60493800

CONTROL DATA CORPORATION

NOS/BE 1 REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 71, 72, 73, 74
6000 SERIES COMPUTER SYSTEMS

CONTROL STATEMENT INDEX

60493800

CONTROL DATA CORPORATION

NOS/BE 1 REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 71, 72, 73, 74
6000 SERIES COMPUTER SYSTEMS

REVISION RECORD	
REVISION	**DESCRIPTION**
A	Manual released.
(11-1-75)	
B	Updated to reflect release of features 145 (844-41/44 Support), 159 and 163 (Job Management
(7-16-76)	and System Control Point Enhancement).
C	Updated to reflect NOS/BE 1.2 at PSR level 447. New features documented include 844 disk drive
(3-15-77)	full/half track recording mode, programmable format control (PFC) for 580 line printers, support
	of CYBER 170 Model 176 with 819 disk drive (device type mnemonic AH), 679 tape unit with
	6250 cpi density capability, and CYBER Control Language (section 5). References to 604 and 607
	tape units are removed. This edition obsoletes all previous editions.
D	Updated to support NOS/BE 1.2 at PSR level 454 and to make editorial and technical corrections.
(8-19-77)	Support of CDC CYBER 170 Model 171 is included.
Publication No. 60493800	

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

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

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

Page	Revision
Front Cover	-
Inside Front Cover	-
Title Page	-
ii	D
iii	D
iv	D
iv-a/iv-b	D
v/vi	D
vii	C
viii	D
ix	C
x	C
1-1	B
1-2	B
1-3	A
1-4	A
1-5	D
1-6	C
1-7	A
1-8	A
1-9	C
1-10	A
1-11	C
1-12	A
1-13	C
2-1	A
2-2	A
2-3	A
2-4	A
2-5	C
2-6	A
2-7	C
2-8	A
2-9	A
2-10	A
2-11	A
2-12	A
2-13	C
2-14	C

Page	Revision
2-15	A
2-16	C
2-17	A
2-18	A
3-1	A
3-2	A
3-3	A
3-4	C
3-5	B
3-6	B
3-7	A
3-8	A
3-9	D
3-10	A
3-11	A
3-12	A
3-13	A
3-14	A
3-15	A
3-16	B
3-17	A
3-18	C
3-19	A
3-20	A
3-21	A
3-22	A
3-23	A
3-24	A
3-25	A
3-26	A
3-27	C
3-28	C
3-29	C
3-30	B
3-31	C
3-32	C
3-33	C
3-34	C
3-35	C

Page	Revision
3-36	A
3-37	A
3-38	A
3-39	A
3-40	C
3-41	C
3-42	C
3-43	C
4-1	D
4-2	D
4-2.1/4-2.2	D
4-3	A
4-4	B
4-5	C
4-6	D
4-7	C
4-8	D
4-9	A
4-10	D
4-11	A
4-12	A
4-13	A
4-14	A
4-15	A
4-16	C
4-17	A
4-18	A
4-19	A
4-20	A
4-21	A
4-22	A
4-23	A
4-24	A
4-25	C
4-26	A
4-27	B
4-28	B
4-29	A

Page	Revision
4-30	A
4-31	A
4-32	A
4-33	D
4-34	A
4-35	A
4-36	A
4-37	A
4-38	A
4-39	B
4-40	B
4-41	B
4-42	B
4-43	C
4-44	A
4-45	C
4-46	A
4-47	A
4-48	C
4-49	A
4-50	A
4-51	C
4-52	C
4-53	C
4-54	D
4-55	C
4-56	C
4-57	D
4-58	C
4-59	C
4-60	C
4-61	C
4-62	A
4-63	A
4-64	D
4-65	D
4-66	A
4-67	A
4-68	C
4-69	C
4-70	C
4-71	C
4-72	D
4-73	C
4-74	A
4-75	C
4-76	C
4-77	B

Page	Revision
4-78	B
4-79	C
4-80	C
4-81	C
4-82	C
4-83	D
4-84	C
4-85	C
4-86	C
4-87	C
4-88	C
4-89	D
4-90	C
4-91	C
4-92	C
4-93	C
5-1	C
5-2	C
5-3	C
5-4	C
5-5	C
5-6	C
5-7	C
5-8	C
5-9	C
5-10	C
5-11	C
5-12	C
5-13	C
5-14	C
5-15	C
5-16	C
5-17	C
5-18	C
5-19	C
5-20	C
5-21	C
5-22	C
5-23	C
5-24	C
5-25	C
5-26	C
5-27	C
5-28	C
5-29	C
5-30	C
5-31	C
5-32	C

Page	Revision
5-33	C
5-34	C
5-35	C
5-36	C
6-1	C
6-2	D
6-3	C
6-4	C
6-5	C
6-6	C
6-7	D
6-8	C
6-9	C
6-10	D
6-11	D
6-12	C
6-13	C
6-14	C
6-15	C
6-16	C
6-17	C
6-18	C
6-19	C
6-20	C
6-21	C
6-22	D
6-23	C
6-24	C
6-25	C
7-1	C
7-2	C
7-3	C
7-4	C
7-5	C
7-6	D
7-6.1/7-6.2	D
7-7	C
7-8	C
7-9	C
7-10	C
7-11	C
7-12	C
7-13	C
7-14	C
7-15	C
7-16	C
7-17	C
7-18	C

Page	Revision
7-19	C
7-20	C
7-21	C
7-22	C
7-23	D
7-24	D
7-25	C
7-26	C
7-27	D
7-28	D
7-29	C
7-30	C
7-31	C
7-32	C
7-33	C
7-34	C
7-35	D
7-36	C
7-37	D
7-38	C
7-39	C
7-40	C
7-41	C
7-42	C
7-43	C
7-44	C
7-45	C
7-46	C
7-47	C
7-48	C
7-49	C
7-50	C
7-51	C
7-52	C
7-53	C
7-54	C
7-55	C
7-56	C
7-57	C
7-58	C
7-59	C
7-60	C
7-61	C
7-62	C
7-63	C
7-64	C
7-65	C
7-66	C

Page	Revision
7-67	C
7-68	C
7-69	C
7-70	C
7-71	C
7-72	C
7-73	C
7-74	C
7-75	C
7-76	C
7-77	C
7-78	C
7-79	C
7-80	C
A-1	A
A-2	A
A-3	A
A-4	A
A-5	A
B-1	C
B-2	C
B-3	C
B-4	C
B-5	C
B-6	C
B-7	C
B-8	B
B-9	C
C-1	C
C-2	C
C-3	C
C-4	C
C-5	C
C-6	A
C-7	C
C-8	C
C-9	C
C-10	C
C-11	C
C-12	C
C-13	C
C-14	C
C-15	C
C-16	C
D-1	C
D-2	C
D-3	C
D-4	C

Page	Revision
D-5	C
D-6	C
D-7	C
E-1	C
E-2	C
Index-1	C
Index-2	C
Index-3	C
Index-4	C
Index-5	C
Index-6	C
Index-7	C
Index-8	C
Index-9	D
Cmt Sheet	D
Inside Back Cover	-
Back Cover	-

PREFACE

This manual describes the NOS/BE Version 1 Operating System for the CONTROL DATA® CYBER 170 Series, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computers. It was written for programmers who use the source languages that operate under NOS/BE 1, and it includes information of specific interest to those who write in COMPASS assembly language.

Extended memory for CDC CYBER 170 Models 171, 172, 173, 174, and 175 is called extended core storage (ECS). Extended memory for CDC CYBER 170 Model 176 is analogous to the CYBER 70 Model 76 large central memory (LCM) or large central memory extended (LCME). When ECS and LCM/LCME are functionally equivalent in this manual, the acronym ECS is used to refer to ECS and LCM/LCME. See appendix E for differences between ECS and LCM/LCME.

Other CDC manuals containing information about NOS/BE 1:

Control Data Publication	Publication Number
NOS/BE 1 Operator's Guide	60493900
NOS/BE 1 Diagnostic Handbook	60494400
NOS/BE 1 Installation Handbook	60494300
NOS/BE System Programmer's Reference Manual	60494100
NOS/BE 1 Station Operator's Guide/Reference Manual	60494200
On-Line Maintenance Software Reference Manual	60453900
UPDATE Reference Manual	60449900
CYBER Loader Reference Manual	60429800
INTERCOM 4 Reference Manual	60494600
INTERCOM 4 MUJ Reference Manual	60494700
CYBER Common Utilities Reference Manual	60495600
CYBER Record Manager Reference Manual	60495700

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 parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

INTRODUCTION 1

The Network Operating System/Batch Environment, NOS/BE 1, is the operating system for the CYBER 170, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems. It is the basic system software that coordinates all other system software, user programs, and hardware action.

The NOS/BE 1 operating system offers a standard set of functions that can be utilized by system programs written in the COMPASS assembly language and by user jobs. It also supports software packages known as the NOS/BE 1 Product Set. The product set includes compilers common to more than one Control Data operating system and products that are unique to the NOS/BE 1 operating system. All products run under the control of the operating system.

NOS/BE 1 is a multiprogramming, multiprocessing operating system. Many jobs can be in the system in various states of processing. It is not necessary for one job to complete before another job begins execution. Among the tasks the operating system performs for a job are: reading the job into the system, assigning it system resources such as central memory and mass storage files, scheduling execution in the central processor, and performing end-of-job procedures that dispose of files used or produced by the job. The operating system also controls the environment of the software and hardware used by a job, such that the resources available to all jobs are used efficiently.

The remainder of this section presents background material about the hardware of the CYBER 170, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems. Product set members that are intimately involved with the operating system, but fully described in other manuals, are also summarized.

HARDWARE FUNCTION AND USE

The CYBER 170, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems have these hardware components:

Mainframe of the computer formed by one or two central processors, central memory, and peripheral processors

Operator console through which the operator oversees software and hardware operation

Peripheral devices including (at minimum) rotating mass storage devices, line printer, card punch, card reader, and magnetic tape units.

Additional hardware that can be part of the system includes:

Extended core storage

Graphics terminals and plotters

Different types of line printers and magnetic tape units

All of the hardware mentioned above usually resides at a central site. However, the CYBER hardware and NOS/BE 1 operating system also can have remote sites connected to the central site through several kinds of communication lines.

More than one central site can be linked together. In particular, a site with 6000 Series Computer Systems can be linked to another 6000 site or to a 7600 site so that users in one location can receive the benefits available through more than one system.

The following discussion introduces the main components of the CYBER 170, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems and shows how they are used during system operation.

MAINFRAME AND CONSOLE

The mainframe consists of central memory, central processor, and peripheral processors operated through a display console.

CENTRAL MEMORY

Central memory consists of 60-bit words. Memory holds instructions to be executed by the central processor, data to be manipulated by the central processor, and data buffered to and from peripheral processors. Any given system can have memory with 49K, 65K, 98K, or 131K words. Memory sizes of 198K or 262K are available with the CYBER 170 Series.

A CYBER 170 has a central memory control that controls the flow of data between central memory and the requesting system components.

Two portions of central memory known as low core and high core are reserved for system use. Low core, the beginning address of central memory, contains central memory resident (CMR) and a small library of system routines frequently used by peripheral processors or the central processor during operating system functions. These library programs exist in memory because they can be loaded from CMR much faster than from the rotating mass storage device on which the rest of the system routines reside, and thereby reduce system overhead. CMR also contains system tables and pointer words, the communication area that links peripheral processors and central memory, and control point areas. High core, the highest numbered addresses in memory, contains information relating to allocation of space on rotating mass storage devices. The amount of memory assigned to low core and high core varies during operation, with space not currently required being released, so that a maximum amount of memory is available for user jobs.

NOS/BE 1 is a multiprogramming system. This means that more than one job can be in central memory at the same time. Although only one of the jobs can be using the central processor in a single-processor system at a given time, all other jobs in memory can have peripheral processors executing tasks for them during that time.

Figure 1-1 shows central memory allocation to the system and user jobs. As shown, the first address is at the extreme low end of central memory and the last address is at the extreme upper end.

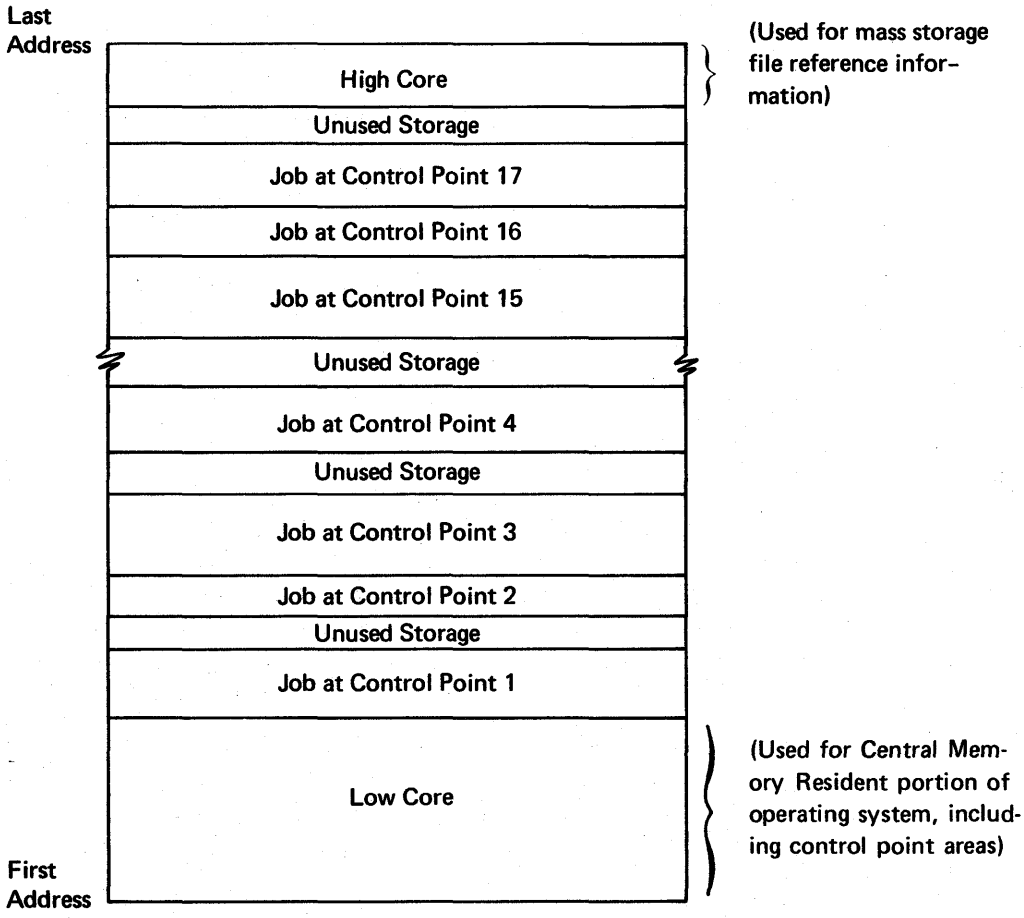


Figure 1-1. Central Memory Allocation

CONTROL POINT DEFINITION

Each job in central memory is assigned a control point number. Control points are the concept by which memory, the central processor and system resources are assigned to a job in memory. Any job in memory has a control point number to identify it and has a 200-word control point area in CMR in which the system stores information about the job. The exchange package for the control point is also stored in the control point area.

The physical portion of central memory allocated to a job is related to the control point number to which the job is assigned. This assignment is made and maintained in numerical order. Thus, the job at control point 2 follows the job at control point 1, and the job at control point 3 follows the job at control point 2, as shown in figure 1-1.

Through a dynamic relocation process, jobs are moved up and down in memory to make room for new jobs assigned to control points. The relocation process occurs continuously as memory requirements change. For example, jobs might be running at all control points except control point 2 when a new job is assigned to control point 2. If sufficient contiguous memory is not available for the new job, other jobs are relocated as necessary to provide sufficient contiguous memory. Each job is moved as a block. It might be necessary to relocate the jobs at both control points 1 and 3, or to relocate only one of them, since unassigned memory can exist between control points.

When a job is moved in storage, MTR suspends all user program activity at the control point, waits for all PPUs assigned to the control point to clear their field access flags, and then starts the system routine that moves the job. When the move is complete, the reference address of the job is modified and job activity resumes. The job itself is not affected by this change in location. Since all program locations are relative to the beginning of the job field length, only the RA address in system tables needs to be changed when the job is moved.

Up to 15 control points, numbered 1 through 17 octal, are available for user jobs. An installation can choose fewer than 15. Control point 0 is used to identify all hardware and software resources not presently allocated to user jobs, or to identify resources known only to the operating system.

At a typical installation, one of the 15 control points is assigned to JANUS, the operating system routine that controls the line printer, card punch, and card reader. JANUS uses central memory buffers, but the actual driving of equipment is performed by peripheral processor, not central processor, programs.

An installation with remote terminals uses INTERCOM to communicate with those terminals. INTERCOM does not use any central processor code to control this communication but executes entirely within the peripheral processors. The central memory required for buffers and control tables is obtained by extending the CMR area. A control point is used only when a task requested from a terminal requires the use of the central processor.

A control point and a job are associated only when the job is in memory or when it has been rolled out. When a job is swapped out, it loses its control point identification.

FIELD LENGTH DEFINITION

Every job in central memory occupies a contiguous block of words. The block is not of fixed size, but rather varies with the needs of the job. The length of the block is the field length (FL) of the job. FL-1 is the relative address of the last word in the block. The first word in the block is known as the reference address (RA); all addresses within each block are relative to RA.

A job can reference locations within its field length, but not outside its field length. Any attempt to read or write outside a job field length is prevented by the hardware, so that all other jobs and system programs in central memory are protected from being accidentally overwritten. For this reason, each job can consider that it is running alone in a computer with a central memory the size of its field length.

The operating system dynamically manages the field length assigned to a job, so that memory is not needlessly tied to a control point when it is not required. Field length increases or decreases as the job progresses. A job step such as a file copy operation, for example, requires much less memory than a step such as a program compilation. The operating system adjusts the field length to the job step needs.

A job normally does not stay in central memory until completion, but moves into and out of memory in relation to its needs for system resources, such as tapes or the central processor itself, and to the needs of other jobs in the system. The scheduler routine of the operating system is responsible for moving jobs into memory to maximize system throughput.

JOB SWAPPING AND ROLLING

When a job with a high priority enters the system, existing jobs of lower priority might be swapped out or rolled out of central memory. The user can specify initial job priority within certain ranges, but the operating system adjusts this priority according to factors such as the system resources requested or allocated and the time consumed in waiting for resources. Some functions requested through remote terminals and those that affect overall system efficiency are assigned high priority. Actions by the central site operator also can affect the priority of any given job.

When a job is swapped out, all information reflecting the current status of the job is written to a mass storage file. The field length and control point associated with the job are made available to the Scheduler. As control points and central memory become available, swapped out jobs are swapped back in to continue processing. A job can be swapped into any free control point; thus, a job might run at several different control points before it reaches termination.

When a job is rolled out, its job field length is written to a rollout file before the field length is freed for another job. The control point is not released when rollout occurs. If extended core storage (ECS) or magnetic tape is being used by a job, that job can be rolled out, but not swapped out.

