CANT DROP OPERATOR	DROP
CANT SUSPEND OPERATOR TASKS	SUSPEND/RESUME
DB ALREADY SUSPENDED	SUSPEND/RESUME
DB NOT ASSIGNED	SUSPEND/RESUME
DB NUMBER INVALID	SUSPEND/RESUME
DB WAS NOT IN A SUSPENDED STATE	SUSPEND/RESUME
DIVISION DOES NOT EXIST	DELETE
DIVISION NUMBER MISSING	DISPLAY REPOSITORY and DELETE
DIVISION NUMBER TOO BIG	DISPLAY REPOSITORY and DELETE
ERROR IN ADDRESS FIELD	ENTER MEMORY
ERROR IN DATA	ENTER MEMORY and CREATE
ERROR IN USER NUMBER	DISPLAY ACCOUNTS DISPLAY TASK DISPLAY FILES TELL DELETE CREATE CHANGE
ERROR IN TASK ID	DISPLAY TASK
FORMAT TYPE MISSING	FORMAT
ILLEGAL FUNCTION	DELETE

Message	Response From Command	
ILLEGAL RESPONSE	CREATE	
ILLEGAL USER--ABORT--	(Illegal user of operator commands)	
INVALID CHARACTERS	DATE and TIME	
INVALID DELIMITER	ENTER MEMORY DATE TIME	
INVALID DEVICE NAME	ASSIGN	
INVALID FORMAT TYPE	FORMAT	
INVALID LEVEL	BREAK and SET	
INVALID OPTION	PROTECT	
INVALID PROGRAM ID	ASSIGN	
INVALID SUB-COMMAND	DISPLAY MEMORY DISPLAY SYSTEM ENTER MEMORY TELL CREATE BREAK SET	ASSIGN DATE TIME DELETE CHANGE REPLACE
INVALID SUFFIX	BREAK and SET	
INVALID TERMINATOR	ENTER MEMORY	
INVALID UNIT NUMBER	ON and OFF	
INVALID USER NUMBER	BREAK SET REPLACE	DROP ASSIGN
INVALID USER/TTY NUMBER	DISPLAY USER and DISPLAY TTY	
MEMORY PROTECT ON	ENTER MEMORY	
MISSING ADDRESS	DISPLAY MEMORY	
NEW ACCOUNT NUMBER MISSING	REPLACE	
NO ADDRESS	ENTER MEMORY	
NO FILES FOR USER	DISPLAY FILES	
NO MORE	MORE	

Message	Response From Command
NON-DECIMAL VALUE	SET, CHANGE, and SUSPEND/RESUME
NO SUFFIX	DISPLAY TASK
NO TIME POOL FOUND	DISPLAY REPOSITORY
NO USER FOUND	DROP
OLD ACCOUNT NUMBER MISSING	REPLACE
OPTION MISSING	PROTECT
PARAMETERS MISSING	CREATE
SUB-COMMAND MISSING	TELL, DELETE, and REPLACE
TASK NOT FOUND	DISPLAY TASK, BREAK, and SET
UNIT ALREADY OFF	OFF
UNIT ALREADY ON	ON
UNIT IN USE	ON and OFF
UNIT NUMBER MISSING	ON and OFF
UNIT UNAVAILABLE	ASSIGN
USER ACTIVE, DID NOT DELETE	DELETE
USER ALREADY EXISTS	CREATE
USER HAS NO ACCOUNTS YET	DISPLAY ACCOUNTS
USER NOT FOUND	DISPLAY ACCOUNTS, DISPLAY FILES, and CHANGE
USER NUMBER MISSING	DISPLAY ACCOUNTS DISPLAY TASK DISPLAY FILES TELL DELETE CREATE CHANGE REPLACE
USER/TTY NOT FOUND	DISPLAY USER, DISPLAY TTY, and TELL
VALUE MISSING	SET and CHANGE

Message	Response From Command
VALUE OUT-OF-RANGE	SET
VALUES TOO LARGE	CREATE and CHANGE
WRONG FORMAT, LENGTH WRONG	DATE and TIME
WRONG PROGRAM ID	ASSIGN

VIRTUAL SYSTEM ERROR MESSAGES

BAD CLASS

Class specified in execute line was not I, B, P or S.

BATCH PROCESSOR RUNNING ON THAT SUFFIX

User tried to LOGON with a batch suffix or BYE to a batch suffix.

CANT DESTROY EXISTING DROP FILE

Self-explanatory.

DROP FILE IOC DOESN'T VERIFY

Either the drop file does not exist or the IOC does not match the file index.

DROP FILE TOO SMALL

New drop file will not hold existing drop file page.

INVALID USER NUMBER

User tried to use a reserved number.

NO FILE

Either the file name does not exist, or the number of characters in filename was zero or more than 8.

NON-EXECUTABLE FILE

File requested is not a virtual code file (file type other than 2).

NOT ENOUGH TIME FOR THIS JOB

Time limit specified in execute line exceeds time remaining in repository bank.

NO TL

Zero time limit (TL) specified in execute line.

NO TIME IN BANK .

Time in repository bank is reduced to zero.

OVERDRAWN

Insufficient time remains in repository bank; results from %G status request.

SAY AGAIN

Special character (sc) is not followed by system inquiry characters S, T, ?, G, I, U, OP, SU, BB, BP, or PR.

SBU MEMORY PARITY ERROR

Parity error occurred on read or write.

SEND AGAIN

The PP state at this DB entry is zero; or job class is priority and privileged job permission flag is zero; or job is currently in interrupt mode, explicit I/O interrupt has occurred.

SOURCE OR DROP FILE ANOMALY

IOC in bound implicit map is not 16 (source); or bound implicit map entry is outside of file bounds; or drop file (free space) map address overlap occurred.

SYSTEM DROP FILE CREATE ERROR

Either the disk or file index is full.

SYSTEM TABLES FULL, TRY AGAIN

The XEQ buffer table is full as more than 8 execute lines have been entered; or no DB entry can be obtained; or no user table entries are available.

TRANSMISSION PARITY ERROR

Parity error occurred on read or write.

ERROR CODES

The error codes which follow will terminate the task. Error codes are placed in word 139 (decimal) in the minus page.

RESIDENT SYSTEM ERROR CODES

5	Illegal instruction
6	No message pointer follows exit force
7	Illegal request

INPUT/OUTPUT ERROR CODES

209	No source file
210	No drop file
212	No Alpha pointer
213	Alpha out of bounds
215	No error exit address

POOL FILE MANAGER ERROR CODES

11	Pool name already attached
12	Pool name undefined
13	Already in four pools
14	Pool not attached
15	May not attach to pool; user has no access to pool
16	Undefined user number
17	Duplicate pool name
18	Unable to destroy pool
19	Pool access directory full
1A	Pool list full
1B	Invalid option
1C	Invalid pool name
1D	Not pool boss
214	Beta buffer length error

MISCELLANEOUS ERROR CODES

33	No time left

PAGER ERROR CODES

21 No available large page while processing a system large page fault.

22 No available large page for assignment while processing a user large page fault.

25 During ADVISE call, page size for requested virtual address did not match size of page found. Size of the page found is located in the core page table entry, the drop file map entry, or the bound virtual map entry for that virtual address.

27 A write violation, such as attempting to write into a read-only page, occurred for a user page while processing a system page fault.

28 A write violation occurred while processing a user page fault.

29 A user fault for a page in the virtual system address range; out of bounds memory reference.

2A During attempt to make a drop file map entry, one of the following conditions prevails: no drop file exists; not enough room in the drop file map for an entry; not enough physical disk space for the entry.

2B A large page request was made when the user's job did not allow large pages, or a large page request caused the count of large pages owned by the user to exceed the class limit or machine limit.

2C User attempted to access a non-existing library page.

USER-1 ERROR CODES

1-00 Unable to lock down core to allow I/O file transfer.

2-00 Unable to unlock core previously locked down for I/O file transfer.

3-00 Unable to read output file, either print or punch, from mass storage.

4-00 Unable to create output file, either print or punch, on service station drum.

5-00 Unable to open file located on service station drum area.

6-50 Error in trying to read an output file that was just written to the service station (Function C300).

7-xx Errors in closing card input file, specifically defined by xx:

01 I/O connector was not for mass storage.

02 I/O connector was out of range.

03 Attempt to alter public file index entry.

04 File type, access right, or lockout specified is illegal.

05 A file page is still locked down for this close.

06 Scratch or output file is open to another program of this user.

07 Invalid name for a file with management category of output.

08 Specified I/O connector was not open.

8-00 Unable to close service station input or output file.

10-xx Error in creating card input file, specifically defined by xx:

01 File already exists.

02 No available mass storage space for this file.

03 Management category is illegal.

04 Parameter or format error occurred.

05 Operator initiated tape error occurred.

06 I/O connector is already in use.

07 File index full.

08 Standby job may not create I-tape.

09 Invalid file name.

10 Response code was preset.

11 Existing drop file cannot be destroyed.

12 Illegal type field.

15 Unable to find requested packid.

21 Bound implicit map area in minus page is full.

22 Virtual address overlap.

11-00 Error occurred in trying to read input file from service station drum area.