If a job is waiting for a permanent file to become available or for a mass storage device to be mounted, the job can be swapped out automatically. When the permanent file or device becomes available, the job becomes eligible to be swapped in.

Swapping or rolling might increase the total time that a job spends in the computer, but it has no effect on the amount of central processor time used by a given job; and it should help overall processing. Job swapping and job rollout are controlled by the Scheduler. The most important system effect is to maintain high central processor utilization. Frequent short central processor access is balanced with longer, less urgent, access.

CENTRAL PROCESSOR UNIT

The central processor unit (CPU) is an extremely high-speed arithmetic processor that executes the instructions of system or user programs. It performs computational tasks, but must use central memory for all its input and output including communication with the operating system.

Depending on the specific hardware model, a system might have one of two types of central processors or might have both types of processors in a single system. The differences in the processors has to do with the number of functional units available for concurrent operations, and hence the relative speed at which a given set of instructions can execute.

The CYBER 170 Models 171, 172, and 173, CYBER 70 Models 71-1x, 72-1x, and 73-1x, and the 6200 and 6400 computer systems each have a single processor that has a unified arithmetic unit in which instructions must be executed serially.

The CYBER 170 Model 174, CYBER 70 Models 71-2x, 72-2x, and 73-2x, and the 6500 computer systems each have two central processing units. Both CPUs have unified arithmetic units; thus, two control points can be executing simultaneously on these models.

The CYBER 170 Model 175 and 176, CYBER 70 Model 74-1x, and the 6600 computer systems have a single processor composed of 9 or 10 arithemtic and logical units in which separate instructions from a single program can be executing simultaneously. Careful arrangement of instructions within a program can be done to take advantage of this concurrent execution capability. (See appendix E for a more detailed discussion of CYBER 170 Model 176 differences.)

The CYBER 70 Model 74-2x and the 6700 computer systems have one processor of each type. When only one control point is to use the CPU, it is given the advantages of the 10-unit parallel processor. When a second control point is ready to execute, it obtains the unified processor, thus not disturbing the first job. During normal execution, a program will usually be allotted some time on each of the two CPUs.

The central processor contains three sets of registers: the 60-bit X registers that hold data and instructions, the 18-bit A registers that hold addresses, and the 18-bit B registers used as index registers and temporary storage. The COMPASS assembly language deals with register manipulation.

Only jobs existing in memory are eligible for assignment to the central processor. The job using the central processor might relinquish its control by executing an exchange jump instruction when it must await completion of a task such as a read from a file. The operating system interrupts the job periodically and gives the central processor to another job in memory so that many jobs can be in some state of execution.

When a job loses the central processor, a 16-word exchange package is stored in the control point area for that job. This package contains information used directly in exchange jumps: the most recent contents of all central processor registers, the RA and FL in central memory and in ECS, and the program address which is the address of the next instruction to be executed.

The exchange package is not under user control. The job is made aware of the package when a job terminates abnormally, however. Experienced programmers often can use exchange package information while debugging programs that abort during execution. The package is printed as part of the standard output from an aborted job. It can also be requested by a job.

PERIPHERAL PROCESSOR UNITS

Peripheral processor units (PPUs) are small computers with 4096 12-bit words of memory. Any given system might have seven to 20 peripheral processors. PPUs are independent computers; they all can simultaneously process programs. In addition, a CYBER Model 176 can have up to six first-level peripheral processors (FLPPs) that are used to transfer data to mass storage.

One of the purposes of the PPUs is to perform input and output of data requested by a program executing in the central processor. All data transferred between central memory and any input, output, or storage device passes through a PPU. Peripheral processors also perform the bulk of the tasks required by the operating system, including such tasks as formatting entries in system tables and driving output devices, so that the central processor is available for user jobs.

One peripheral processor holds only the monitor routine, MTR, which oversees and controls all operating system functions. (Part of the monitor also resides in central memory and is known as CPMTR.) Another peripheral processor is devoted exclusively to routine DSD which drives the system display console and input keyboard. This routine interprets and processes all requests typed by the operator and displays all messages from the

operating system routines. Coordination between the central processor and a peripheral processor, or between peripheral processors, is achieved by the MTR routine. Peripheral processor programs are normally the concern only of system analysts.

STATUS AND CONTROL REGISTER

The CYBER 170 computer systems contain a special 204 bit register which is accessible only from a peripheral processor program. The user program is not aware of the existence of this hardware feature in that it does not affect normal execution of programs. The primary purpose of a status and control register is to record the occurrence of abnormal hardware execution or environmental conditions which might affect hardware performance. Parity errors are recorded, as well, to compile a history of operation malfunction valuable to an analyst in determining the origin of a particular problem.

The status and control register also contains bits that can be used to control various optional modes of operation.

OPERATOR CONSOLE

The operator console consists of a keyboard and one or two cathode ray tube display screens. Commands entered through the keyboard are interpreted and processed by the operating system. The displays present a wide variety of information to the operator, ranging from lists of jobs in the systems through hardware status, the control statement any job is currently executing, and the contents of memory for a particular job.

Operator action is required for some jobs, such as mounting requested magnetic tapes. The operating system contains many features that minimize the need for operator commands through the keyboard. Automatic tape assignment, for example, allows the operator to mount a tape and have the system determine which job is using it, rather than having the operator tell the system which job the tape is for. Most jobs can proceed without operator action, but the operator always has the ability to change the automatic functioning of the system.

Normally, a user job does not communicate directly with the operator, although the capability is available through control statements in the job and in some programs.

ROTATING MASS STORAGE

Rotating mass storage is a disk pack used to store operating system files and routines, user jobs, and user files. Permanent files, which are files protected against accidental destruction and unauthorized use, must reside on rotating mass storage.

Rotating mass storage is a random device, as opposed to magnetic tape which is a sequential device. On a random device, information that is logically part of the same file might be physically scattered throughout the storage areas of the device. The operating system is responsible for maintaining the logical order of a file.

No physical distinction exists between binary and coded information on rotating mass storage. Data from an integral number of central memory words is transferred between a buffer in memory and the device with no change. A file declared to be binary when it was written can be read as a coded file, and vice versa. Rotating mass storage is the only device in which this is possible.

Storage space on rotating mass storage devices is assigned to a file as it is required by the file. When a job creates a file, it does not request a particular size of file; no preallocation occurs. Files on mass storage grow as they are written and can overflow to another physical device.

All rotating mass storage devices belong to a logical grouping known as a device set. The installation configures these sets to its own needs.

Public device sets hold system files and user files from any job.

Private device sets hold only files that a job specifically indicates should be on a private device set.

The user job selects the device set on which files are to reside by specifying a specific setname or by default.

UNIT RECORD EQUIPMENT

Unit record equipment is of two categories:

Standard unit record equipment is the line printer, card punch, and card reader necessary for the operation of all systems.

Other unit record equipment can include graphics consoles, plotters, and paper tape readers and punches. These are not a part of the basic system. The operating system defines codes pertaining to files on these devices but does not include the programs needed to operate the equipment. Non-standard unit record equipment runs under control of software provided by an installation.

Standard unit record equipment runs under control of the part of the operating system known as JANUS. All files to be processed by JANUS must be in a special format in which each card or line is terminated by a word with 12-bits of zero in bit positions 0-11.

The card readers can accept, and the card punches produce, files punched with either of two different sets of Hollerith punched codes. Binary punched cards can also be processed in two formats.

Various line printers are available. Models with removable print trains offer character sets with uppercase and lowercase English, fonts with other languages, etc. Fewer unique characters on the train generally increase print speeds. Depending on the code sent to the controller and the controller translation of that code, a character that is producted on one printer can appear as a different character on another printer. For example, a quotation mark output on one printer might well appear as a $\neq$ on another. This often occurs when the character desired is not present on the printer to be used for output.

When an installation has different types of unit record equipment, the job is responsible for providing information in the format required for processing on a particular device.

MAGNETIC TAPE UNITS

The NOS/BE 1 operating system supports both 7-track and 9-track magnetic tape units. When an installation has both types of units available, the job is responsible for specifying the type of hardware unit required to process a given tape. The system default is a 7-track tape. Both binary and coded information can be written.

For a binary tape, bit patterns are written to the tape as they appear in memory.

For coded tape, 6-bit characters in memory are translated to a different 6-bit pattern, known as external BCD, before they are written to the tape.

Density for a 7-track tape can be 200, 556, or 800 bits per inch.

9-track tape corresponds to tapes in industry-standard format. Both binary and coded information can be written, but the information is not the same as 7-track binary or coded information.

For a 9-track binary tape, bits are packed, with three 8-bit characters on tape corresponding to four 6-bit characters in memory.

For 9-track coded tape, bits are either packed or are in 8-bit character codes; the two possible codes are the 64-character ASCII and the 128-character EBCDIC characters.

Density for a 9-track tape can be 800 characters per inch (cpi), 1600 cpi phase-encoded, or 6250 cpi group-encoded.

Another type of control over recording of tape information deals with the number of characters that appear between the physical blocks on the tape and how files and records are recorded. On both 7-track and 9-track tapes, one of three formats must be selected: SI, S, or L; each offers advantages depending on the use made of the tape.

EXTENDED CORE STORAGE

Extended Core Storage is a second, supplementary form of memory that has two main uses. It is used as a mass storage device or as an auxiliary direct access memory. Its large amount of storage and very fast transfer rates make it suitable for many tasks.

CYBER 170 Model 176 systems have a form of extended memory different than other CYBER 170 models but functionally similar. The CYBER 170 Model 176 extended memory cannot be shared with other systems and does not have a distributive data path (DDP) capability. Other minor differences are in appendix E of this manual. References to ECS in the remainder of this document apply to extended memory of all CYBER 170 Models except as limited by the CYBER 170 Model 176 differences described in appendix E.

The use of ECS at any particular site depends on the options selected when the system is installed. Frequently used operating system routines can be placed on the ECS library file, rather than in the central memory low core library area, to reduce the size of low core used by the system without using rotating mass storage. In a multi-mainframe environment, ECS might be used to link the two computer systems.

ECS can be used for buffering sequential files on public devices or for storing sequential or random files (ECS resident files). Each job specifies whether or not a given file will be buffered through ECS or reside on ECS. In this respect, ECS is the same as other mass storage devices except that ECS resident files may not overflow to other mass storage devices.

ECS can be accessed directly from a running program; in this case, a block of ECS is assigned to the user's control point. The block is delimited by RE (reference address for ECS) and FE (field length for ECS) fields in the exchange package. These fields are analogous to the RA and FL fields for central memory. In this mode, ECS is accessed by the ECS direct read/write hardware instructions which perform very high-speed block transfers of user specified length between the ECS and central memory field length addresses specified by the user. The main use of ECS in the direct access capacity is to hold large arrays and tables that do not fit in central memory and would otherwise require partitioning and partial residence on disk; or to otherwise reduce central memory requirements by moving the arrays and tables to ECS as their main residence.

REMOTE TERMINALS

Remote terminals are physically linked to the central site by communication lines. Logically, they are under control of the portion of the operating system known as INTERCOM. INTERCOM allows a user at a remote site to access the central site facilities. INTERCOM is controlled by the central site operator and might not be available to remote terminals all the time the central site is in operation.

Remote terminals are of many different types and complexities. General categories of remote terminals are:

Teletype terminals, which might be a physical Teletype or a display terminal.

Display terminals, which include a keyboard and a display screen, and possibly a character printer.

Remote batch terminals, which have a card reader, line printer, and possibly a card punch attached. Some remote batch terminals have a display screen.

All of the remote terminals provide interactive access to the operating system control statements. That is, control statements can be entered and executed one at a time without being submitted as a complete job. The remote batch terminals allow complete jobs to be entered through the card reader and printed output to be received. Users at remote terminals without a card reader can submit jobs constructed with INTERCOM features or permanent files stored at the central site.

Different terminals operate in different character set modes. Some terminals can be reinitialized to accommodate either ASCII or BCD data; others run only in one mode at all times. Frequently, the line printers of a remote terminal operate in a different mode than those at the central site.

A job can be submitted at one site and specify that its output is to be returned to another site. All job output can be sent to any remote terminal, although it is usually not practical to send lengthy print files to terminals without line printers. Files can be routed between remote sites and the central site in either direction. Each terminal has an identifier assigned when communications are established between the terminal and the central site. This identifier is used to specify the location to receive files.

INDIVIDUAL PRODUCTS

In addition to the capabilities described later in this manual, the operating system includes several features which in turn provide many user options. Several of these features and product set members that are referred to by name in this manual are introduced below.

INTERCOM

INTERCOM interfaces remote terminals with the central site computer. The central site operator must initiate INTERCOM as a program before remote access is possible.

Hardware connected to the remote terminals is controlled through drivers within INTERCOM. Any particular terminal might interface through the L, C, or LCC mode of operation. If the mode required for a terminal has not been initiated by the central site operator, that terminal cannot be used even though other terminals are in operation.

Commands entered at the terminal keyboard call for a variety of INTERCOM capabilities. The first command at many terminals is LOGIN, which establishes the user's authority to use INTERCOM; other terminals do not require LOGIN.

INTERCOM has three distinct capabilities. All three are available from remote batch terminals; only the first two are available from terminals without batch capabilities.

The interactive capabilities of INTERCOM encompass two types of commands. INTERCOM commands allow the terminal user to receive status about files associated with that terminal, display contents of files, and send messages. Any keyboard entry that is not an INTERCOM command is assumed to be an operating system control statement. Consequently, control statements that can be submitted as part of a job, except for magnetic type requests, can be executed one at a time through INTERCOM with a few minor exceptions.

The file creating and editing capabilities of INTERCOM are the primary features of EDITOR. When the terminal user calls EDITOR through a terminal keyboard command, subsequent keyboard entries can become part of a file being created or updated. Interactive commands can also be submitted through EDITOR. When the created or updated file is a source program, EDITOR allows the program to be compiled and executed through a single keyboard entry. EDITOR displays the results on the display screen. When the file is a series of card images corresponding to a job deck, another command causes the file to be entered into the input queue of jobs awaiting execution as though the job had been entered as a card deck through a card reader.

The remote batch capabilities of INTERCOM give the remote terminal user commands for line printer and card reader control. Jobs that originate through the remote batch terminals can be controlled to some extent through the terminal; jobs that originate through interactive commands are beyond terminal user control until the job completes.

CYBER RECORD MANAGER

CYBER Record Manager is the software package that performs execution time input/output for many members of the NOS/BE 1 product set. It is a common product described in full in the CYBER Record Manager reference manual.

The NOS/BE 1 operating system recognizes CYBER Record Manager only as a central processor routine. The operating system itself does not use CYBER Record Manager for any of its functions. Rather, all CYBER Record Manager capabilities are implemented through the standard operating system functions described in the later sections of this manual.

CYBER Record Manager defines five file organizations, eight record types, and four blocking types for sequential files. None of these are known to the operating system in the same terminology or implementation, although operating system actions and CYBER Record Manager functions often result in an identical sequential file.

COBOL programmers access CYBER Record Manager through language statements. FORTRAN Extended programmers can access its capabilities through language statements or calls to CYBER Record Manager routines. COMPASS programmers can use CYBER Record Manager macros instead of the macros described later in this manual. Sort/Merge and FORM users can use CYBER Record Manager through the language in which these utilities are called or through a FILE control statement available to all programs using CYBER Record Manager for execution input/output.

FORM

FORM is a file transformation utility. It is a common product described in full in the FORM reference manual.

FORM can reformat files or records. As a file reformatting utility it has two capabilities:

Reformat files defined to CYBER Record Manager as sequential, indexed sequential, direct, or actual key organization. Files can be transformed into another of these organizations or into the same organization with a different physical structure.

Reformat binary tape files in System/360 format for use under NOS/BE 1.

As a record reformatting utility, FORM has the capability to add or delete characters from each record, blank or zero fill records, convert bit patterns to representations of characters or numbers, and in general change the contents of a specific record. FORM can select all records or only particular records for processing.

FORM is called by a control statement or a COMPASS, COBOL, or FORTRAN Extended statement that specifies the general operations to be performed. Detailed instructions for FORM are submitted as directives that are part of the job deck or are on a separate file for a control card call. Programs pass directives to FORM through common blocks.

UPDATE

UPDATE is a utility program used for modifying files of coded data. It allows a Hollerith punched card or card image to be stored on rotating mass storage, while retaining the ability to modify file contents without recreating the entire card file. UPDATE is a common product described in full in the UPDATE reference manual.

Systems programmers make frequent use of UPDATE when they make local modifications to the operating system or its products. UPDATE is not merely a systems capability, however. Any file of character data can be processed by the utility, whether that file contains a single program being converted from one language version to another, a group of subroutines, or a series of independent statements that a COPY sentence incorporates into a COBOL source program.

A specially formatted file called a program library is created when UPDATE first manipulates a file. This program library should not be confused with a library defined for LOADER purposes. UPDATE files, commonly named OLDPL and NEWPL, are Hollerith card images with history information provided by UPDATE. Files identified as user or system libraries must contain assembled binary programs in a format suitable for loading. UPDATE program libraries must be manipulated only by UPDATE.

UPDATE is called by a control statement that specifies the general operations to be performed. Detailed instructions for UPDATE are submitted as directives that are part of the job deck or on a separate file.

More than 40 directives can be specified, giving the user a wide latitude in modifying the original program library and otherwise manipulating files produced by UPDATE. Among UPDATE capabilities are:

Inserting or deleting cards

Dividing the file into decks for manipulation as a group

Declaring decks common so that a single copy can be used repeatedly without duplication

Temporarily or permanently removing corrections previously made

Producing a new program library incorporating present corrections

Producing a compile file of active cards returned to a format acceptable to assembler or compiler input

COMMON UTILITIES

The common utilities are file listing and updating utilities. Two utilities exist: ITEMIZE and COYPL (see the CYBER Common Utilities Reference Manual).

ITEMIZE is a utility program used for listing information about the content of each record of a binary file. It processes files with system-logical-records and produces printed output. Output specifies the type of record, as determined from the prefix table or other information at the beginning of the file. Depending on the type of record, other information such as entry point names in libraries, overlay level, library table fields, or full text of records can be obtained. ITEMIZE is useful in determining the contents of user libraries, load tapes, and deadstart tapes.

COPYL and its variation COPYLM replace binary records while copying one file to another. COPYL and COPYLM differ only in their handling of multiple occurrences of a record on the file being copied. They operate with binary or text records. These utilities are commonly used to maintain files of procedures or subroutines.

CYBER LOADER

CYBER Loader is the software package that places programs into memory so that they are ready for execution. Loader input is obtained from local files and libraries. Upon completion of loading, execution of the program is initiated if requested.

Loading also involves performance of services such as generation of a load map, presetting of unused core storage to a specified value, and generation of overlays or segments.

JOB PROCESSING AND DECK STRUCTURE 2

A job is a sequence of control statements followed by optional source programs, object programs, data, or directives. A job begins with the job statement and ends with an end-of-information indicator. Jobs exist as physical card decks or images of card decks.

Jobs can enter the system in several ways:

Batch jobs on cards are read in through card readers at the central site. Batch jobs of card images are read from a load tape under the direction of the central site operator.