12-00 Error in trying to de-queue input files from service station.

13-00 Error in trying to destroy input file located in service station.

14-xx Error occurred in trying to destroy a file that could not be given to a user; specifically defined by xx:

01 File name does not exist.

02 File name conflict with I/O connector.

03 File still open to another active program.

04 Parameter or format error.

05 Destruction of source or drop file was attempted.

06 Illegal device number specified.

15-00 Error in a list file index call.

16-xx Error in trying to destroy an output file after it was processed. xx is the same as for 14-xx.

17-xx Unable to start batch processing task; specific reasons are defined by xx:

01 Suffix is busy or no free suffix is available.

02 User directory not on drum.

03 File not found.

04 Non-executable file.

05 Bad class.

06 Invalid user number.

07 Not authorized to run at priority level.

18-00 Unable to queue service station for processing output files.

19-00 Error in trying to transfer card input file to mass storage.

20-xx Error in trying to give an input file to its owner; specific cause defined by xx:

01 File name already exists for this user.

02 File has same name as a public file.

03 No such file exists.

04 No such user number exists.

05 Output file is improperly named.

06 File to be given is still active.

07 User number specified is that of the public list.

08 File to be given is a source or drop file.

09 Receiver of the file has a security classification less than that of the file.

10 Pool name specified does not exist.

30 Bad userid or account number.

21-xx Warning messages, specifically defined by xx:

2 unable to reduce a service station file, the full length of the file was used (Function C312).

3 unable to queue service station file, will try again later (Function C311).

55 PRINTOUT has been given files over #1000 blocks long; these files will be printed in parts.

22-00 Unable to find zip codes during output processing (Function 9, Option 9).

23-00 Output file internal characteristic is not ANSI or ASCII (Function 9, Option C).

24-xx Unable to requeue (GIVE) output file. For xx see error response code, SS, of GIVE file (Function 8).

SYSTEM DEAD CODES

A dead code is the product of an error detected in a part of the operating system and renders the system inoperative (dead). The codes are grouped according to the part of the system affected.

RESINIT

1 Checksum error occurred in station log-on.

2 First message received from station was not log-on.

3 Checksum error occurred on station initialization response.

5 Bad response from station.

6 Page frame in system area bad.

KERNEL

11 Job illegal; p-counter address not in core.

12 Illegal instruction in virtual system.

13 Illegal instruction in resident system.

14 Invalid message indicator from virtual system.

15 Illegal virtual system call from PAGER.

16 Illegal function code from virtual system.

17 Message has invalid station number.

18 Virtual system page zero missing.

19 User's page zero missing before run.

1A KERNEL cannot find page zero for user at initialization.

1B Checksum error on station request.

1C Attempt to release page not reserved by station.

1D Currently unused.

1E Checksum error on station response.

1F Message page is not in core.

20 No virtual process matches the resume address.

21 Too many parameters in KERNEL call.

22 Illegal call for a KERNEL function.

23 Duplicate page table entry.

24 Attempt to delete a locked page.

25 List buffer overlaps a page boundary.

26 Missing page table entry.

27 Task queue overflowed.

28 Invalid service station request received.

29 Missing page table entry.

2A Overflow of message table MESTAB.

2B Currently unused.

2C User's virtual zero page not found.

SUPPORT

31 System task queue overflow.

32 Invalid message in call to FAUL RET.

33 Missing page table entry.

34 Assigned free page not in core.

35 Duplicate page table entry detected for a free page.

36 New page entry missing from page table.

37 Invalid page table entry for a delete request.

38 Missing page table entry for requested delete.

39 Missing page table entry for requested key change.

3A Call spanned more than two pages.

3B Free page list overflow.

3C Free page stack overflow.

3D Free page list overwrite.

⋮

4D Free boat not available.

VIRTUAL SYSTEM

101 Currently unused.

102 Cannot recognize USER-1 name.

103 Error from GET INPUT BUFFER call (FOOD).

104 UT entry not found upon completion of writing UDs to drum.

105 Error return on C307 call.

106 File management returned error on execute line.

107 UDSP exceeds 8 (UDSTM pointer).

108 MESP and PROG are non-zero in BYE path.

109 TTY is logged in; error and last bit set.

10A State of PP in DBMIDL queue is less than 1E (hexadecimal).

10B UT in output chain, not a break with DB number zero.

10C File name error: zero or not user 999999, 1 or 2.

10D No PP controller when not attached to TTY.

10E Job task table full.

10F DBIN or DBOUT do not point to a valid DB; or a message from a controllee is already there.

110 Error received from file management.

111 I/O call buffers full.

112 No REGTBL entry found.

113 At least one STCK interrupt should be set.

114 User not in UT.

115 MFPP full (HANDLE).

116 Error on first read of UDs.

117 Error on first write of UDs.

118 Error on second read of UDs.

119 Error on verifying UDs.

11A Division is incorrect.

11B TTY buffer page is not in page table.

11C Page not in page table.

11D No file index hole.

11E Already in file index.

11F Error on writing a public file to disk.

120 Error in creating system drum file.

121 Error in writing system pages to drum.

122 Security level incorrect or unrecognized error number from disk station.

123 Cannot find page in list range.

124 Not enough room in core to hold read buffer and drum overflow is not running.

125 Bad function code encountered upon completion of sequential I/O (greater than 4).

126 No Alpha pointer in register parameter.

127 Page not in core after fault.

128 No external in data base.

PAGER

201 Access interrupt; all cause bits are off.

202 Write/read/execute violation in system range.

203 Control returned after terminate message.

204 Input queue full; will lose KERNEL request.

205 User minus page not in core.

206 IDOVER array full; an impossible condition.

207 Upon initializing a created page, either starting address was zero or ending address is out of core.

208 Cannot find enough unlocked small pages.

209 Unlocked system page in free range is about to be written to drum.

20A Unused.

20B During attempt to charge for I/O, all slot numbers in user's DB block were found to be zero.

20C Cannot find a free entry in user's drop file map although pointers in BSMC indicate entries are available.

20D No room for a special small page to contain user's zero or minus page.

20E FWQ indicates sufficient free words, but they could not be found in I/O write queue.

20F Bad ADVISE message; number of pages or DB number is zero.

210 Cannot find a large page in order to reserve it; however, machine has one available and user limits have not been reached.

211 KERNEL request message is bad; number of pages requested is zero.

212 Access interrupt message is bad; DB number is zero.

213 Absolute address for user minus page in the alternator slot is zero.

214 Virtual system faulting for address in user range without user key.

215 No drop file entry available for modified small page (occurs only with NODRUM option).

216 Virtual system FPGE alternator bit set.

217 Virtual system NFPGE alternator bit set.

218 Cannot find large page in class.

219 Unused.

•

•

•

222 Incomplete multiple write.

VIRTUAL SYSTEM

301 XEQBUF is empty; or error response on disk read; or NEXB is greater than FMCE.

302 Unused.

303 Unused.

304 Unused.

305 Error on write disk pattern.

306 Error on delete range from core (DESTROY).

307 Unused.

308 Unused.

309 Error on disk read.

30A Error response on translation of virtual address (F007 of OPEN).

30B MP3BUF page not found.

30C Error on deletion of MP3BUF from page table.

30D Error on deletion of range from page table (MAP).

30E Error in F007.

30F Error on deletion of virtual range from page table (MAP).

310 Error on writing page to disk (MAP).

311 Error on reading page from drum (MAP).

312 Error on deletion of virtual range from drum (MAP).

313 Anomaly detected in virtual map.

314 Unused.

315 Error on deletion of range from core (CLOSE).

316 Error on translation of virtual address (CLOSE).

317 Error on deletion of range from page table.

318 Error on reading page from drum (CLOSE).

319 Error on deletion of virtual range from drum (CLOSE).

31A Error on writing page to disk (CLOSE).

31B Unused.

31C Unused.

31D Unused.

31E Unused.

31F Unused.

320 Error on ADVISE message for page.

321 Error on translation of virtual address (ADVISE).

322 Error on list drum table entry.

323 Error on reading page from drum (ADVISE).

324 Error on deletion of page from drum (ADVISE).

325 Error on deletion of range from page table (ADVISE).

326 Error occurred on writing page to disk (ADVISE).

327 Unused.

328 Unused.

329 Unused.

32A Error on writing UD to drum (GIVEF).

32B Unused.

•
•

340 A free alternator is indicated, but none is found to exist.

341 Job class of the task in the alternator cannot be identified.

342 Alternator empty but waiting tasks are not being loaded.

343 Unused.

•
•
•

350 Interrupt with no REGTBL pointer.

351 Bad previous state.

352 Unused.

•
•
•

36D Error from zero time for all division code.

36E Error on creating new UD entry (UDMOD).