Remote batch jobs on cards are read in through card readers at remote sites. Remote batch jobs of card images are transmitted from a file created at a remote terminal. All remote batch jobs interface with the central site facilities through INTERCOM.

Interactive jobs are control statements submitted one at a time from a remote terminal keyboard under INTERCOM control. These jobs execute as a series of batch jobs created by INTERCOM in response to individual keyboard entries.

All batch jobs have the same characteristics no matter what their origin. Remote batch jobs differ from central site batch jobs only in that output returns to the terminal and that remote jobs are subject to the limitations of the physical equipment at the remote site. Although all remote sites might not have the capability to produce line printer output, the file that normally would be printed is available on mass storage for display on the terminal. The following information about job decks applies to both decks and deck images.

See the INTERCOM reference manual for specific details of output file handling and specific interface to the operating system, as well as for interactive procedures.

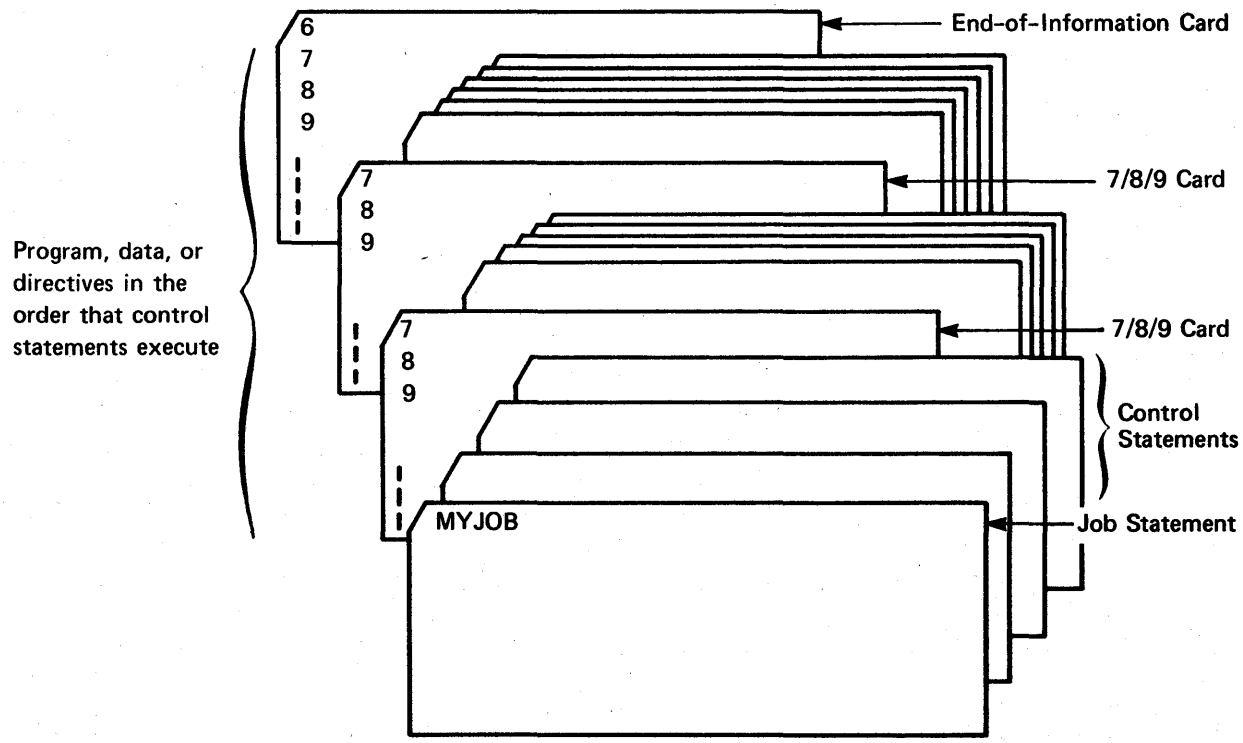
All jobs in the system waiting to begin execution are collectively known as the input queue. Each job enters the system with the name specified by the first five characters on the first card in the job deck. The operating system adds two unique characters to this name to distinguish it from all others in the system.

Once a job enters central memory and begins execution, the image of the job deck is known as a file by the name of INPUT. During job execution, a file with the name OUTPUT is generated by the operating system. When the job completes execution, the file OUTPUT becomes part of the output queue. The output queue is the collective name for output files remaining in the system when the jobs that generated them have completed execution. All print and punch files,and special disposition files such as plot, are part of the output queue. As printers, punches, or remote devices become ready, the operating system causes files from the output queue to be physically output. Files normally return to the user with the name of the job that created them.

Jobs do not read cards directly from the card reader; neither do they directly punch cards or print lines. All job input and job output is stored on mass storage files and on job process images of card or printer files. Physical card reader, card punch, and line printer operations proceed under operating system, not user job, control.

DECK STRUCTURE

The first card of any deck is the job statement; the last card has a 6/7/8/9 multiple-punch in column 1. Cards with a 7/8/9 multiple-punch in column 1 divide the deck into sections.†



Control statements are instructions to the operating system or its loader. They are grouped together at the beginning of a deck. Collectively, the control statements form a job stream. Individually, the control statements are job steps.

Control statements execute in the order in which they appear in the job stream. Consequently, the order of the control statements governs the order of other sections in the deck.

The user is responsible for structuring the job deck such that there is a one-to-one correspondence between each control statement that reads from the file INPUT and the sections of the job deck. The operating system handles each section of the job deck only once, unless the job specifies contrary handling. For example,

†When a job deck is being created as card images through INTERCOM, the *EOR and *EOF entries result in the physical equivalent of 7/8/9 and 6/7/8/9, respectively.

consider two source programs to be compiled and executed with two different sets of data: when one program is compiled and executed before the other is compiled and executed, the control statements and deck structure must be:

```
DECKA.
COBOL.                                  Compile first source program and write binary file LGO
LGO.                                    Execute binary file
REWIND,LGO.
COBOL.                                  Compile second source program and write binary file LGO
LGO.                                    Execute binary file
7/8/9
      first source program
7/8/9
      data for first source program execution
7/8/9
      second source program
7/8/9
      data for second source program execution
6/7/8/9
```

On the other hand, if both programs were compiled before either was executed, the corresponding deck structure would be:

```
DECKB.
COBOL.                                  Compile first source program and write binary file LGO
COBOL,B=ABC.                            Compile second source program and write binary file ABC
LGO.                                    Execute binary file LGO
ABC.                                    Execute binary file ABC
7/8/9
      first source program
7/8/9
      second source program
7/8/9
      data for first source program execution
7/8/9
      data for second source program execution
```

The two decks above illustrate the principles of all deck structuring.

SEPARATOR CARDS

One job is separated from another job by a card with a 6/7/8/9 multiple-punch in column 1. This card is known as an end-of-information card.

Within a single job deck, each section is separated by a card with a 7/8/9 multiple-punch in column 1. Once on mass storage, these cards are represented by system-logical-record terminators of level 0, as discussed with rotating mass storage files in section 3. A compiler or assembler encountering a 7/8/9 card image during processing treats the card as a partition or file end.

An octal level number 0 through 17 can be punched in columns 2 and 3 of a separator card. A level number of only one digit can be punched in column 2. When columns 2 and 3 are blank, a level number of 0 is assumed. Level numbers are not normally used on separator cards.

Interpretation of a 7/8/9 level 17 card depends on whether the ST parameter on the job statement indicates the job might be run on a system under control of the SCOPE 2 operating system. JANUS, the system routine that controls standard unit record equipment, converts a 7/8/9 level 17 card to the equivalent of a 6/7/8/9 end-of-information card when the job cannot execute under SCOPE 2. No such equivalencing occurs for job decks that might execute under control of both NOS/BE 1 and SCOPE 2. A 7/8/9 level 17 card should not be used in place of a 6/7/8/9 card when the job might execute under the SCOPE 2 operating system.

Separator cards can be used to indicate whether the cards following them are punched in 026 or 029 character codes, as discussed in appendix A.

CONTROL STATEMENT SECTION

The first section of a job deck contains only control statements. Each control statement results in the execution of a program in the central processor or in a peripheral processor. Many control statements call programs that make entries in system tables; others call programs that perform utility functions such as file copy. Several broad categories of control statements are:

Operating system functions such as assigning a tape unit to the job or routing a print file to a remote terminal. These functions are fully described in section 4 of this manual.

Utility functions such as file copy or creation of user libraries. These functions are also described in section 4 of this manual.

Loader functions such as load, but not execution of a program, and satisfying program references from different libraries. Only the simplest LOAD and EXECUTE statements are summarized in this manual; the LOADER reference manual has complete details of all loader functions.

Program call functions which are a request to the operating system to load and execute information existing on a file attached to the job. This function is discussed below.

Each of the control statements discussed in this manual is available to the job because the control statement name is the entry point to a program on a system library named NUCLEUS.

LIBRARY USE

A library is a collection of programs in executable form accompanied by library tables that specify the content of the library. The operating system uses the libraries as the source of programs with entry point names specified on control statements.

Two types of libraries exist: system libraries and user libraries.

A system library is available automatically to all jobs. It is named in the Library Name Table in central memory resident (CMR). It is contained on a permanent file that can be read by more than one job at a time, and parts of it can be contained in CMR.

A user library is a file formatted as a library, but it is not available to a job until it has been explicitly brought to the job. The job might create the file before using it as a library, or it might be a permanent file that a job would attach explicitly. A permanent file might be such that more than one job could read it at once; but every job must explicitly declare the job. The EDITLIB utility can be used to create a user library.

The particular libraries that are used for each job, or for each loading operation within a job, depend on the library set defined by the job. The total library set consists of the global library set, the local library set, and the system library NUCLEUS.

NUCLEUS is a system library that cannot be removed from the library set. It contains the items listed under the heading System Texts in section 5.

The local library set is defined by the loader control statement LDSET(LIB= . . .). Local library sets are valid only for the current load operation. At the start of each load operation, the local library set is defined as empty unless the LIB parameter of LDSET is specified (see the CYBER Loader Reference Manual).

The global library set is defined by the loader control statement LIBRARY. Global library sets are valid throughout the job or until another LIBRARY control statement changes the global library. At the start of each job, the global library set is defined as empty.

The loader uses the library set in the following order:

Local libraries

Global libraries

NUCLEUS.

Any program name on a control statement is loaded first if a file with that name is attached to the job. Then the library set is searched and a program loaded for any matching entry point. In a simple job, the local library set and global library set are both empty, so that the NUCLEUS library is the source of control statements executed. Given the library set search order, however, any user program with the same name as a system program is executed when the proper library set is declared in the job.

See the LOADER reference manual for further details of library use during loading.

LOAD SEQUENCE

A load sequence is a consecutive series of control statements that begins with a call that causes a program to be loaded into central memory. A load sequence ends with a call that initiates execution. The following is a load sequence with three control statements:

```
LOAD(ABC)
LOAD(DEF)
EXECUTE.
```

All control statements in a load sequence must contain only instructions for the loader. Both LOAD and EXECUTE are loader statements. The other control statements that appear in this manual are not loader statements, unless they are specifically identified as such.

Any control statement that calls for execution terminates a load sequence. Any name call such as LGO, ABC, REQUEST(. . .), terminates a load sequence. In most instances, a control statement initiates and terminates a single statement load sequence.

Other statements that are part of a load sequence, or that affect the loading of programs are:

LOAD	Loads modules from file specified.
LIBLOAD	Loads modules specified by entry point names from the library named.
SLOAD	Loads specified modules from the file named.
EXECUTE	Completes load and executes.
NOGO	Completes load and produces a core image on specified or default file.
SATISFY	Specifies name of a library to be searched for unsatisfied externals.
LDSET	Specifies a list of independent options that can preset central memory field length, alter default rewind options, control load map generation, define the libraries in the local library set, select loading error handling, and force loading or inhibit loading of routines.

See the LOADER reference manual for a full description of these control statements.

LGO AND PROGRAM EXECUTION CALLS

All assembler and compiler calls allow the user to specify the name of the file to contain executable code. In the absence of another name, a file with the logical file name LGO is created. A job does not necessarily have a file with the name LGO.

When LGO is encountered in the job stream, the operating system searches for a file with that name. In the default instance, such a file exists and it is loaded and executed. LGO contains the relocatable object code produced by the compilers in the absence of a source program statement that directs absolute code. (See the LOADER reference manual for absolute code information.)

Similarly, any file name presented among the control statements is assumed to contain a program that can be loaded and executed. For example,

```
FTN,B=OLIVER.      Writes object code on file OLIVER
OLIVER.            Calls for load and execution of OLIVER
```

Parameters can appear on the program call, depending on the object program itself. The FORTRAN Extended compiler, for instance, produces object code that can process file names. The following program call substitutes files TAPE2 and TAPE3 for whatever file names are compiled into the object code:

```
OLIVER,TAPE2,TAPE3.
```

The COBOL compiler, on the other hand, does not produce object code that can accept parameters on the program call. The reference manuals for the individual products describe any such capability.

Any user program that can access the first 100 octal locations of the job field length can be written to accept program call parameters. Positioning of the file named on a program call is controlled by installation default. At most installations, rewind occurs automatically before loading. In a straightforward compile-and-execute job, the file LGO or its equivalent need not be rewound.

When more than one program is written on LGO, however, manipulation of LGO might be required. If the first program is a main program and the second is a subroutine called by the main program, a single call for LGO rewinds the file, loads both programs, and executes.

If the two programs are independent, however, execution stops at the end of the first object program. A second call to LGO rewinds the file, such that the first program executes a second time, rather than having the second program execute. The previous example job DECKA shows a deck structure with one file name that executes two independent programs with a control statement to rewind this file so that the second program overwrites the first. An alternative is example DECKB in which the second independent program is written to a separate file and executed by a call with the name of the file ABC.

COMPILER AND ASSEMBLER CALLS

Names that should be used on the program execution call statement to assemble or compile a user program are listed below:

Source Language	lfn	Source Language	lfn
FORTRAN Extended	FTN.	SYMPL	SYMPL.
COBOL Version 4	COBOL.	Sort/Merge	SORTMRG.
COBOL Version 5	COBOL5.	PERT/TIME	PERT66.
ALGOL	ALGOL.	APT	APT.
ALGOL Editor	ALGEDIT.	QUERY UPDATE Version 2	Q2.
COMPASS	COMPASS.	QUERY UPDATE Version 3	QU.
SIMSCRIPT	SIMS.	FORM	FORM.
BASIC	BASIC.	Data Definition Language	DDL.

Parameters on the control statements are used for such functions as:

Naming the file containing the program to be assembled or compiled (default name INPUT)

Naming the file to which the program is to be translated in object code (default name LGO)

Producing source language or object code listings of the program (listing options such as S in FORTRAN)

Parameters for many products are the default I=INPUT, B=LGO, and L=OUTPUT. See the reference manual for a particular compiler for a full description of parameters that can appear on the control statement. When a compiler or assembler call specifies INPUT as the name of the file containing the source program, the next unexecuted section of the job deck must contain the program.

EFFICIENT CONTROL STATEMENT ORDERING

Placement of some control statements, particularly those that cause hardware devices to be assigned to a job, can affect the efficiency with which all jobs execute. Parameters on those statements can also affect job throughput.

A REQUEST control statement for a magnetic tape assigns a tape drive unit to the job as soon as the tape is made ready and the operating system is aware of the tape location. The tape unit remains assigned to the job either until the job executes a control statement that releases the unit or the job terminates.

The examples below presume a job compiles a FORTRAN Extended program and executes the program twice using different sets of data on individual tape volumes.

An inefficient ordering of control statements is:

```
INEFFICIENT,MT2                    Job statement indicates 2 tape units required
REQUEST,DATA,MT.  ASSIGN 3456.
REQUEST, DATA2,MT.  ASSIGN 3457.
FTN.
LGO.
LGO.
```

The same operations performed more efficiently are:

```
EFFICIENT,MT1.
FTN.
REQUEST,DATA,MT,VSN=3456,NORING.
LGO.
UNLOAD,DATA.
REQUEST,DATA2,MT,VSN=3457,NORING.
LGO.
RETURN,DATA2.
```

The second job is more efficient in several ways:

Only the number of tapes required at one time is indicated on the job statement, not the total required in all. Jobs with tape requirements are captured in a tape queue when they enter the system; they are not released to the input queue, and consequently cannot begin execution, until certain tape availability requirements are met.

A tape is requested when it is required, not before. Since the compiler does not use the data tape, the tape is not requested until after compilation is complete.

The VSN parameter on the REQUEST control statement permits the operating system to assign the mounted tape to the job without operator command. Without VSN information, the operator must inform the operating system of the location of the tape.

The tape unit is returned to the system when it is no longer needed, instead of having the job hold the unit until job termination.

In general, control statement placement can affect job execution time whenever a magnetic tape or private device set is used.

DIRECTIVE SECTION

Directives are defined as control information that does not appear within the control statement section of a job deck. They are required by several of the utilities, including EDITLIB and COPYN, and by several common products such as UPDATE and FORM.

When directives specify instructions which will not fit on a single control statement, the programmer has the option of:

Placing directives on a file and making the file available to the job before the directives are needed; or

Placing the directives within the job deck.

The name of the file containing the directives must be specified in the call to the utility or product. The default file name for most calls in INPUT.

When directives are part of a job deck, they must appear in a separate section. The deck must be structured such that the directives are the next unprocessed section of the deck at the time the utility or product executes.

DETAILED JOB FLOW THROUGH SYSTEM

The following information describes the system procedures that occur as a job passes through the system. An understanding of this information is not required for system use.

From the time a job is assigned to a control point and execution is completed, many other jobs are being executed. Each job is assigned a job descriptor table (JDT) ordinal when it is first assigned to a control point. If the scheduler routine swaps out the job (returns it to mass storage in its present state of execution), the JDT ordinal maintains the identity of the job when the control point association is lost. A job can be swapped out by the scheduler when a job with higher priority enters the system or when the job is delayed waiting for a resource such as a disk pack. A job can be rolled out also, freeing central memory but retaining a control point, while awaiting operator action. The scheduler directs swapping and rolling, taking into consideration the relative needs of batch jobs and interactive jobs. When jobs are swapped or rolled into central memory, they resume execution at the point of interruption.

EXAMPLE JOB

The manner in which control statements establish user program handling is illustrated by following a sample job as it is processed. For example, consider a job to assemble and execute a program written in COMPASS, with the output to a line printer. The user gives the operator a tape to be used for output. In the sample job shown below, the tape has a label containing 1972 as the volume serial number.