36F User number check failed.

370 Unused.

•
•

400 Error on service station read.

401 Illegal program state found by loader.

402 Error in station request.

PERMANENT FILES DEAD CODES

500 Error in C500 during read of first page of PFI. (PACKSUP)

501 Error in F007. (PACKSUP)

502 Error in C500 during read of one page at a time of PFI. (PACKSUP)

503 Error in F007. (MODPFI)

504 Error in C500. (MODPFI)

505 Failed to find a page for disk read. (GETFI)

506 Failed to read a page of PFI. (GETFI)

507 Failed to release page of disk read. (GETFI)

508 Failure in C501 write. (MODPFI)

509 Error in F002. (MODPFI)

50A Unused.

⋮

530 Failure in C500 read. (PACKSUP)

531 Error return from ENTR. (PACKSUP)

532 Failure in C501 write. (PACKSUP)

533 Error returned from ENTR. (NEWAF)

534 Failure of F007. (WRTACC)

535 Failure of C501 write. (WRTACC)

536 Failure of F007. (WRTACC)

537 Failure of C501 write. (WRTACC)

538 Failure of C501 write. (WRTACC)

539 Failure of F002 delete page. (WRTACC)

540 Failure of F002 delete page. (PACKSUP)

580 Error in F007. (BADSEC)

581 Error in C500. (BADSEC)

4201 Unrecognizable pack type indicated in pack label; 841, etc. (PACKSUP)

4202 Disk not up logically; usually a virtual system problem. (MODPFI)

4203 Unused.

⋮

4221 Error in setting up DMAP indicated overlapping files on pack. Hand eliminate one file. (PACKSUP)

4223 File index is full. (GETFI)

4224 Unused.

4225 Duplicate pack labels detected. Re-label one pack and try again. (PACKSUP)

4226 Unused.

⋮

4231 PFI entry in PFI is not properly labeled; hand-change the label. (PACKSUP)

4232 Too many file fragments for dimensions of sorting array in PACKSUP. Increase dimensioned size of array and try again. (PACKSUP)

4233 Unused.

.

.

.

4244 Duplicate filenames have been entered into file for user. One file on PFI must be hand zeroed with 00001F1C groups. (TTYMES)

4245 Unused.

.

.

.

9996 Zeros in DISKREAD buffer on PFI read.

9997 Zeros in DISKREAD buffer after PFI write.

9998 Zeros in DISKREAD buffer before PFI write.

9999 Zeros in DISKREAD buffer on PFI read.

POOL FILE MANAGER DEAD CODES

600 Could not find I/O in page table. (FINDPL)

601 Could not open PLIST or PAD files in SS drum (FINDPL); or could not close PLIST or PAD. (CLPLFI)

602 Could not read PLIST file. (FINDPL)

603 Could not delete I/O pages from page table. (CLPLFI)

604 Could not write PLIST. (WRITPL)

605 Could not read PAD. (READPD)

606 Could not write PAD. (WRITPD)

ERROR PROCESSING INFORMATION I

The error processing information contained in the controllee file is used by error processing and debugging routines. The information is provided for every object module loaded to produce the controllee file, including object modules on user-specified files and required object modules for library files. The format of the error processing information area for each module resembles the format of the binary object module. (See section 7.) Information from the object module that would be useful for debugging or error processing is retained in this appendix; only the loader directive tables have been deleted. The deleted tables are included in appendix D, Loader Conventions.

Figure I-1 is a dump of a typical controllee file, illustrating the error processing information area at the end of the dumped file.

ERROR PROCESSING INFORMATION TABLES

This appendix contains the following tables and their descriptions:

Module Header Table

Code Block Table

External/Entry Table

Debug Symbol Table

Symbol Definition Table

Pseudo Address Vector Table

Each table begins with a standard two-word header. In word 1, the name of the table in ASCII code appears; in word 2, the length of the table is indicated in the left-most 16 bits while the remainder of the word contains a back pointer, set either to zero, or as indicated in the table description.

MODULE HEADER TABLE

The module header table, as it appears in the error processing information area, contains general information concerning the object module. It provides linkage to all the other tables in the module.

Word				
1	△MODULE△			64
2	Length	16	0	48
3	Module Name			64
4	Date + Time Created			64
5	T Length	16	Processor	48
6	C Length	16	Data Base Length	48
7	Ttype	16	Taddr	48
8	Ttype	16	Taddr	48

Word 3 Name of module in ASCII expressed as 8 characters, left justified and blank filled.

Word 4 Date and time module was created, in packed decimal with a positive sign. Date and time format is: year, year, month, month, day, day, hour, hour, minute, minute, second, second, millisecond, millisecond, millisecond.

Word 5 Word length of tables, excluding code, followed by ASCII name of processor that created module.

Word 6 Word length of code, followed by bit length of data base area.

Word 7 & on Each word contains a table type and an address pointer to a table of that type. The pointer is the virtual bit address of the table. By convention, the first table described is the code, and the second is the external/entry table.

The following types may appear:

HEX Type	ASCII Name	Description
0001	CODE	Code Block Table
0002	EXT ENTR	External/Entry Table
0006	SYMB TAB	Debug Symbol Table
0301	PAV	Pseudo Address Vector Table

CODE BLOCK TABLE

The code block table has a pointer in the error processing information area. The table contains relocated program code and has the following format:

Word		
1	Program Name 64	
2	Length 16	Pointer 48
3	Relocated Code	

Word 1 Program name in ASCII.

Word 2 A pointer to the beginning of the error processing information area for that program.

Word 3 & on The relocated code.

EXTERNAL/ENTRY TABLE

The external/entry table, as it appears in the error processing information area, has the following format. It contains definitions for all entry points, external symbols, and common blocks.

Word		
1	EXT△ENTR 64	
2	Length 16	Back Pointer 48
3	m 16	n 48
4	Entry Name 1 64	
5	Entry Name 2 64	
	Entry Name m 64	
	External Name 1 64	
	External Name 2 64	
	External Name (n-m) 64	
	Entry Descriptor 1 64	
	Entry Descriptor 2 64	
	Entry Descriptor n 64	
	External Descriptor 1 64	
	External Descriptor 2 64	
	External Descriptor (n-m) 64	

Word 3	m is number of entry point names in table. n is number of names in table.
Word 4 through 3+m	List of entry point names.
Word 4+m through 3+n	List of external names.
Word 4+n through 3+m+n	List of entry point descriptors.
Word 4+m+n through 3+n+n	List of external descriptors.

Each descriptor is of the following form:

Type	Value
16	48

Type Field	Symbol Type	Value Field
1	Entry point in code	Relative bit address in the code
2	Entry point in data	Relative bit address in the data section
3	Constant entry point	48-bit constant
14	External procedure	0
15	External datum	0
16	Common block	Bit length of the common block

ENTRY POINTS

An entry point is a name value defined in the procedure; it is to be referenced as an external by an external procedure. It may be an address in the code block, an address in the data base, or a constant value.

COMMON BLOCKS

A common block is a name alterable space referenced by one or more procedures. A labelled common block can be initialized with relocatable data. Blank common is a common block with name of eight blanks.

EXTERNAL PROCEDURE

An external procedure reference is used in a call. Having a symbol doubly defined as a common block and external procedure is specifically allowed. All names are eight characters, left justified and blank filled.

EXTERNAL DATA

An external datum is an external that is referenced by a method other than a procedure call.

DEBUG SYMBOL TABLE

The debug symbol table, as it appears in the error processing information area, has the following format. It contains the ASCII representation of symbols which appear in a program. The table is useful in a debugging package as it allows a symbol to be referenced by name rather than by address. This table appears in the error processing information area if the compiler/assembler used is capable of generating the table and the appropriate option is selected and used during compilation/assembly.

Word		
1	SYMB△TAB 64	
2	Length 16	Back Pointer 48
3	Number of Symbols 16	0 48
4	Symbol 1 64	
	Symbol 2 64	
	⋮	
	Symbol N 64	

Word 2 Length of table including the symbol definition table. Back Pointer is the bit difference between word 1 of this table and word 1 of the module header table.

Word 3 Number of symbols in this table.

Word 4 & on Symbols, which can be any of the following:

Variable or array names in ASCII; must be left-justified and blank-filled.

Statement line numbers in ASCII; must be hexadecimal values, right-justified and ASCII zero-filled.

Statement labels in ASCII. Labels which are symbolic names are stored left-justified and blank-filled; labels which are statement numbers are stored right-justified and ASCII zero-filled.

SYMBOL DEFINITION TABLE

The symbol definition table, as it appears in the error processing information area, has the following format, and is an extension to the debug symbol table. It provides further definition to the debug symbols including the type of symbol, address, and mode.