The job would be structured as follows:

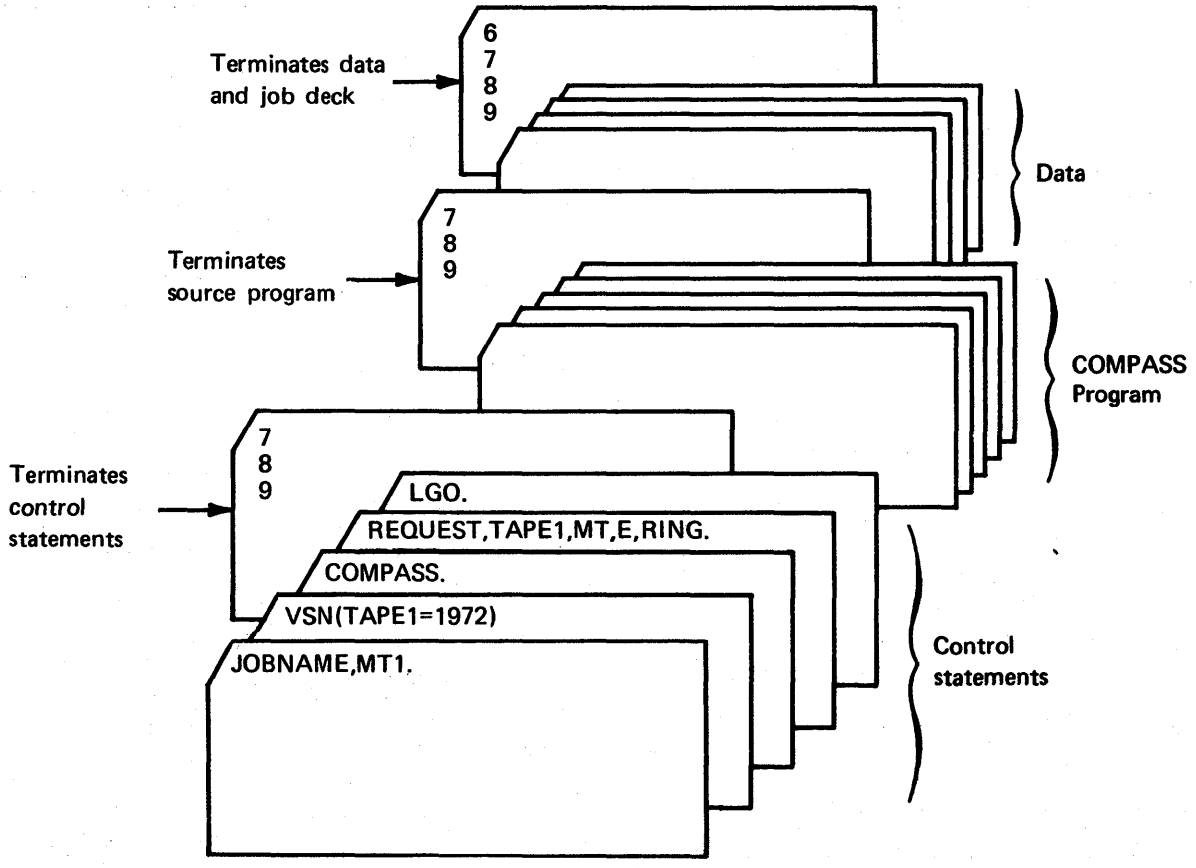


Figure 2-1. Sample COMPASS Job

When the sample job is input through the card reader, the operating system calls a PP routine to translate the job statement, check the validity of its entries, and assign a priority to the job. Next the PP copies the job through a central memory input/output buffer onto mass storage. At this point, the operating system identifies the job by its file name JOBNA01 (from the job statement).

EXAMPLES OF JOB DECK ARRANGEMENTS

The order in which control statements are arranged depends upon the purpose of the job and the programs it contains. The following examples illustrate typical arrangements. Automatic rewind before a load is assumed.

1.
```
JOBA,MT1.
REQUEST,SALLY,MT,VSN=123456.
SALLY.
6/7/8/9
```

JOBA requests a tape file named SALLY, and loads and executes an object program from that file.

2.
```
JOBB.
FTN.
LGO.
7/8/9
FORTRAN Extended Program
6/7/8/9
```

JOBB, containing a FORTRAN Extended program on Hollerith cards, compiles, loads and executes that program.

3.
```
JOBC,T50.
INPUT.
7/8/9
Program on Binary Cards
6/7/8/9
```

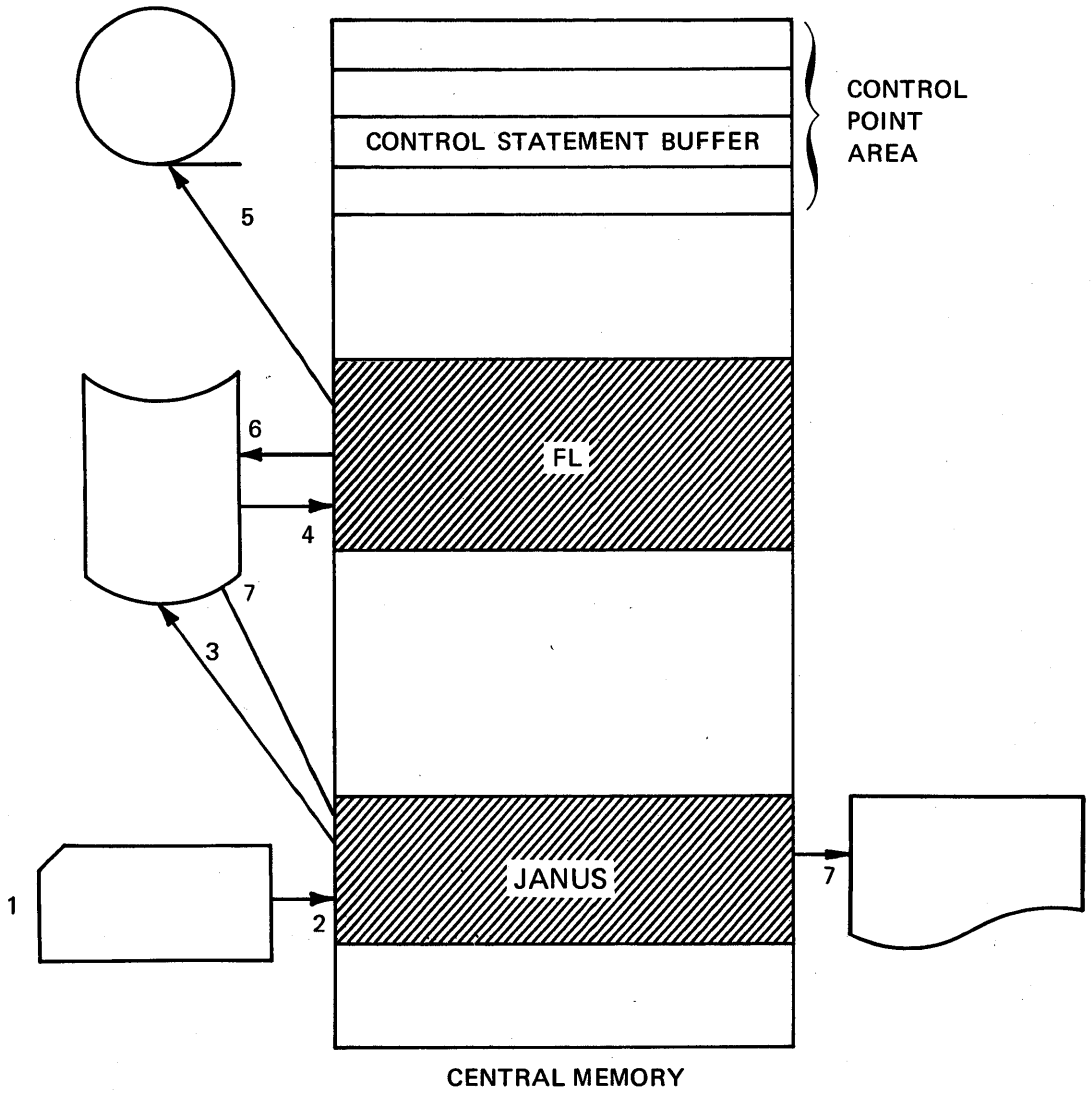
JOBC, containing a program on binary cards, loads and executes that program.

4.
```
JOBD.
FTN.
LGO.
LGO.
7/8/9
FORTRAN Extended Program
7/8/9
First Data record
7/8/9
Second Data record
6/7/8/9
```

JOBD compiles and executes a FORTRAN Extended program and executes this program with one set of data, and then with another.

5.
```
JOBE.
ATTACH, MYLIB, ID=MINE.
COBOL.
REWIND, LGO.
EDITLIB, USER.
7/8/9
COBOL program
7/8/9
LIBRARY(MYLIB, OLD)
ADD(NEWPROG, LGO, AL=1)
FINISH.
6/7/8/9
```

JOBE compiles a program and adds it to a user library named MYLIB. Directives required by the EDITLIB utility during library manipulation are the last section of the deck.



1	Job read into card reader	5	Some output to a tape
2	Job read through buffer onto disk	6	Job assigned to output queue
3	Job in mass storage input queue	7	Output to printer through buffer to printer
4	Job assigned control point; goes into execution		

Figure 2-2. Job Flow at Central Site

When the job is in the input queue of jobs awaiting execution, it comes under control of a NOS/BE 1 scheduling routine. The following factors are considered in assigning jobs to available control points: the priority entered with the job, available system resources such as central memory, direct access ECS, tape units, and the total time the job has been in the system. A job descriptor table ordinal is assigned to the job; this ordinal is used to identify the job while it is in execution regardless of whether it is in central memory or not.

The job then waits for the scheduler to assign it to a control point. When a control point becomes available, the scheduler assigns the job and initializes the control point with pertinent information about the job. NOS/BE 1 saves the assigned job name for later use.

The job file name is changed to INPUT and the file is positioned at the statement following the first 7/8/9 card (the beginning of the user's program). The first control statements are read into a buffer within the related control point area in low core, and are ready for execution. As job output is created, it is written to a file named OUTPUT.

Accounting processing, if selected by the installation, occurs as the first step of actual job execution. Accounting information extracted from the job statement or the statement following it is validated and saved for later use by the system. The accounting information defined by the system can include such items as name, account number, project number, etc. If accounting is not selected by the installation, as in this example, accounting information need not be present.

After accounting processing, the system copies the BATCH system bulletin to the job's OUTPUT file. If the installation has not specified BATCH system bulletin information, no information is written to the OUTPUT file. The installation can specify other standard procedures to be executed at this time.

Upon completion of all standard procedures, job control is advanced to the second statement, COMPASS, which directs assembly of the user's program. NOS/BE 1 requests the loader to load the COMPASS assembler into the field length. Control passes to COMPASS to assemble the next cards on the file INPUT and put the object program on the file LGO. The assembler stops when it reads a 7/8/9 card. (For assembly or compilation, the user can designate files other than INPUT as an input file and other than LGO as binary output by entries on the COMPASS control statement; but unless such alternative files are named on the assembly or compilation card – the COMPASS statement in this case – INPUT and LGO are used by default.) COMPASS also writes a source language listing of the program onto a file named OUTPUT. At job termination OUTPUT is printed unless the user specifies otherwise.

Control is then advanced to the next REQUEST statement. The VSN parameter provides the volume serial number for the tape label. NOS/BE 1 automatically assigns the tape if it is mounted. (If the installation does not choose the automatic assignment feature of NOS/BE 1, the REQUEST statement appears on the operator console; and the operator must assign the tape to the job manually.) Control proceeds to the next control statement, LGO.

The LGO statement directs program execution. The loader loads the LGO file containing the user's program in object code into central memory and writes a map of this program onto the file OUTPUT; library subprograms required are loaded also. Control passes to the user's program for execution, input data is read from the next element of the INPUT file (user's data), and output is written on TAPE1 and OUTPUT.

As each control statement is executed, it is copied onto the job and system dayfiles. Control statement processing stops when the first 7/8/9 card is encountered. NOS/BE 1 writes job accounting information and job statistics on the dayfile and copies this file to OUTPUT, which then is detached from the control point. The name OUTPUT is changed to JOBNA01 (the assigned job name) and TAPE1 is released so that the tape unit can be available for another job. INPUT and LGO are cleared and released from NOS/BE 1 control. All equipment associated with the job is released from control point n and assigned to control point 0, where it can be requested by other jobs. The control point area and field length in central memory are made available for other jobs. When a printer is available, JOBNA01, containing the assembly language program listing, load map, output, and dayfile, is printed.

JOB TERMINATION DETAILS

When a job is processed without error, normal termination activity begins upon reaching the end of the control statements or some form of EXIT control statement. First, execution time of the job is written onto the job dayfile and on the system dayfile. Then, the job dayfile is rewound and copied onto the file OUTPUT. Next, OUTPUT and any other files on mass storage designated for output, such as PUNCH or PUNCHB, are rewound and placed in the output queue. OUTPUT is designated for the printer, and PUNCH (Hollerith) and PUNCHB (binary) for the card punch by disposition codes. These files names are then changed to the job name and assigned to control point 0.

The files listed below are treated as special cases. Unless the user overrides the default disposition of such files, they are designated for output at job termination and automatically assigned a specific disposition code.

OUTPUT	PUNCH	FILMPR	HARDPR	PLOT
	PUNCHB	FILMPL	HARDPL	P80C

Files on magnetic tape are rewound (unloaded if the programmer requested save status), and released from the system. Permanent files are released from the job and returned to permanent file manager jurisdiction, and private device sets are dismounted. All remaining files in central memory and mass storage associated with the job including INPUT, LGO, and the job dayfile, are cleared and released. The job is released from the control point area.

All hardware devices assigned to a job are assigned to control point 0, so they can be reassigned to other jobs. At this point, only files in the output queue relating to the job remain. When an output device of the type requested by the file's disposition code is free, the file is output through that device.

ABNORMAL TERMINATION

When a fatal error occurs, the operating system sets a flag indicating the error. If the error has been previously identified in the current job step by a call to RECOVR, control is returned to the user program for processing. Otherwise error processing continues.

A diagnostic message that reflects the reason for abnormal termination is written to the job dayfile.† A standard abnormal termination dump then occurs. The dump appears on the file OUTPUT with the heading DMPX; this dump shows the contents of the exchange package for the job, the contents of central processor registers, and the contents of words before and after the location at which the program stopped. See the DMP control statement for a description of the dump output.

†When a file is designated for output (output, punch, and so forth), the system finishes the write operation in progress at the time of termination.

The operating system then clears the error flag and searches the control statements for an EXIT statement. Depending on the parameter of EXIT and the type of error that occurred, processing might resume with the first control statement after the EXIT statement. See the EXIT control statement for a description of the different error conditions and EXIT parameters. If no EXIT statement exists, the job terminates as described above for normal job termination.

OPERATOR COMMAND TERMINATION

When the operator types in a DROP command, the job terminates prematurely. End-of-job procedures are initiated as described under abnormal termination.

When the operator types in a KILL command, the job terminates prematurely. All files associated with the job, including the OUTPUT file, are dropped regardless of name or disposition. Permanent files are treated the same as for normal termination. The programmer does not receive a dayfile listing.

When the operator enters a RERUN command, the job is terminated and its INPUT file is returned to the input queue, so that it can be run later. The OUTPUT file is dropped, and a new output file is created. The job dayfile is copied to the new output file called a pre-output file, and becomes the OUTPUT file when the job is run again. The OUTPUT file for the rerun job will contain the dayfile from the previous partial run of the job and the output and dayfile from the complete run of the job.

Permanent files and mounted private device sets for a rerun job are treated as for normal termination. All other files, regardless of name or disposition, are dropped.

In some cases, a job might perform a function which would make it impossible to restore conditions to their initial state before the job was run. For example, if a job writes on an existing permanent file, that information cannot be erased. When such a job is rerun, results are unpredictable. To avoid this condition, the system will set a no-rerun flag in the control point area to reject a RERUN type-in by the operator. The no-rerun flag will be set when the job has performed a catalog, purge, alter, rewrite, rename, or extend of a permanent file; modified a permanent file; added or deleted a member of a device set.

Should a job be caught at a control point during a deadstart recovery, it is either dropped or rerun depending upon the no-rerun flag. If possible, the job is rerun; however, if the flag indicates no rerun, the job will be dropped and an appropriate message added to its dayfile. Any job swapped out during a deadstart recovery will be given a message indicating that recovery was performed.

JOB DAYFILE

The last item of the file OUTPUT from any job is the job dayfile. It gives a history of job execution. Any program or job that terminates abnormally produces dayfile messages identifying a fatal error. Normal job completion is indicated by the absence of fatal error messages.

Each control statement that is called to execution is listed in the dayfile. System response to a control statement might follow. The dayfile shows, for example, the VSN of a scratch tape assigned; such information might be needed as input in another job using that tape. The NOS/BE 1 Diagnostic Handbook gives the meaning of status and error messages originating in the operating system. Messages that originate from a member of the product set are explained in the individual product reference manual.

The programmer can cause information to be sent to the job dayfile by using the COMMENT control statement or the MESSAGE macro in a COMPASS program. Several other language processors also allow messages to be sent to the operator or to the dayfile.

Figure 2-3 shows a typical dayfile.

```
  MFS        NOS/BE 1          sys level    mm/dd/yy
16.42.19.BASIC60  FROM
16.42.20.IP  00000192 WORDS - FILE INPUT  , DC 00
16.42.20.BASIC31,T40,P2,MT1.
16.42.26.REQUEST(COMPILE,*Q)
16.42.27.REQUEST(OLDPL,E,HY,VSN=4174,NORING)
16.43.50.( MT30 ASSIGNED)
16.44.36.UPDATE(Q,D,8,*==)
16.44.38.MT30 VOLUME SERIAL NUMBER IS  004174
16.45.58. UPDATE COMPLETE.
16.45.59.ROUTE(COMPILE,DC=IN)
16.45.59.UNLOAD(OLDPL)
16.46.06.OP  00001920 WORDS - FILE OUTPUT , DC 40
16.46.07.MS    3584  WORDS  (    3584 MAX USED)
16.46.07.CPA      2.171 SEC.        2.171 ADJ.
16.46.07.CPB      1.164 SEC.        1.164 ADJ.
16.46.07.IO      14.143 SEC.       14.143 ADJ.
16.46.07.CM     285.807 KWS.       17.444 ADJ.
16.46.07.SS                        34.923
16.46.07.PP      34.835 SEC.     DATE mm/dd/yy
16.46.07.EJ  END OF JOB, **
```

Figure 2-3. Sample Dayfile

The system header identifies the system on which the job executed. Installations might change the information shown. In the example above:

MFS	Mainframe identifier
NOS/BE 1	Operating system level
10/15/75	Date the operating system was built; time and type of deadstart recovery appears if recovery has occurred.

The first line after the system header gives the name of the job as modified by the operating system to make the name unique among all jobs, and the job's origin in the following format:

jjjjjjj from sss/tt

jjjjjjj	Jobname
sss	Source Mainframe ID
tt	Terminal ID

The lines giving statistics about the input and output files have the following format:

```
IP nnnnnnnn WORDS – FILE xxxxxxx, DC yy
or
OP nnnnnnnn WORDS – FILE xxxxxxx, DC yy
```

IP	Indicates that this message refers to an input file
OP	Indicates that this message refers to an output file
nnnnnnnn	Decimal number of words in the file
xxxxxxx	Logical file name
yy	Disposition code of an output file. DC 40 is for print on any printer. See the DISPOSE macro for a list of disposition codes.

Accounting messages are added to the dayfile at the end of the job and each time a SUMMARY control statement executes. They have the following format:

```
MS  aaaaaaaa WORDS (bbbbbbbb MAX WORDS USED)
CPAcccccccc.ccc SEC.  ddddddddd.ddd ADJ.
CPBcccccccc.ccc SEC.  ddddddddd.ddd ADJ.
IOeeeeeeeee.eee SEC.  fffffffff.fff ADJ.
CMggggggggg.ggg KWS.  hhhhhhhhh.hhh ADJ.
ECiiiiiiiii.iii KWS.  jjjjjjjjj.jjj ADJ.
SS                    kkkkkkkkk.kkk ADJ.
PPmmmmmmmm.mmm SEC.       DATE MM/DD/YY
```

All values are in decimal, with leading zeros omitted:

aaaaaaaa	Mass storage currently used by the job, not including the INPUT file nor any permanent files the job attaches. Newly created permanent files are included in the word count. This message is issued only if the job has executed a LIMIT control statement or if the installation has established a mass storage limit. The decimal value in words is computed by multiplying the number of record blocks used by the number of words in a record block.
bbbbbbbb	Maximum mass storage used by the job. Otherwise, the same as aaaaaaaa.
cccccccc.ccc	Central processor time; dual processors are reported separately.
ddddddddd.ddd	Adjusted central processor time for each processor. The time is multiplied by an installation selected weighting constant.
eeeeeeee.eee	Input/output time.
fffffffff.fff	Adjusted input/output time. The time is multiplied by an installation selected weighting constant.

gggggggg.ggg	Central memory kilo-word seconds. This value indicates central processor usage, and is a sum of terms, each term computed as follows. Central processor time and IO time are weighted, to compensate for overlapped IO processing, and then added together. This sum is multiplied by central memory field length divided by 1000 octal. Each time central memory field length changes, a new term is computed. Thus, the number of terms summed is the same as the number of times central memory field length changes during job execution.
hhhhhhhh.hhh	Adjusted central memory kilo-word seconds. Statistic is the same as control memory kilo-word seconds with weighting factors selected by the installation.
iiiiiiii.iii	Extended core storage kilo-word seconds. This value is computed in the same way central memory kilo-word seconds are computed, except ECS field length divided by 1000 octal is used.
jjjjjjjj.jjj	ECS kilo-word seconds adjusted by installation selected weighting factors.
kkkkkkkk.kkk	System seconds. The sum of the adjusted values of central processor time, IO time, central memory kilo-word seconds, and ECS kilo-word seconds.
mmmmmmmm.mmm	Peripheral processor time.

FILE CONCEPTS AND STRUCTURE 3

A file is defined as a set of information that begins at beginning-of-information, ends at end-of-information, and has a logical file name.

This section summarizes job responsibilities for files and the devices on which they reside and introduces the control statements used to process different types of files. Structure of files within the system is also defined.

GENERAL FILE USAGE

A job is responsible for:

Specifying the logical file name by which a file is known during the job;

Assigning the file to a particular device, if necessary;

Disposing of the file if it is to be preserved when the job ends.

NAMING FILES

Each file associated with a job is known by its logical file name. The operating system associates two files with each job; one with the logical file name INPUT and another with the logical file name OUTPUT. All other logical file names must be specified by the job. The logical file name is valid only for the duration of the job. The name is not part of the file itself; it is not written in the label of a file on tape, and it is not a part of the permanent file table information.

Each logical file name must be unique within a job and must not duplicate the name of a multi-file tape set associated with the job. Logical file names are one through seven letters or digits and must begin with a letter.

RESERVED LOGICAL FILE NAMES

Logical file names that begin with ZZZZZ are reserved for use by the system. User jobs are not prevented from creating or reading files with the name ZZZZZxx, but use of these files might adversely affect the job.

SPECIAL-NAMED FILES

Special-named files are those with an inherent set of characteristics and disposition. The operating system assumes the following characteristics for those files named below:

INPUT

INPUT is the name of the file with the images of the job deck. Each separator card in the deck, or its logical equivalent, is an end-of-partition when processed by system routines in the operating system or the standard compilers. The separator cards trigger end-of-file processing. Each card image is a separate record to compiler and assembler programs.

OUTPUT

Every job has a file of the name OUTPUT associated with it. OUTPUT is created by the operating system on a queue device. The operating system writes the job dayfile to this file when the job terminates. Other information that might appear on OUTPUT as a result of processing by system routines is:

Source program listing produced by compiler

Object listings requested by compiler call in the job

Diagnostics or error messages produced during compilation

Results generated during program execution

Exchange package dump generated by the operating system when a program aborts during execution

OUTPUT always is printed or otherwise associated with a remote terminal when a job ends. The job can rewind OUTPUT and overwrite existing data, or it can evict all data with a DISPOSE control statement, but it cannot prevent the job dayfile from being printed at batch job termination.

OUTPUT is a print file with a maximum line length of 137 characters. The first character is the carriage control character which must be supplied by any user program that writes to OUTPUT. System routines supply the carriage control as needed. The remaining 136 characters of the line can be printed. Some system routines have the ability to format OUTPUT for Teletype device processing with a line length less than 136 characters.

Any file copied to OUTPUT is printed at the end of the job. If the file does not have carriage control characters at the beginning of each line, the COPYSBF utility should be used to shift each line one character to the right and insert a leading blank for single spacing control.

PUNCH

PUNCH is a file with an associated disposition code. Any data written to the file is assumed to be display code. The file is punched in Hollerith format at the end of the job.

PUNCHB

PUNCHB is a file of binary information. Any data written to it is assumed to be binary. The file is punched in standard binary format at the end of the job. Any assembled or compiled program that is written on PUNCHB is an object program that can be loaded and executed by specifying the name of the file on which the program resides.

P80C

P80C is a file of binary information. Any data written to it is assumed to be binary. The file is punched in free-form binary format at the end of the job. They are used only in special circumstances.

OTHER SPECIAL-NAMED FILES

Files with names FILMPR, FILMPL, HARDPR, HARDPL, and PLOT also have an associated disposition. The operating system defines codes for these files, but does not supply the routines needed to drive the associated hardcopy or microfilm devices. Only some installations have these devices.

ASSIGNING FILES TO A JOB

Before a file can be read or written, the operating system must be informed of the device on which the file resides. If a file is not associated with a specific device before it is created, it is written on a public mass storage device at the time an executing program calls for file open. The job does not need to inform the system of the residence of files on mass storage unless the file has special characteristics.

Files that exist only for the duration of the job are known as scratch files. They are created as they are needed and destroyed when the job terminates. The INPUT file for the job, temporary files written by the compilers during compilation, and some user files are useful only for a short time. Scratch files are created on mass storage as the file is referenced. They need not be specifically requested.

The devices on which rotating mass storage files are written are divided into two classes: public device sets and user device sets. The programmer determines the device on which a file resides by the use or absence of the REQUEST control statement and the SETNAME control statement or parameter. Public and private device sets are described later in this section.

Situations in which it is necessary to inform the operating system of the device on which a file is to be created include those when:

A file is to be subsequently declared a permanent file. Permanent files must be referenced on a REQUEST control statement with a *PF parameter.

A file is to be released to the output queue for print or punch processing. Unless the file name is OUTPUT, PUNCH, PUNCHB, or P80C, a REQUEST control statement with a *Q parameter is required.

A file is on magnetic tape. All tape files require a REQUEST or LABEL control statement that describes the characteristics of the tape data format, label, and recording mode.

A file is to reside on a private device set. A MOUNT control statement is required to associate the private device set with the job. Subsequently, each file that is to reside on the device set must be referenced in a REQUEST control statement specifying the device set name.

Existing files that must be specifically associated with the job include:

All tape files. Tape files require a REQUEST or LABEL control statement.

Permanent files. Permanent files are associated with a job through an ATTACH or GETPF control statement.

Private device set files. Permanent files are attached with an ATTACH control statement that names the device set.

The file INPUT, and all other special-named files described above, are assigned by the operating system to a mass storage device designated for input and output queue files.

DISPOSING OF FILES AND EQUIPMENT

Whether the file is to be temporary or permanent, is controlled by the programmer. All files created on mass storage are temporary files that disappear when the job terminates, unless the job includes steps to preserve the file. A file can be preserved on mass storage or on external media by transferring it to printed pages, punched cards, or magnetic tape.

Files are preserved in printed or punch card form when they are assigned a disposition code that results in processing by the line printer or card punch. Disposition codes are covered under discussions of the DISPOSE and ROUTE control statements and macros, and the preceding special-named file discourse.

Files are preserved on mass storage by cataloging them as permanent files. Permanent files are explained later in this section.

Normally, all files assigned to a job are retained by that job until termination. When the files reside on non-allocatable devices such as magnetic tapes, both the file and the hardware device are unavailable to other portions of the system for the duration of the entire job even though the file is in process for only a short part of the job.

When DISPOSE, ROUTE, UNLOAD, or RETURN is used, files can be released before job termination, making both the logical file name and the resident device available for other uses, within the circumstances noted below. Files named in UNLOAD or RETURN are unavailable for the remainder of the job. An OPEN macro issued later in the job creates another file.

New files to be retained between jobs as permanent files on mass storage must be cataloged as permanent files before the job ends. Existing permanent files return to permanent file manager jurisdiction when they are referenced in either an UNLOAD or RETURN control statement or macro. They are no longer available to the job until referenced in a subsequent ATTACH.

FILE STRUCTURE

All files on rotating mass storage are implemented through software conventions known as system-logical-records and physical record units. These conventions are also applicable to magnetic tape in SI format and card files, although the physical representations of these files are not precisely the same as for mass storage files.

The following paragraphs describe the structure of files produced by the system. They define the terms used throughout this manual; specifically:

System-logical-record (equivalent to SCOPE logical records)

Level terminators

Physical record units

Partitions

SYSTEM-LOGICAL-RECORDS AND PHYSICAL RECORD UNITS

A PRU (physical record unit) is the amount of information that can be accessed in a single read or write operation for a given device. On rotating mass storage, a PRU is equivalent to the contents of 64 central memory words.

One write operation from a higher level language program usually does not result in the creation of a single PRU, however. Routines called by compiler programs block program data in a central memory buffer during program execution, so that one record generated by the program can become part of a single PRU or a string of PRUs containing records from write calls issued by a program.

System-logical-records are written as one or more PRUs, the last of which is a short PRU or a zero-length PRU containing a record terminating marker. The terms short PRU and zero-length PRU refer to the amount of valid user data within the PRU, not to the physical size of the PRU.

A short PRU contains fewer than 64 words of user data followed by a system-supplied record terminator at the end of user data.

A zero-length PRU contains a system-supplied record terminator, but does not contain any user data.

When user data does not fill the last PRU needed to write a system-logical-record, the record terminator is appended to the data and the remaining space in the PRU is ignored. If the record terminator cannot be accommodated in the last PRU with data, a zero-length PRU is created to hold the record terminator. A zero-length PRU has only system information.

The record terminator for a system-logical-record contains a level number of 0 through 17 to indicate the relation of that record to other records in the file. The lowest level is 0; it is associated with a single system-logical-record. A higher level number defines a set of records that begins immediately after the last record of that level and continues through all system-logical-records of a lower level number until the end of a record with that level or a higher level number is encountered.

A level number of 17 establishes a partition boundary for the file. Level 17 always is recorded in a zero-length PRU. Level 17 records are written in response to a COMPASS macro WRITEF and to compiler program requests to close a file or to write an end-of-file. When a file has only one partition, the level 17 terminator marks the logical end of the file. However, a file can contain any number of partitions defined by level 17 before the physical end of the file.

To summarize rotating mass storage file structure:

Physical Structure	Logical Interpretation
One or more PRUs terminated by a short or zero-length PRU of level 0 through 16	System-logical-record of level indicated. Sets end-of-record bits in system tables.
One or more PRUs terminated by a zero-length PRU of level 17	Partition. Sets end-of-partition bits in system tables. End-of-file exits occur.
End of mass storage allocated in system RBT table.	End-of-information. Sets end-of-information bits, if any, in system tables; otherwise sets end-of-partition bits.

System-logical-records with particular level numbers can be accessed through SKIPF, SKIPB, COPYBF, and COPYCF control statements and through the COMPASS macros SKIPF, SKIPB, and READSKP.

A system-logical-record of level 16 has special meaning to the Checkpoint/Restart feature of the operating system. Consequently, level 16 should not be specified in user programs that might be checkpointed.

Sequential files are written directly in system-logical-record format. Random files are implemented through a higher-level structure imposed upon the system-logical-records. Two types of higher level structures are:

Name/number index random files using operating system routines described later in this section.

CYBER Record Manager files using the capabilities of the CYBER Record Manager. These are described in the product reference manual.

FILE DIVISIONS

The physical representation of beginning-of-information and end-of-information depends on the storage device.

Device	Beginning-of-Information	End-of-Information
Card deck	Start of first card in deck	Card with 6/7/8/9 multiple-punched in column 1
Labeled magnetic tape file	Start of data after labels	Start of EOF label
Unlabeled SI format tape	Start of data	Start of EOF label
Unlabeled S or L format tape	Load point	Undefined
Mass storage file	Start of data in system table	End of data designated in system table
ECS	Start of data in system table	End of data designated in system table

The operating system recognizes these divisions within a file:

Partitions are divisions within a file. On a mass storage file or a tape in SI format, a partition is synonymous with a system-logical-record of level 17. On an S or L tape, a partition is indicated by a tape mark. All files have at least one partition.

System-logical-records of level 0 through 16 are defined by the operating system on SI format magnetic tape and rotating mass storage. These records are divisions of a partition.

Zero-byte terminated records are divisions within a system-logical-record or within a partition of an S or L tape. These records are the representation of a single print line or single punch card processed by the JANUS routine of the operating system.

Tapes in S or L format do not have system-logical-records. For some purposes such as copy of a coded record, the operating system recognizes each physical record recorded on the tape as a single record that is logically equivalent to a system-logical-record.

The operating system recognizes only the divisions indicated above. Individual products that are supported by the operating system have different definitions of the term record. For instance, CYBER Record Manager defines eight types of records, only one of which (S type) is equivalent to a system-logical-record. CYBER Record Manager uses a slightly different definition for some record types. From a program standpoint, a record is usually associated with a single read or write request.

DEVICE SETS

All rotating mass storage devices attached to a system are grouped into device sets. One device in a set is designated as the master; it holds all tables related to the set. Each device in the system belongs to one and only one set. Two types of device sets exist:

A public device set is always available to all jobs. It is used by the system to hold system files, permanent files, and special-named files such as INPUT and OUTPUT.

Unless a job requests that a file be written to another device, files are assigned to a public scratch device.

A private device set is available to a job only by specific request. Depending on the installation, private device sets may or may not be physically mounted at all times. Files to be preserved on private device sets should be made permanent on that set. Private device sets can be used simultaneously by jobs that have mounted the device set.

Device sets can have a varying number of members within the set. Some device sets might have only a single device associated with them. The single device in such a set is both the master device for the set and the only member of the set. The set is identified by the set name. The individual members of the set are identified by a volume serial number.

A job need not know the volume serial numbers of members of device sets, however. Parameters on the REQUEST control statement that assigns a file to a device allow a member to be identified explicitly by its volume serial number or implicitly by its attributes.

Attributes are assigned when a device set is created. The attributes of most concern to applications programmers are:

Public permanent file default set. Permanent files reside on this public set unless another set is requested.

Queue set. Files with the name INPUT, OUTPUT, or any other special name reside on this set. Any file to be named in a ROUTE or DISPOSE control statement must reside on this set.

Permanent file device. A member of a public or private device set can hold permanent files when the device has the permanent file attribute.

Queue device. A device on which queue files can reside if the device is a member of the queue set.

Master device. The master device of each private device set must be known before the set can be accessed by a job.

A file on a rotating mass storage device can be of arbitrary length, and it can be segmented over more than one device. The data is recorded in a logical sequence of record blocks which can be arbitrarily scattered about the disk surface. The operating system maintains a central memory table for each file, called the record block table (RBT), in which the sequence of allocated record blocks is defined. The end-of-information position and end-of-volume position are also defined in the RBT.

PUBLIC DEVICE SET USAGE

Public device sets are the default. Unless a private device set is requested, mass storage files are on public devices. All public device sets are available to a job at all times; the MOUNT and DSMOUNT control statements applicable to private device sets are not needed for public device sets and will be ignored if enountered.

The REQUEST control statement assigns a file to a public device. Normally, a REQUEST is not needed except for:

Files that subsequently will be cataloged as permanent files;

Files that have a disposition code for printing or punching;

Files that are to reside on a particular public device set or member.

The *PF parameter of REQUEST assigns the file to a permanent file device.

The *Q parameter of REQUEST assigns the file to a queue device. A file cannot be referenced by a ROUTE control statement or DISPOSE control statement unless it resides on a queue device.

Files named INPUT, OUTPUT, PUNCH, PUNCHB, P80C or any other special-named files always reside on public devices by default. A REQUEST with a *Q parameter is not needed for special-named files.

PRIVATE DEVICE SET USAGE

A private device set is established by the following steps:

Each pack to be included in the set is blank-labeled with the LABELMS utility.

The master device is established by an ADDSET control statement that defines the name of the set, the volume serial number of the master device, the maximum number of packs that can exist in the set, and the maximum number of permanent files that can exist in the set. The master device need not be a permanent file device, but at least one member device should be designated as a permanent file device.

Members of the device set are added by additional ADDSET control statements that specify the device set name, the master device volume serial number and the volume serial number for the pack being added. Additional members are not required; the master device can be the only pack in the device set. All ADDSET control statements can define the permanent file attribute for the device being added. The queue attribute can be defined if the device is being added to the public queue set.

Since tables relating to all packs that are subsequently added to the set reside on the master device, the master device must be available each time a pack is added to or deleted from the device set and must be available each time any file is accessed from the set. The master device is also required when any of the permanent file utilities (AUDIT, DUMPF, LOADPF, or TRANSPF) references a private device set.

To access a file existing on the device set or to create a file on the device set, the job must perform the following steps:

The master device must be associated with the job by a MOUNT control statement. Since private device sets can be used by many jobs at the same time, the device might already be physically available. If not, the operator must make the master device available.

Any permanent file to be attached must be identified as a file on that particular set. The SETNAME control statement can establish the set name prior to the attach request, or the SN=setname parameter can be used on the ATTACH control statement.

The REQUEST control statement assigns a file to a private device; in addition, all files to be created on the device set must be associated with the device set by a REQUEST control statement. An SN=setname parameter explicitly names the set; an SN parameter implicitly names the set specified in the last SETNAME control statement.

Once the job has processed the files associated with the device, the device set should be disassociated from the job by execution of a DSMOUNT control statement. Execution of DSMOUNT might free a disk drive for other packs before the job ends, and thereby increase overall system throughput. If the job omits DSMOUNT, the system disassociates the device set from the job during end-of-job processing.

The REQUEST control statement is required to assign a file to a private device set. The SN=setname or SN parameter establishes the name of the set; the VSN parameter can specify a particular member of the set. The *PF parameter can be used to ensure that the file resides on a permanent file device.

The SETNAME control statement can be executed before any files are requested. SETNAME can establish the device set to which all subsequent ATTACH control statements are directed. This eliminates the need for an SN=setname parameter on each individual ATTACH control statement. It also defines the set to which REQUEST control statements with SN parameters are directed.

PRIVATE DEVICE SET EXAMPLE

1.
```
HOKEY.
LABELMS,DT=AY. PLEASE USE PACK 844A
LABELMS.       PLEASE USE PACK 844B
ADDSET(VSN=844A,MP=844A,SN=MORE,*PF)
ADDSET(MP=844A,VSN=844B,SN=MORE,*PF)
6/7/8/9
```

This job creates a device set with two members.

2.
```
SUBSTITUTE.
MOUNT(SN=MORE,VSN=844A)
DELSET(MP=844A,SN=MORE,VSN=844B)
MOUNT(SN=OTHER,VSN=123)
ADDSET,VSN=844B,SN=OTHER,MP=123,*PF.
6/7/8/9
```

This job deletes a pack from one device set and adds it to another.

3.
```
FIX UP.
PAUSE. OPERATOR PLEASE ENSURE SN=MORE, VSN=844A IS ON AN RMS DRIVE.
RECOVER,SN=MORE,VSN=844A.
6/7/8/9
```

This job runs a RECOVER on device set MORE, assuming the master device is physically on a disk drive.

4.
```
SET.
MOUNT(VSN=844A,SN=MORE)                    Mounts master device
REQUEST(TAPE5,*PF,SN=MORE)
FTN.
LGO.
CATALOG(TAPE5,PERMANENT,ID=FRIEND)
7/8/9
FORTRAN program that creates TAPE5
7/8/9
data cards for FORTRAN program
6/7/8/9
```

This job makes a permanent file on the device set MORE.

5.
```
USE A SET.
MOUNT(VSN=844A,SN=MORE)                    Mount the master.
SETNAME(MORE)
ATTACH(A,PERMANENT,ID=FRIEND)              Taken from device set MORE by default.
REQUEST(TAPE6,*PF)                         Assigned to public device since no SN parameter.
COPY(A,TAPE6)
CATALOG(TAPE6,PERMANENT,ID=FRIEND)         Makes file permanent on the permanent file default set.
FTN.
REQUEST(TAPE5,*PF,SN)                      Assigned to device set MORE as SN is specified but
                                           not equivalenced.
LGO.                                       Job uses data and file TAPE6 to create file TAPE5.
CATALOG,TAPE5,PERMFILE,ID=FRIEND.
7/8/9
FORTRAN program
7/8/9
data
6/7/8/9
```

Permanent file PERMANENT is copied from device set MORE to the public device and recataloged with the same permanent file name and owner ID. A new permanent file is created and cataloged on device set MORE.

6.

```
TWO SETS.
MOUNT,SN=OTHER,VSN=123.
MOUNT(VSN=844A,SN=MORE)
SETNAME(MORE)
ATTACH(TAPE5,PERMFILE,ID=FRIEND)

REQUEST(A,*PF,SN=OTHER)

COPY(TAPE5,A)
FTN.
LGO.

COPY(TAPE6,A)
CATALOG(A,PERM,ID=FRIEND)
7/8/9
FORTRAN program that creates TAPE6
7/8/9
data cards
6/7/8/9
```

Mounts master device.

Mounts master device.

File is taken from device set MORE because of preceding SETNAME.

File directed to device set OTHER since explicitly requested.

FORTRAN job creates file TAPE6 on system device as no REQUEST card used.

Permanent file PERMFILE is attached from device set MORE and copied to device set OTHER. A new file is created on a system device and copied to the same file on device set OTHER. Then the file on device set OTHER is made permanent.

OPERATING SYSTEM RANDOM FILES

The term random denotes several different concepts, depending on the context in which the word is used.

From a hardware standpoint, random refers to a device. All rotating mass storage devices and ECS are random access devices. Any physical address on the disk or ECS is read when the hardware driver receives a request for information at that address. This is in contrast to a sequential device, such as a card reader or tape, in which a card or tape block can be read only in the physical order in which it was written. Files written to random access devices can, but need not, have random structure.

From an applications programmer standpoint, random refers to a file structure and to the means of accessing records in a file. CYBER Record Manager and compiler products provide several different random access file structures in which each record has a key that uniquely identifies the record. The program can access any record by specifying its key, without considering the records that physically exist before or after that record. To the operating system, CYBER Record Manager files with random organization are sequential files.

From an operating system standpoint, random refers to the means by which the operating system receives input/output address information. A file on a rotating mass storage device is a random file only when the random bit is set in the file environment table (FET) which controls all file input/output. When the random bit is set and a write is issued, the system writes a record to the device, then returns

address information to the FET. The program is responsible for preserving the information returned and for respecifying that information when the associated record is to be read. See the description of the Record Request/Return Information field of the FET in section 5 for additional details.

A COMPASS programmer has the option of providing indexing routines for files in which the random bit is set, or of using the operating system supplied indexing routines. These routines create an index in which records are identified by name or by number of the entry within the index.

References to random or indexed files in section 5 and 6 assume the name/number index structure described below. No other random, indexed, or random indexed file structures are recognized by the operating system.

For information about the random file structures available through CYBER Record Manager or through various languages, see the reference manual for those products or languages.

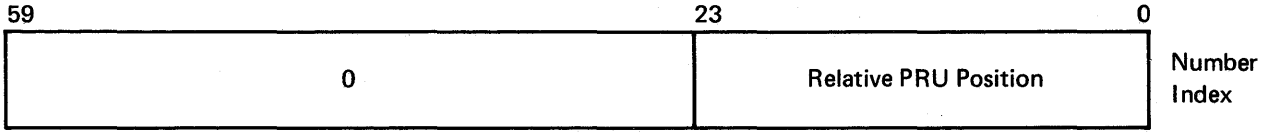
NAME/NUMBER INDEX FILES

Name/number indexed files can be created, read, written, and rewritten using the COMPASS macros OPEN, CLOSE, READIN, WRITOUT, WRITIN, and WRITER. Management of a single index level is provided through macros OPEN and CLOSE.

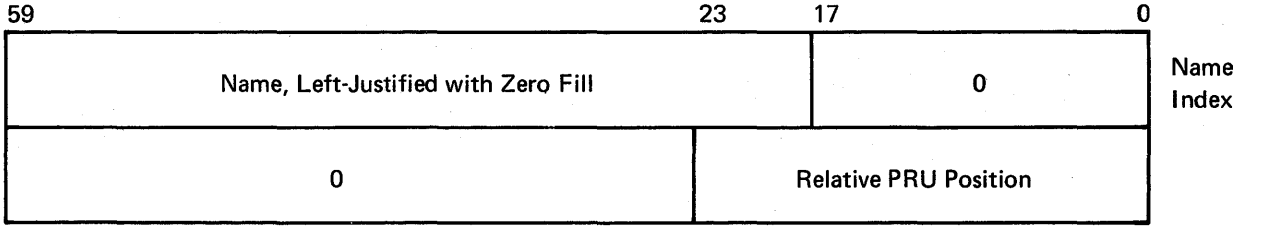
Each file has an associated index. The index contains a relative PRU position for each system-logical-record in the file. The file beginning is equivalent to the start of the record associated with the first index entry; the file end is equivalent to the end of the record associated with the last index entry. Any record can be read by identifying it in the index without the need to skip records from some beginning file position.

If a random file is to be saved, the file index must be written as the last logical record on the file. A user can write the index or call the COMPASS macro CLOSE or CLOSE/UNLOAD to write the index. CLOSE automatically writes out an index for a random file if the file contents were changed by a write with the FET random bit set. A permanent file must also have EXTEND permission before the index can be written.

The first word in the index determines how the records are referenced. The index is generated through the WRITOUT macro. A positive non-zero value indicates reference must be by number; a negative value indicates reference can be by name or number. Number index entries are one word; name index entries are two words. The number of a record is equal to the relative position of the index entry for that record; the first entry in the index points to record 1, the second to record 2, etc. If a name index is used, the record name can be 1 to 7 letters and digits. The value of index word 1 is determined when the first record is written. The formats of index entries are shown below.



Number Index



Name Index

The smallest unit of information that can be indexed is a system-logical-record. Each system-logical-record must begin in a new PRU. For the most economical index, data record length should be equal to an integral number of PRUs minus one word.

USER-DEFINED INDEX FILES

Single-level name/number indexed files can be created and maintained using system macros READIN, WRITOUT, OPEN, and CLOSE; data record management at any level lower than a system-logical-record falls to the user.

READIN/WRITOUT can be used to create and maintain index contents during program execution without using OPEN/CLOSE to manage the index records. The user must manage his index records. They could be kept on a separate file, for example.

Multi-level name/number indexed files can be created and maintained using READIN/WRITOUT and system macros OPEN and CLOSE plus a user generated sub-index management routine. A master index record would contain addresses of sub-index records interspersed throughout the file. The master index record would be processed by OPEN/CLOSE as is a single-level index record. The user routine would need to ensure that READIN/WRITOUT would reference the correct index or sub-index block.

Other index formats can be defined by supplying a user routine to format and retrieve record names and mass storage addresses. Mass storage addresses can be computed on files containing fixed length records, provided the file is not ECS resident, since the addresses are in the form of a relative PRU count and the PRU size is fixed.

PERMANENT FILES

A permanent file is a rotating mass storage file cataloged by the system, so that its location and identification are always known to the system. Frequently used programs, subprograms, and data bases are immediately available to requesting jobs without operator intervention. Permanent files cannot be destroyed accidentally during normal system operation, including normal deadstart; they are protected by the system from unauthorized access according to the privacy controls specified when they are created.

Any file associated with a job, regardless of mode or content, which resides on a permanent file device, can be made permanent at the option of the user. Unless the user explicitly requests the system to catalog a file, it is not made permanent.

Files to be made permanent should be created on devices designated for permanent files. Files can be made permanent on either a public device set or a private device set.

Privacy in permanent files is intended to minimize software interference by thwarting threats to user files from non-authorized central processor programs. The permanent file system offers a standard set of privacy controls. If an installation requires a different kind of protection, a privacy procedure can be defined to replace the standard.

In addition to normal system protection, the individual file owner can prevent unauthorized access to his permanent file. The owner can stipulate, in cataloging a file, the degree to which the file is to be protected from read, write, and rewrite access. Once a file is cataloged, it cannot be used by any job unless the necessary passwords are given when a request is made to attach the file.

Permanent files are processed by the portion of the operating system known as the permanent file manager. The permanent file manager routines create and maintain several tables: the permanent file directory contains a record of all permanent files, their cycles, and passwords; the permanent file catalog contains a record of the physical location and statistics associated with each permanent file. As long as these tables are intact, permanent files are available.

Permanent files can be processed through control statements and macros. For information pertinent only to COMPASS programmers, see section 6.

CONCEPTS

The following information describes concepts applicable to all permanent files.

FILE IDENTIFICATION

A permanent file is identified in system tables by the combined information supplied by a pfn, ID, and CY parameter when the file is made permanent with a CATALOG control statement.

pfn	Permanent file name of 1-40 letters or digits.
ID=name	Name of user responsible for file, 1-9 letters or digits. The ID specified must be unique if pfn is duplicated within the system. ID=SYSTEM is reserved for system use.
CY=cy	Cycle number 1-999. As many as five physical files can exist for each permanent file name and ID combination. Each is called a cycle. Each file shares the same ID and set of passwords. No restrictions are imposed on the content or size of any cycle, since each is a unique file.

The pfn parameter is required for both the catalog request that makes a file permanent and the attach request that associates an existing permanent file with a job. When the first seven characters of the permanent file name are the same as the logical file name, the permanent file name can serve as both the pfn and lfn parameters. If the ID is not specified, ID=PUBLIC is assumed. If the file is cataloged with ID=PUBLIC, the ID parameter can be omitted for the attach; for any other name except PUBLIC, the ID parameter is required on the attach. An installation defined password is needed to catalog a file with ID=PUBLIC.

The CY parameter is optional. Cycle numbers need not be consecutive nor contiguous; they can be created in any order. At CATALOG time, the system assigns a cycle number one greater than the largest existing cycle number in the following cases:

CY parameter is omitted

CY parameter duplicates the number of an existing cycle

CY parameter is not within range of 1-999.

System assignment of a cycle number is not possible when the cycle 999 exists, and the catalog request for an additional cycle is unsuccessful.

PERMISSIONS AND PASSWORDS

All user files have a 4-bit permission code. Each bit represents an access permission as defined below:

READ permission: Required to read a file, load a file, or copy a file.

MODIFY permission: Required to rewrite existing data or evict part of a file.

EXTEND permission: Required to evict part of a file or increase the amount of mass storage allocated to a particular file.

CONTROL permission: Required to purge a file, or catalog a new cycle of an existing pfn/ID file.

The RENAME and CATALOG functions require all four permissions.

Files in use by a job, other than permanent files, have all access permissions except for the file INPUT, which has only READ and EXTEND permissions. Permanent files have only those permissions granted by ATTACH parameters. A purged permanent file, when still associated with the job that purged it, has only those permissions it had as an attached permanent file.

Permissions are established originally by parameters on the CATALOG control statement or macro, although they can be changed through RENAME. Passwords are a string of 1-9 letters or digits. They are defined on a CATALOG control statement by the following parameters:

RD=rd	Establishes password required for read permission.
EX=ex	Establishes password required for extend permission.
MD=md	Establishes password required for modify permission.
CN=cn	Establishes password required for control permission.
XR=xr	Establishes password required for extend, modify, and control permission. Any EX, MD, or CN parameter overrides this password.
TK=tk	Establishes password that is required in addition to a password for a particular permission.

Any job using an existing permanent file must supply correct passwords in order to receive permission for functions protected by a password. On an ATTACH, RENAME, or PURGE, or on a CATALOG of a new cycle, passwords are submitted with the PW parameter, not the parameter used to create the password:

PW=pw1,pw2,pw3,pw4,pw5 1-5 passwords for specific permissions.

MULTIPLE ACCESS

More than one job might have a given permanent file attached at the same time depending on the permissions involved and the use of the RW (single write/rewrite) and MR (multi-read) parameters. Many jobs can be reading a file, but only one can have modify, extend, or control permission. Use of parameters that allow multi-access is encouraged.

When a file is cataloged initially, it remains associated with the job with all permissions, except when MR=1 or RW=1 is specified on the CATALOG request. In the absence of RW=1 or MR=1 on the CATALOG request, no other job can attach the file until the creating job returns it to the control of the permanent file manager, since any job with control permission has exclusive file access. However, an RW=1 or MR=1 parameter makes the file immediately available, on a read-only basis, to any other attaching job, but cancels all permissions except read for MR=1 and cancels control permission for RW=1.

An RW=1 or MR=1 parameter on an attach request restricts permissions that might otherwise be granted. An MR=1 cancels all permissions except read; an RW=1 parameter cancels control permission but retains modify, extend, and read permission. RW=1 overrides MR=1.

RW=0 or RW unspecified on an attach results in exclusive access if control, modify, or extend permission is granted.

QUEUED AND ARCHIVED FILES

Job requests to attach a permanent file usually are executed immediately. If a job cannot attach a file immediately, it attempts to enter that file in a queue. Four conditions can cause a job making a permanent file request to be placed into the permanent file queue:

The TRANSPF utility is running

The attached permanent file table, which is necessary for CATALOG or ATTACH, is full

File to be attached is not available for type of access requested

File to be attached is archived

The job remains in the permanent file queue until the attach can be honored or until the user or operator aborts the request.

At some installations, permanent files physically reside on rotating mass storage devices at all times and are immediately available to a requesting job. At other installations, some permanent files might be dumped to a tape through the DUMPF utility; such files are not available to a requesting job until they are reloaded through the LOADPF utility.

A permanent file physically on tape, but known to the system through permanent file table information, is defined as an archived file. The archiving process does not affect the file's status as a permanent file, so the file does not need to be re-cataloged. An archived file must be returned to mass storage before the job can read or write the file. An archived file can be purged, however, when still on tape, since only system tables are affected by a purge function.

Requests for attach of an archived file might or might not be honored depending on installation procedures. When the system receives a request for an attach of an archived permanent file, the system informs the operator of the request and indicates the VSN of the tape required. The operator mounts the specified tape, then authorizes the load by entering a command from the keyboard. The job continues when the file is available.

A request for an archived file submitted interactively through a remote terminal produces a message at the terminal:

REQUEST FOR ARCHIVED FILE – WAITING FOR CENTRAL OPERATOR DROP OR GO

In response to a GO command from the operator, the job is put into the permanent file queue, the message WAITING FOR ARCHIVED FILE is sent to the terminal user, and a job is set up at another control point to retrieve the file from tape. The INTERCOM user must wait for retrieval to be completed before the file is attached. In response to DROP, the file is not brought into the system and the attach request is terminated.

Once the WAITING FOR ARCHIVED FILE message appears at the terminal, the terminal user has the option of waiting for the file to be made available or of continuing with other tasks. An abort command after the central site operator enters GO affects the attach request itself, but does not affect the reloading of the file to mass storage. Consequently, the following procedure can save time during interactive processing:

Enter command to attach file,

Wait until WAITING FOR ARCHIVED FILE message appears,

Enter abort command,

Continue with other operations,

Reissue attach command.

The second attach command should execute immediately since the file should have been returned to mass storage while other terminal operations proceeded.

INCOMPLETE CYCLES

Incomplete cycles might exist as the result of abnormal termination of a permanent file manager function. They might also be created by a normal deadstart taking place during a permanent file function. The file is automatically purged when the file is returned, or during end-of-job processing. To remove an incomplete cycle from the system, the file must be attached with the cycle number explicitly stated and with control permission.

Execution of the AUDIT utility with an MO=I parameter reveals the existence of any such incomplete cycles.

USAGE

BATCH JOB USAGE

Permanent files are manipulated by the following control statements at a single mainframe installation. At linked multi-mainframe sites, these statements are used when the permanent file resides at the site at which the job is submitted and executed.

CATALOG	Make a local rotating mass storage file permanent with a particular name and owner. Parameters on the CATALOG statement become part of a system table that controls all further file use.

ATTACH — Associate a permanent file with a job. Parameters on the ATTACH statement must agree with privacy controls of CATALOG to establish the right to access the file.

PURGE — Delete a permanent file by deleting system table information. The file itself remains attached to the job as a local file.

EXTEND — Increase the size of an attached permanent file.

RENAME — Change system information established when the file was cataloged.

ALTER — Change the size of an attached permanent file.

When the permanent file resides at a linked multi-mainframe site other than that at which the job executes, the following statements must be used instead of the ones listed above:

SAVEPF — Create a permanent file on a public device at the system identified by the ST parameter. Parameters on the SAVEPF statement become part of a system table that controls all further file use.

GETPF — Assign permanent file residing on the system specified by the ST parameter to the job. Parameters on the GETPF must agree with privacy controls of SAVEPF to establish the right to access the file.

For a single file, the CATALOG, SAVEPF, ATTACH, and GETPF control statements can be combined as required to access the permanent file from a given system. A file cataloged with CATALOG can be attached with GETPF.

Table 3-1 summarizes parameters applicable to permanent file functions. Any parameter not applicable to a given control statement is ignored. The control statements and their parameters are explained in section 4.

TABLE 3-1. PERMANENT FILE PARAMETERS

	lfn/pfn	AC	CN	CY	EC	EX	FO	ID	LC	MD	MR	PW	RB	RD	RP	RW	TK	XR	SN	ST
CATALOG	both or one	*	*	*		*	*	+		*	*	*		*	*	*	*	*		+
SAVEPF	both or one	*	*	*		*	*	+		*	*	*		*	*	*	*	*	o	+
ATTACH	both or one	*		*	*			+	*		*	*				*			*	
GETPF	both or one	*		*	*			+	*		*	*				o			o	+
PURGE	both or one	*		*	*			†	*		*	*	*			*			*	*
RENAME	lfn pfn††	*	*	*		*		*		*		*		*	*		*	*		
EXTEND	lfn																			
ALTER	lfn																			

+ required * optional † special case o ignored with message

††applicable only when pfn is not attached

Four utility routines exist explicitly for permanent file use:

AUDIT	Reports the status of permanent files.
DUMPF	Dumps files to tape for backup or temporary storage as archived files.
LOADPF	Loads permanent files that have been dumped by DUMPF.
TRANSPF	Moves permanent files and permanent file tables between members of a device set and moves files from one device set to another.

These utilities can be called such that all permanent files are affected or that only files pertaining to a given ID, device, or use are affected.

Files to be made permanent must reside on a device that the ADDSET control statement establishes as a permanent file device. The user job can create a file on a permanent file device in two ways:

If the file is to be cataloged on a public permanent file device or on a private device whose VSN is not known, the *PF parameter should be specified on the REQUEST statement that establishes the file.

If the file is to be cataloged on a public or private device with a volume serial number known to be the number of a permanent file device, the VSN parameter should be specified on the REQUEST.

Cataloging a file results in entries in system permanent file tables. The file remains attached to the job and can be used as any attached permanent file. At the termination of the job that cataloged the file, the system detaches the file. The job can, but need not, execute a RETURN or UNLOAD function to detach the file.

INTERCOM USAGE

From the terminal, the INTERCOM user can create, attach, and purge permanent files in any of three ways:

Through standard macros within the user's own interactively run COMPASS program.

By entering the commands ATTACH, CATALOG, etc., as if they were control statements in a batch INPUT file.

By using the special INTERCOM commands FETCH, STORE and DISCARD. These commands allow the user to create and use permanent files with certain restrictions.

Files created by the STORE command cannot have any passwords. The only parameters for STORE are: filename, user id. The permanent file name and the local file name are the same, user id is required according to installation options. If a required parameter is missing, it is requested from the user.

When a permanent file has been created through the STORE command, the user can access it through the ATTACH or FETCH commands. FETCH parameter requirements are the same as for STORE.

Similarly, the DISCARD command as well as the PURGE command can be used to purge a permanent file created by the STORE command. DISCARD has the same parameter requirements as STORE, with the exception that the user id parameter can be omitted if the file is already attached. Since execution of the DISCARD control statement involves both a PURGE and a RETURN, the purged file does not remain as a local file after the DISCARD is executed.

From an INTERCOM terminal, private device sets can be used but not created. The commands MOUNT, DSMOUNT, etc., can be entered as if they were control statements in a batch input file. LABELMS, RECOVER, DELSET and ADDSET commands cannot be entered from INTERCOM. A MOUNT of the master device must be the first reference to a device set. After the master has been mounted, the REQUEST command and the permanent file commands ATTACH, CATALOG, etc., with SN parameters can be used to access device sets. A file written on a private device set can be made permanent with the STORE command. FETCH can be used to attach a device set resident permanent file only after a SETNAME command has been issued. If a private device set resident permanent file has been attached, it can be purged with DISCARD; if it has not been attached it cannot be purged with DISCARD.

If an INTERCOM job enters into the permanent file queue because a permanent file request cannot be honored immediately, the user is informed by one of the following messages:

WAITING FOR PF UTILITY

WAITING FOR APF SPACE

WAITING FOR ACCESS TO FILE

WAITING FOR ARCHIVED FILE

WAITING FOR VSN=nnnnnn,SN=setname

Examples of INTERCOM interactive permanent file use:

In these examples, the information output by the INTERCOM system on the terminal display is underlined to distinguish it from that entered by the user; this does not actually occur on the output. The symbol Λ denotes carriage return.

1. **COMMAND-STORE,MYFILE.Λ**

   ID=RKCΛ

   The installation requires a user id parameter; the user file called MYFILE is made permanent.

2. **COMMAND-FETCH,MYFILE,RKC.Λ**

   COMMAND-DISCARD,MYFILE.Λ

   During a later session the user attaches the file and then purges it.

ACCOUNTING

If the installation chooses, messages are sent to both the system and user dayfiles whenever the status of a referenced permanent file changes. The messages are as follows:

For CATALOG:	CT ID=xxxxxxxxx PFN=pfname CT CY= mmm nnnnnnn WORDS. CT SN=sssssss
For EXTEND/ALTER:	EX ID=xxxxxxxxx PFN=pfname EX CY= mmm nnnnnnnn WORDS.
For PURGE:	PR ID=xxxxxxxxx PFN=pfname PR CY= mmm nnnnnnnn WORDS.
For RENAME (old permanent file):	NM ID=xxxxxxxxx PFN=pfname NM CY= mmm nnnnnnnn WORDS.
For RENAME (new permanent file):	RN ID=xxxxxxxxx PFN=pfname RN CY= mmm nnnnnnnn WORDS.

The first two characters of each line identify the permanent file function that caused a status change. Other parameters are:

ID=xxxxxxxxx	Name which identifies the file owner or creator
PFN=pfname	Permanent file name which identifies the file
CY= mmm	Cycle number, 1-999, assigned by creator
nnnnnnnn WORDS	Amount of mass storage space occupied by the file, given in decimal number of central memory words.
SN=sssssss	Setname of file if it resides on a public set which is not the PF default.

EXAMPLES

The examples below form a continuous set. Many ATTACH, RENAME, and PURGE examples presume files established by CATALOG examples.

CATALOG EXAMPLES

The first set of examples demonstrate initial catalogs; the permanent file name is unique to the ID specified.

1. `CATALOG(LFN,LFN,ID=RENOIR)`

   `CATALOG(LFN,ID=RENOIR)`

These statements achieve the same effect. Any time the permanent file name is omitted, it is assumed to be the same as the logical file name. The cycle number is one.

2. `CATALOG(LFN1,PERMANENTFILE,ID=RENOIR,CY=10)`

   The first cycle cataloged can have a cycle number greater than one.

3. `CATALOG(LFN2,PFILE,ID=RENOIR,CY=0)`

   The cycle number of the permanent file, PFILE, is one since an illegal cycle number is specified. The cycle number must be 1 through 999. Otherwise, the parameter is ignored.

4. `CATALOG(WATER,LILIES,ID=CMONET,XR=X)`

   `CATALOG(WATER,LILIES,ID=CMONET,MD=X,CN=X,EX=X)`

   These two control statements demonstrate the XR parameter and have the same effect. X is the password for control, modify and extend access.

5. `CATALOG(AA,B,ID=SEURAT,XR=Y,CN=Z)`

   `CATALOG(AA,B,ID=SEURAT,MD=Y,EX=Y,CN=Z)`

   These two control statements have the same effect, further demonstrating use of the XR parameter.

6. `CATALOG(C,F,ID=SIGNAC,FO=IS,MD=X,EX=Y)`

   If a data validity check reveals the file is an indexed sequential, direct access, or actual key file, extend permission becomes insert permission, and modify permission becomes replace permission. If the file is not an IS, DA, or AK file, the FO parameter is ignored.

7. `CATALOG(LFF,PF,ID=MATISSE,RP=5,CY=4,RD=X,CN=Y,MD=A, TK=C,AC=777,MR=1)`

   Since the MR parameter is non-zero, LFF has only read permission upon catalog completion. The following items are defined at catalog time:

Read password	X
Control password	Y
Modify password	A
Turnkey password	C
Account parameter	777
Cycle number	4
Retention period	5 days

Assuming the previous examples to be successful initial catalogs, the following examples demonstrate new-cycle catalogs. A file already has been cataloged with the permanent file name and ID specified.

8. `CATALOG(Z,LFN,ID=RENOIR)`

   `CATALOG(Z,LFN,ID=RENOIR,CY=2)`

   These control statements catalog a cycle with a cycle number one higher than the largest (in this case 1). This new-cycle catalog does not require passwords because a control password was not defined.

9. `CATALOG(LFN22,PERMANENTFILE,ID=RENOIR,CY=10)`

   Assuming a cycle 10 already exists, this control statement causes cycle 11 to be cataloged. An invalid cycle number is treated as no cycle number. This new-cycle catalog does not require passwords because a control password was not defined at initial catalog time.

10. `CATALOG(LFF,PF,ID=MATISSE,CY=5,PW=Y)`

    If a control password is defined at initial catalog, it is necessary to submit the control password using the PW parameter. Control permission is required to add a new cycle.

11. `CATALOG(LFF,PF1,ID=PUBLIC,PW=XYZ)`

    A file can be cataloged with an ID of PUBLIC if the public password is submitted—defined by the installation as XYZ in this example. This enables an installation to define permanent files that can be attached by all users without specifying an ID.

12. `CATALOG(PERMANENTFILENAME,ID=MOREAU)`

    A catalog function is attempted using the first seven characters of the permanent file name as the logical file name. If the logical file name is omitted, the first character of the permanent file name must be alphabetic, or the job is terminated.

ATTACH EXAMPLES

1. `ATTACH(LFN,ID=RENOIR)`
   `ATTACH(LFN,LFN,ID=RENOIR)`

   Assuming catalog example 8 was successful, these two control statements perform the same function. If the permanent file name is omitted, it is assumed to be the same as the logical file name. Cycle 2 is attached since that is the highest cycle number.

2. `ATTACH(LFA,PF,ID=MATISSE,PW=X,C,EC=K)`

   Assuming catalog example 7 was successful, cycle 4 of the permanent file, PF is attached with read and extend permission. During execution the permanent file is referred to by the logical file name, LFA. A standard size ECS buffer is established for the file.

3. `ATTACH(PERMANENTFILENAME,ID=RENOIR)`

   An attempt is made to attach the permanent file, PERMANENTFILENAME, under the logical file name, PERMANE. The first seven characters must be letters or numbers and begin with a letter if the logical file name is omitted in the attach call.

4. `MOUNT(SN=SCIFI,VSN=999)`
   `SETNAME,SCIFI.`
   `ATTACH(DUNE,ID=HERBERT)`
   `SETNAME.`

   `MOUNT,VSN=999,SN=SCIFI.`
   `ATTACH,DUNE,ID=HERBERT,SN=SCIFI.`

   Both examples have the same effect, the permanent file DUNE is attached to the job. The master device of the device set SCIFI must be mounted before this function is issued.

```
ATTACH(WATER,LILLIES,ID=CMONET,MR=1)
or
ATTACH(WATER,LILLIES,ID=CMONET)
```

Assuming CATALOG example 4 was successful, these two control statements perform the same function of attaching logical file WATER with multi-read permission.

RENAME EXAMPLES

1. Assume PFILE was cataloged by owner ABC with read password X, extend password Y, and modify password Z. Control is granted automatically.

```
ATTACH(LFILE,PFILE,ID=ABC,PW=Y,Z,X)
RENAME(LFILE,PFILE2,RD=,CN=W)
```

The permanent file name PFILE is replaced by PFILE2 (if no other permanent file named PFILE2,ID= ABC exists). The read password is removed (succeeding users are given read permission automatically) and a password for control permission is cataloged. The existing passwords for extend and modify remain unchanged. Since the changes involve the permanent file name and passwords, the changes apply to all cataloged cycles of the file. This would also have been true if the owner ID had been changed.

2.
```
ATTACH(LFN,ID=UTRILLO)
RENAME(LFN,,ID=UTRILLO,RD=A,RP=9)
RENAME(LFN,LFN,ID=UTRILLO,RD=A,RP=9)
```

RENAME defines a READ password for the permanent file LFN, and redefines the retention period. Omission of the permanent file name in the first RENAME indicates no name change is to occur. The two RENAME control statements are identical in function. This example also demonstrates that more than one RENAME function can be issued consecutively.

3.
```
ATTACH(LFN,,ID=SISLEY,PW=A)
RENAME(LFN,,ID=SISLEY,RD=)
```

The definition of A as the READ password is removed from the permanent file, LFN.

PURGE EXAMPLES

1.
```
ATTACH(LFN,ID=RODIN)
PURGE(LFN)    or    PURGE(LFN,ID=RODIN)
```

Both sequences perform the same function.

When a purge is performed, permanent file table information for the file is removed, but the file remains available to the job with permissions existing when it was purged. At least control permission is implied.

2.
```
PURGE(PERMANENTFILENAME,ID=PISSARO)
```

If the purge is successful, the permanent file, PERMANENTFILENAME, no longer exists. Permanent file table information for the file is removed. The purge is not successful if the logical file name is omitted in the call and the first character of the permanent file name is not alphabetic.

3. `PURGE(PERMANENTFILE,ID=RENOIR,LC=1)`

   Assuming catalog examples 2 and 9 were successful, cycle 10 is purged.

4. ```
   ATTACH,FAUVE,PF,ID=MATISSE,PW=Y,C.
   PURGE(FAUVE)
   ```

   Assuming catalog examples 7 and 10 were successful, cycle 5 is purged and remains attached to the job as a non-permanent file FAUVE with only control permission.

5. `PURGE(DUNEMESSIAH,ID=HERBERT,SN=SCIFI)`

   Assuming the master device of the set SCIFI was mounted by this job, the permanent file DUNEMESSIAH is purged and remains as a local file with lfn DUNEMES.

6. ```
   ATTACH(RED,LASER,ID=LIGHT,PW=CONTROL)
   PURGE(BLUE,LASER,ID=LIGHT)
   ```

   Because the PF cycle specified on the PURGE control statement was already attached, (even though with a different lfn) the purge is successful with RED as the resultant local file.

ALTER/EXTEND EXAMPLE

To replace an existing cataloged permanent file by using the ALTER/EXTEND sequence:

```
ATTACH(LFN,PFN,ID=WHO,PW=MD,EX)      passwords for modify and extend are required
REWIND(LFN)
ALTER(LFN)                           release old PF data
COPYBF(NEW,LFN)                      write new data
EXTEND(LFN)                          make new data permanent
```

EXTENDED CORE STORAGE FILES

Extended Core Storage (ECS) can be used to buffer files and/or store files (as ECS resident files). Each file so designated is assigned a single buffer in the ECS paged partition. This paged buffer is assigned pages up to the limit specified by REQUEST or ATTACH. User input/output through ECS buffers or to an ECS resident file is performed in the same manner as any other mass storage input/output. ECS buffered files are more flexible than ECS resident files since ECS resident files are not allowed to overflow to other mass storage devices.

ECS BUFFERED FILES

Sequentially accessed mass storage files on public device sets can be buffered through ECS to avoid the costly access time of rotating mass storage devices each time a small amount of information is transferred. In order to optimize the access to such devices, a larger amount of information is transferred between the device and ECS at the time of each access. For each CIO call, regular smaller transfers between ECS and the user central memory buffer take place at a high transfer rate without mass storage device access.

The information read ahead (input file) or waiting to be written (output file) is stored temporarily in an ECS buffer. The underflow and overflow functions for these ECS buffers are performed automatically by the system. On a write function, system programs transfer data from the file's circular buffer in central memory to the ECS buffer. When the ECS buffer is filled to the maximum size defined by REQUEST or ATTACH, it is written to mass storage. On a read, the ECS buffer is filled in advance from disk, and data is transferred to the circular buffer in central memory as the circular buffer is emptied.

The ECS buffers are requested on a file-by-file basis through the REQUEST control statement or macro, or through an ATTACH statement or macro. A different buffer size can be specified for each file if the standard buffer size is not desired.

The data contained in an ECS buffer is written to a mass storage device only if the file is closed or exceeds the limit of the ECS buffer.

For optimum performance, the ECS buffer should be many times the size of the user's CM circular buffer. This ensures that the system overhead associated with ECS buffer management is small compared to the time saved as a result of performing fewer device accesses. Suggested relative buffer sizes are:

CM Circular Buffer	ECS Buffer
1000 octal words or less	10000 octal words or less
1001 - 2000 octal words	10000 - 20000 octal words
2001 octal words or greater	20000 octal words or more

For I/O bound programs using large central memory circular buffers there is little advantage in using I/O buffering. In general, an I/O buffer can be used to reduce the central memory buffer size while maintaining the high transfer rates associated with having large central memory circular buffers. Throughput on I/O buffered files is primarily a function of the ECS buffer size, rather than the central memory circular buffer size.

If an unrecovered ECS parity error is encountered with the EP bit set, control is returned to the user program with the error noted in the code and status field of the FET. If the error occurs with the EP bit off, a GO or DROP decision is required of the operator.

ECS RESIDENT FILES

This facility is provided as an installation option selected when the system tape is built. Except for some specific applications where a faster, limited rotating mass storage device is needed, it is generally preferable to use the I/O buffering scheme instead of ECS resident files. I/O buffering allows an overall optimization of the system.

Nevertheless, under this option any non-permanent sequential or random file can be ECS resident. ECS resident files are requested on a file-by-file basis. REQUEST has the same format as the one used for buffer allocation with the addition of the device type mnemonic of AX. If no EC parameter is present on the REQUEST, the file is limited to the default I/O buffer size specified at deadstart time. Otherwise, the EC parameter specifies the file size limit.

When an overflow occurs, i.e., all ECS pages are allocated or the maximum file size is exceeded, an error code 10 (device capacity exceeded) is stored in bits 9-13 of the code/status field and control is transferred to the user if the EP bit is on, else the job is aborted.

NOTE

If ECS is turned OFF, all requests for ECS buffers are ignored and the files requested on ECS are allocated on other mass storage devices.

MAGNETIC TAPE FILES

A single reel of magnetic tape is known as a volume. A volume set can consist of:

A single file on one volume

A multifile set on a single volume

A multivolume file extending over more than one volume

A multivolume, multifile set extending over more than one volume

All information on a magnetic tape begins after a physical reflective spot known as the load point. When this is sensed by a photoelectric cell, the tape is at its load point. Another physical reflective spot appears near the end of all tapes; it warns the software to initiate end-of-tape procedures.

The structure of a tape file, such as the number of characters in a block and the definition of end-of-information, is affected by these characteristics:

Physical recording is 7-track or 9-track

Format is SI format (standard system format), S format, or L format

Standard labels exist or do not exist

See appendix D for a summary of tape characteristics.

The default tape characteristics assumed by the system are an unlabeled 7-track tape recorded at an installation-defined default density in SI format. Any other tape density, format, or label must be explicitly declared by a REQUEST or LABEL statement.

NOISE BRACKETS (657 AND 659 TAPE DRIVES)

Noise brackets, or system noise records, are 24-bit blocks written on both sides of a damaged section of magnetic tape to aid the hardware in positioning over a spot where erasure has not been complete. Noise brackets are transparent to the user because the operating system discards them as noise when reading the tape. The size of the records causes them to fall in the category defined by ANSI as noise, and any system conforming to strict ANSI standards would discard them as such. The user should be aware, however, that the presence of noise brackets on a tape would make that tape unreadable on any system that did not discard 24-bit blocks as noise. For this reason, it is suggested the user include the IB parameter on the tape request when writing tapes for data interchange purposes, but remove it to take advantage of the added reliability associated with noise brackets when writing tapes not destined for interchange. Noise brackets are always inhibited by the operating system when phase encoded (PE density) tapes are written. The IB parameter is ignored when L tapes are written; noise brackets are never inhibited on L tapes.

TAPE MARKS

A tape mark is a short record used on SI tapes to separate label groups and files from label information. On S and L tapes, it can also separate files in addition to separating label groups. Interpretation of multiple tape marks depends on the tape format. The format of a tape mark is defined by the ANSI standard, described later in this section. These tape mark records are written by operating system routines. On S and L tapes, tape marks can be written by the COMPASS macro WRITEF.

DATA FORMAT

Three data formats exist:

- SI System default format
- S Stranger tape format
- L Long Stranger tape format (supported on 657, 667, 669, and 679 tape drives only)

SI format is assumed unless an F=S or F=L parameter appears in a LABEL control statement, or unless S or L is explicitly declared on a REQUEST control statement. Both binary and coded data can be recorded in any of these formats.

SI TAPES

SI format tape is the system standard. The structure of information on these tapes corresponds to the structure of files on rotating mass storage: each block on the tape is a physical record unit, with the end of a system-logical-record defined by the presence of a short or zero-length PRU.

The size of a PRU on tape depends on whether the data is written in coded or binary mode:

For coded tapes, a PRU is the contents of 128 central memory words.

For binary tapes, a PRU is the contents of 512 central memory words.

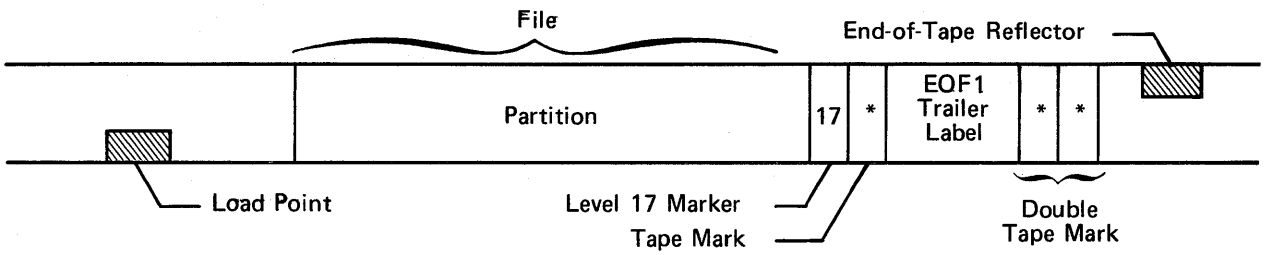
The short or zero-length PRU that terminates a record is less than full PRU size.

Each system-logical-record is terminated with a 48-bit marker that contains a level number. The marker is appended to the data in the peripheral processor when the tape is written and stripped from the data when the tape is read; only data passes from the tape to a user program in central memory.

A level number of 17 indicates an end-of-partition. Level 17 is always written as a zero-length PRU.

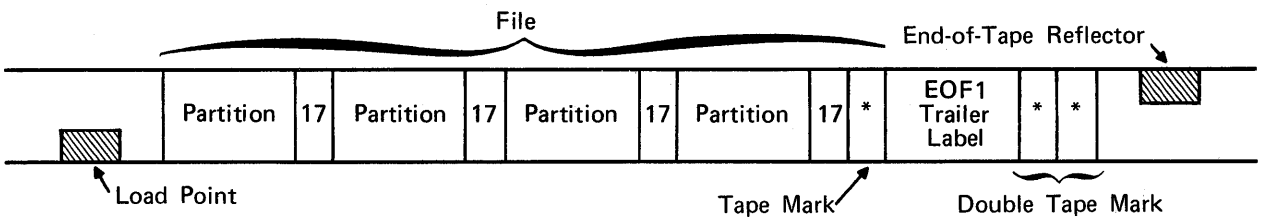
When an output file on an SI tape is closed, the operating system appends up to four items: a level 17 zero-length PRU,† a single tape mark, trailer label information for both labeled and unlabeled tapes, and a double tape mark. The file is then positioned to the beginning of the single tape mark. If more information is written to the tape, only the level 17 marker indicating an end-of-partition remains. If the tape is, instead, rewound or unloaded, the four items exist to define end-of-information.

The SI tape structure that results from a request for an unlabeled tape is:



†The presence of a level 17 PRU depends upon the procedures the programmer uses to close the file. For example, a COBOL CLOSE or a FORTRAN ENDFILE statement writes the level 17 PRU.

The SI tape structure that results from an unlabeled tape in which the file specified on the REQUEST control statement is opened and closed four times is:



The same structure is obtained when the program opens the file, writes data and issues an instruction to write an end-of-partition, repeats the data and partition instructions twice more, then closes the file.

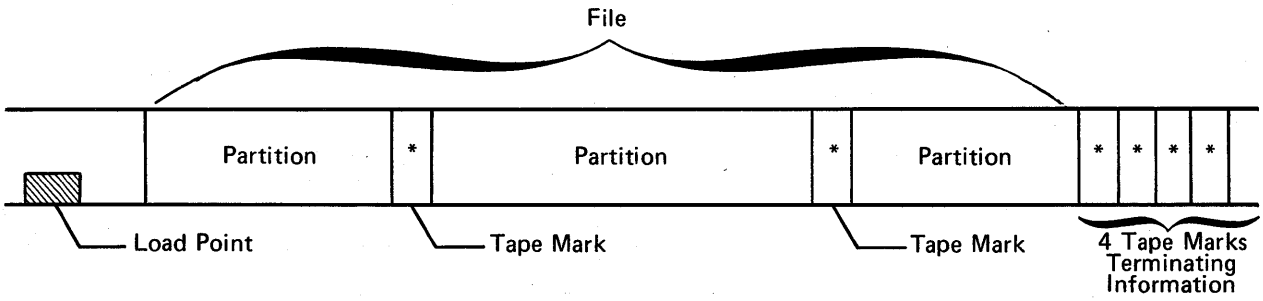
Coded information is written on 7-track SI tape in external BCD format shown in appendix A. On a 9-track SI tape, data is written in packed (binary) mode for both coded and binary data. Records are always an even multiple of 10 characters.

S AND L TAPES

Data on S and L tapes is written in physical blocks separated by interblock gaps. S tape blocks are longer than noise size and shorter than or equal to 512 central memory words. L tape blocks are longer than noise size and shorter than the user buffer for the tape.

Neither S nor L tapes contain system-logical-records of various levels as do SI format tapes. The only records are the physical blocks; and the file is physically delimited by tape marks. The last file on an unlabeled S or L tape is terminated by four tape marks, but these are not recognized as end-of-information in the same sense as a label. The user must use the four tape marks, or marks within the data, to recognize end-of-information and initiate end-of-information processing.

The S or L tape structure that results from a request for an unlabeled tape when the file is opened and closed three times, or is opened once and has two partitions written before the file is closed, is:



On a labeled S or L tape, an EOF1 label replaces the second terminating tape mark. The system recognizes the EOF1 label as end-of-information for the tape and initiates end-of-information processing as indicated by the user.

Noise size, nominally 6 characters for both S and L tapes, can be changed by the installation. Blocks shorter than or equal to noise size are not delivered to the user on read operations. An attempt to write a block shorter than or equal to noise size causes an error.

In COMPASS, the maximum logical record size (MLRS) and unused bit count (UBC) fields in word 7 of the FET should be declared when S or L tapes are processed. MLRS declares the maximum number of 60-bit central memory words in the block. The last word might not be full of data since S and L tape blocks are measured in characters instead of words. UBC must declare the number of bits not used in the last transmitted word. On a write operation, the operating system rounds down the UBC so that an integral number of characters are written. The discussion of the FET fields that appears in section 5 explains these concepts in more detail.

If the MLRS and UBC are not declared, the system assigns default values. The default for UBC is zero. The default for MLRS is 512 words for S tapes and two words less than the user buffer size for L tapes.

7-TRACK VS. 9-TRACK TAPES

Both 7-track and 9-track ½-inch magnetic tape can be processed by the operating system. Parameters on REQUEST and LABEL statements differ for recording densities, data format, and character conversion. Otherwise, label characteristics and tape usage are the same for both, except that 9-track L tapes are supported only on 669 and 679 Tape Drives.

7-TRACK TAPE

Seven-track tapes are processed by 657 and 667 Tape Drives. Data can be recorded in three densities:

LO†	200 bpi	(low)
HI	556 bpi	(high)
HY	800 bpi	(hyper)

Installation-defined default densities are used for reading unlabeled tapes and writing both labeled and unlabeled tapes in the absence of explicit declaration. The density of the label determines data density for reading labeled input tapes. However, it is always advisable to specify density because of the reading peculiarities of the tape drives. A tape label can be read at an incorrect density without causing a parity error; longer data blocks read at an incorrect density cause parity errors.

On a REQUEST statement, MT explicitly defines this tape as 7-track: LO, HI, or HY provides an implicit definition. On a LABEL statement, 7-track is assumed unless 9-track is specifically declared.

9-TRACK TAPE

Nine-track tape is processed on CDC 659, 669, and 679 Tape Units. On a REQUEST control statement, an NT parameter explicitly specifies a 9-track tape. On both REQUEST and LABEL control statements, a density specification of HD, PE, or GE implicitly specifies a 9-track tape.

†Data cannot be written at 200 bpi on CDC 667 Tape Drives although 667 drives can read 200 bpi tapes.

Under hardware control, 9-track tapes are always read at the density at which they were written. Writing density is determined by an installation default or by the density parameter on the REQUEST or LABEL control statement. Density parameters are:

HD	800 cpi (high density) applies to 659 and 669 Tape Drives
PE	1600 cpi (phase encoded) applies to 659, 669, and 679 Tape Drives
GE	6250 cpi (group encoded) applies to 679 Tape Drives

Data on SI format 9-track tape appears in memory as it exists on tape. Data is not converted while being transferred between devices.

When S or L format 9-track tapes are written or read, however, processing depends on whether the tape is binary or coded. Binary tape processing is the same as SI format tape processing, with no conversion. Data on coded S and L tapes is converted between the tape and memory. Data in the user buffer in central memory is assumed to consist of a string of 6-bit display code characters. The display code characters are mapped into 8-bit characters when written to the tape. The 8-bit characters can be a subset of either ASCII or EBCDIC, as specified by the REQUEST or LABEL control statement. Conversion from 8-bit characters to 6-bit characters takes place when the tape is read in conversion mode. The parameters on the REQUEST or LABEL control statement that select conversion mode are:

US	ASCII conversion
EB	EBCDIC conversion

TAPE LABELS

Labels on a tape consist of 80-character records that identify the volume of tape and files it contains. They are the first records after the load point marker. Labels can appear on all tapes. A label record has a particular format: the first four characters of the label are VOL1. Any tape that begins with characters other than VOL1 is considered to be unlabeled.

Two types of labels are recognized: standard system labels and Z labels.

Standard system labels conform to labels defined by the American National Standard, Magnetic Tape Labels for Information Interchange, X3.27-1969. Density of the label is the same as the density at which the data on the tape is recorded. Standard system labels are requested with a U parameter on a REQUEST control statement or macro; on a LABEL control statement or macro, the absence of a Z parameter requests a standard label.

Z labels conform to an earlier ANSI standard in which the density of the label and the density of the data were not necessarily the same. Z labels are similar to standard labels, except that character 12 of the VOL1 specifies the density of the data. When a Z-label tape is being read, the system changes the read density, if necessary, during label processing. When a Z label is written, the system treats a Z label as a standard label. Z labels are requested with a Z parameter on a REQUEST or LABEL control statement or macro.

Labeled tapes provide the following advantages for the user:

When a write ring is left inadvertently in an input tape reel, software checking ensures that no part of the tape is over-written without the express permission of the operator.

The number of blocks written on a file is recorded in the file trailer label, as well as in the job dayfile. On subsequent file reading, the count serves as additional verification that data was read properly.

The volume number field of the label ensures processing of all volumes in the proper sequence.

Multi-file volumes with ANSI labels can be positioned by label name, rather than by file count only.

The volume serial number of any ANSI label read or written is recorded in the dayfile.

Overall job processing time is reduced when the system can use the VSN field to locate and assign a tape to the requesting job without operator action at the keyboard.

The maximum benefit from the operating system tape scheduling and automatic tape assignment features can be derived only if all magnetic tapes used at an installation are labeled.

ANSI defines 10 types of labels. The first three characters identify the label type; the fourth character indicates a number within the label type.

Type	No.	Label Name	Used At	Operating System Processing
VOL	1	Volume header label	Beginning of volume	Required
UVL	1-9	User volume label	Beginning of volume	Optional
HDR	1	File header label	Beginning of file	Required
HDR	2-9	File header label	Beginning of file	Optional
UHL	†	User header label	Beginning of file	Optional
EOF	1	End-of-file label	End of file	Required
EOF	2-9	End-of-file label	End of file	Optional
EOV	1	End-of-volume label	End of volume	Required when appropriate
EOV	2-9	End-of-volume label	End of volume	Optional
UTL	†	User trailer label	End of file	Optional

†Any member of CDC 6-bit subset of ASCII character set.

Table 3-2 shows the contents and defaults of label fields. All required labels are checked by the operating system on input and generated by the operating system on output if the user does not supply them. The user must supply all optional labels to the operating system. Optional ANSI label types are accepted for reading or writing when extended label processing capabilities are requested through the XL bit of the file environment table, as explained in section 5. However, all manipulating of such labels must be done by user code. The NS parameter of REQUEST or LABEL inhibits operating system processing of labels on S or L tape.

TABLE 3-2. ANSI STANDARD TAPE LABEL FORMATS

	Character Position	Field	ANSI Name (NOS/BE 1 Name)	Length	Contents	Default Written	Checked On Input
Volume Header Label	1-3	1	Label Identifier	3	VOL	VOL	Yes
	4	2	Label Number	1	1	1	Yes
	5-10	3	Volume Serial Number	6	Any a characters	As typed from console	Yes if file assigned by VSN
	11	4	Accessibility	1	Space	Space	No
	12-31	5	Reserved	20	Spaces	Spaces	No
	32-37	6	Reserved	6	Spaces	Spaces	No
	38-51	7	Owner ID	14	Any a characters	Spaces	No
	52-79	8	Reserved	28	Spaces	Spaces	No
	80	9	Label Standard Level	1	1	1	No
First File Header Label	1-3	1	Label Identifier	3	HDR	HDR	Yes
	4	2	Label Number	1	1	1	Yes
	5-21	3	File Identifier (File Label Name)	17	Any a characters	Spaces	Yes
	22-27	4	Set Identification (Multi-File Set Name)	6	Any a characters	Volume Serial Number of first volume of set	No
	28-31	5	File Section Number (Volume Number)	4	4 n characters indicating number of volume in file	0001	Yes
	32-35	6	File Sequence Number (Position Number)	4	4 n characters indicating number of file in multi-file set	0001	Yes
	36-39	7	Generation Number	4	Not used	Spaces	No
	40-41	8	Generation Version Number (Edition Number)	2	2 n characters indicating the edition of file	00	Yes
	42-47	9	Creation Date	6	Space followed by 2 n characters for year, 3 n characters for day	Current date is used	Yes
	48-53	10	Expiration Date	6	Same as field 9	Same as field 9	Yes
	54	11	Accessibility	1	Any a characters	Space	No
	55-60	12	Block Count	6	Zeros	Zeros	Yes
	61-73	13	System Code	13	Any a characters	Spaces	No
	74-80	14	Reserved	7	Spaces	Spaces	No

TABLE 3-2. ANSI STANDARD TAPE LABEL FORMATS (Contd)

	Character Position	Field	ANSI Name (NOS/BE 1 Name)	Length	Contents	Default Written	Checked On Input
Additional File Header Labels	1-3	1	Label Identifier	3	HDR	HDR	Yes
	4	2	Label Number	1	2-9	2-9	Yes
	All other fields are not checked on input; they are written as received from user.						
First End-of-File Label	1-3	1	Label Identifier	3	EOF	EOF	Yes
	4	2	Label Number	1	1	1	Yes
	5-54	3-11	Same as corresponding HDR1 label fields				
	55-60	12	Block Count	6	6 n characters; number of data blocks since last HDR label group		Yes
	61-80	13-14	Same as corresponding HDR1 label fields				
Additional End-of-File Labels	1-3	1	Label Identifier	3	EOF	EOF	Yes
	4	2	Label Number	1	2-9	2-9	Yes
	All other fields are not checked on input; they are written as received from user.						
First End-of-Volume Label	1-3	1	Label Identifier	3	EOV	EOV	Yes
	4	2	Label Number	1	1	1	Yes
	All other fields are identical to EOF1 label.						
Additional End-of-Volume Labels	1-3	1	Label Identifier	3	EOV	EOV	Yes
	4	2	Label Number	1	2-9	2-9	Yes
	All other fields are not checked on input; they are written as received from user.						
USER Labels	1-3	1	Label Identifier	3	3 letter code: UVL, UHL, or UTL		Yes
	4-80	Any a characters. Content of these fields is not checked on input; content is written as received from the user.					

a any character
n any digit

STANDARD LABELED TAPE STRUCTURE

The four ANSI labels required are used as follows. Tape marks separating items are completely system controlled.

VOL1 — Must be the first label on a labeled tape volume. This label contains the volume serial number which uniquely identifies the volume.

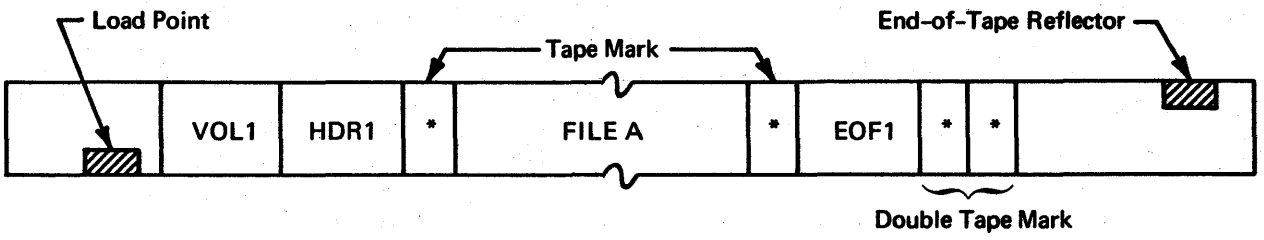
HDR1 — Required label before each file or continuation of a file on another volume. It is preceded by a VOL1 label or tape mark. Each file must have a HDR1 label which specifies an actual position number for multi-file sets.

EOF1 — Terminating label for file defined by HDR1 label; the EOF1 label marks the end-of-information for the file. A single tape mark precedes EOF1. A double tape mark written after the EOF1 label marks the end of a multi-file set.