Word				
1	SYMB△DEF 64			
2	Length 16	0 48		
3	Type 16	Location 48		Symbol 1 Definition
4	Mode 16	0 32	Ordinal 16	
	Type 16	Location 48		Symbol 2 Definition
	Mode 16	0 32	Ordinal 16	
	Type 16	Location 48		Symbol N Definition
	Mode 16	0 32	Ordinal 16	

Word 3 Symbol type:

0 Unknown

1 Half-word register variable name

2 Full-word register variable name

3 Variable or array name

4 Line number

5 Label

Location field for symbol type:

1 Half-word address within register file. Since half-word values may be stored in full-word registers, location can range up to hexadecimal 1FF.

2 Full-word register number

3 Bit address relative to the start of the data base

4 Bit address relative to the start of the code base

5 Bit address relative to the start of the code base

Word 4 Mode: Symbol mode, consisting of three parts: precision, description, and data type. In the case of a descriptor, P and Dtype describe the contents of the referenced vector.

P	Desc	Dtype
1	3	12

P	= 0	Precision base is 32 bit, or irrelevant
	= 1	Precision base is 64 bit
Desc	= 0	Not a descriptor
	= 1	Vector descriptor
	= 2	Vector descriptor array
	= 4	Sparse vector descriptor
	= 5	Sparse vector descriptor array
Dtype	= 0	Unknown
	= 1	Logical
	= 2	Integer
	= 3	Real
	= 4	Complex
	= 5	Double Precision
	= 6	Character
	= 7	Bit

Ordinal: The pseudo address vector ordinal of the data base or common block

PSEUDO ADDRESS VECTOR TABLE
(Ordinal Description)

The table, as it appears in the error processing information area, has the following format:

Word		
0	Code Address	64
1	Data Base Address	64
2/3	External Address 1	64
4/5	External Address 2	64
6/7	External Address 3	64
8/9		64
2n+1 2n+2	External Address n	64

For common:

0 16	Address 48
0 16	Bit Length 48

For external symbol, referencing entry point in code:

0 16	Entry Address in code 48
Data Base Length 16	Data Base 48

For external symbol, referencing entry point in data:

0 16	Entry in Data Base 48
Data Base Length 16	Data Base 48

For external symbol, referencing constant entry point:

0 16	Constant Entry Value 48
Data Base Length 16	Data Base 48

DUMP OR A CONTROLLEE FILE

```
January 1, 1901  1:10:09
    10000000  0000 0000 4000 0080   0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0000       @
                  *** ZERO ***
    10002200  0000 0000 0000 00A0   0000 0000 0200 0080   0000 0000 0000 0100   0000 0000 0000 0000
                  *** ZERO ***
    10002C00  0000 0000 0000 0000   0000 0000 4000 0000   0000 0000 0000 0000   0000 0000 0000 0000               @
                  *** ZERO ***
    10003600  0000 0000 6002 0001   0000 0002 2002 0002   0000 0000 0000 0000   0000 0000 0000 0000
                  *** ZERO ***
    10008500  0000 0000 0000 0800   0000 0000 0000 0680   0000 0000 0000 0001   0000 0000 0000 0000
    10008600  0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0000   0006 0000 4004 0000                               @
    10008700  0006 0000 4004 0000   0000 0000 0000 0000   0065 0000 4001 0000   0000 0000 4000 0080       @               @       @
                  *** ZERO ***
    10010000  4D41 494E 2E20 2020   03AC 0000 4003 8800   781E 001D 7818 001C   BE11 0000 4000 E640   MAIN.       @               @  @
    10010100  3610 0011 3E23 0000   7F24 0023 7F25 0023   7F26 0023 7F27 0023   3E28 0048 7F29 0028   6   >#   $ # % # & # ' #>( H ) (
    10010200  3E23 0040 7F2A 0023   3E28 0001 7F2E 0028   9E23 FFFF 0202 0014   3E2C 0013 7F2D 2C23   ># @ * #>(   + ( #      >,   -,#
    10010300  7800 2E2F 3E30 0014   7F2D 302F 8E31 0004   0100 0013 7F2D 0031   3E32 0001 7E2D 3233     ./>0   -0/ 1       - 1>2   -23
    10010400  7800 2E34 3833 0034   7F2D 3234 3E35 0001   7F36 0035 7800 2D37   7837 00FE 0900 0000     .483 4 -24>5   6 5  -7 7
    10010500  00EE 00FE 3E38 0080   6320 3838 3E39 0002   1238 393A 3E38 00FF   3E3B 0001 8004 003A       >8   -88>9   89:>8  >;     :
    10028000  2020 2020 2020 2028   2020 2020 2020 2029   2020 2020 2020 202C   2829 2C00 0000 0000          (       )       ,(),
    10028100  0000 0000 0000 0048   0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0000          H
    10028200  0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0001   5041 5241 4D45 5445                           PARAMETE
    10028300  5220 4F52 2046 4F52   4D41 5420 4552 524F   5220 464F 554E 442E   2020 5554 494C 4954   R OR FORMAT ERROR FOUND.  UTILIT
    10028400  5920 5445 524D 494E   4154 4544 2E00 0000   554E 4142 4C45 2054   4F20 434F 4D50 4C45   Y TERMINATED.   UNABLE TO COMPLE

    10028500  5445 2055 5449 4C49   5459 2E20 5452 5920   4147 4149 4E2E 0000   0000 0000 0000 0000   TE UTILITY. TRY AGAIN.
    10028600  0000 0000 0000 0000   5350 4543 4946 5920   5041 5241 4D45 5445   5253 2D00 0000 0000           SPECIFY PARAMETERS-
    10028700  2020 2028 4649 4C45   312C 4649 4C45 322C   4C3D 4C4E 472C 413D   4144 5231 2C42 3D41      (FILE1,FILE2,L=LNG,A=ADR1,B=A
    10028800  4452 322C 4E3D 4552   524C 494D 4954 2900   5553 4552 2053 5045   4349 4649 4544 204C   DR2,N=ERRLIMIT) USER SPECIFIED L
    10028900  4E47 2049 5320 4C4F   4E47 4552 2054 4841   4E20 4649 4C45 204C   4E47 2E20 5452 554E   NG IS LONGER THAN FILE LNG. TRUN
    10028A00  4341 5445 4420 544F   2030 3030 3030 3030   0000 0000 0000 0040   5448 4953 2046 494C   CATED TO 0000000       @THIS FIL
    10028B00  4520 434F 554C 4420   4E4F 5420 4245 204F   5045 4E45 4420 2D00   2020 2020 2020 2020   E COULD NOT BE OPENED -
    10028C00  2020 2020 2020 2020   2020 2020 2020 2020   2020 2020 2020 2020   434F 4D50 4152 4520                           COMPARE
    10028D00  5445 524D 494E 4154   4544 2E00 0000 0000   4649 4C45 4E41 4D45   2020 2020 2020 2020   TERMINATED.     FILENAME
    10028E00  2020 434F 554C 4420   4E4F 5420 4245 204D   4150 5045 442D 494E   2E20 434F 4D50 4152     COULD NOT BE MAPPED-IN. COMPAR
    10028F00  4520 5445 524D 494E   4154 4544 2E00 0000   2020 2020 2020 2020   2020 2041 4E44 0000   E TERMINATED.              AND
    10029000  2020 2020 2020 2020   2043 4F4D 5041 5245   4420 4551 5541 4C4C   592E 2055 5449 4C49            COMPARED EQUALLY. UTILI
    10029100  5459 2044 4F4E 452E   0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0000   TY DONE.
```

ERROR PROCESSING INFORMATION AREA

```
                  *** ZERO ***
    10048800  204D 4F44 554C 4520   0009 0000 0000 0000   4D41 494E 2E20 2020   594D 4448 4D53 2020    MODULE         MAIN.   YMDHMS
    10048900  0016 2020 204C 4C4C   03AC 0000 0000 1940   0001 0000 4000 0000   0002 0000 4003 3A40        LLL       @    @       @  @
    10048A00  0301 0000 4003 8C80   4558 5420 454E 5452   0009 FFFF FFFF 0440   0001 0000 0000 0003       @   EXT ENTR       @
    10048B00  4D41 494E 2E20 2020   2020 2020 2020 2020   244D 4149 4E2E 2020   0001 0000 0000 0080   MAIN.           $MAIN.
    10048C00  0014 0000 0000 0000   0016 0000 0002 3800   2020 5041 5620 2020   0004 0000 4000 0000                 8   PAV       @
    10048D00  0065 0000 4001 0000   0000 FFFF FFFF FFFF   0000 0000 4001 0000   0000 0000 4001 0000       @               @       @
    10048E00  0000 0000 0002 3800   000C 1F1C 000C 1F1C   000C 1F1C 000C 1F1C   000C 1F1C 000C 1F1C         8
```

INDEX

COMMENT SHEET

CONTROL DATA CORPORATION

TITLE: STAR Operating System Reference Manual

PUBLICATION NO. 60384400 REVISION D

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

CUT ON THIS LINE

FROM NAME: ______________________ POSITION: ______________________

COMPANY
NAME: ______________________

ADDRESS: ______________________

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

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 and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD FOLD

STAPLE STAPLE

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD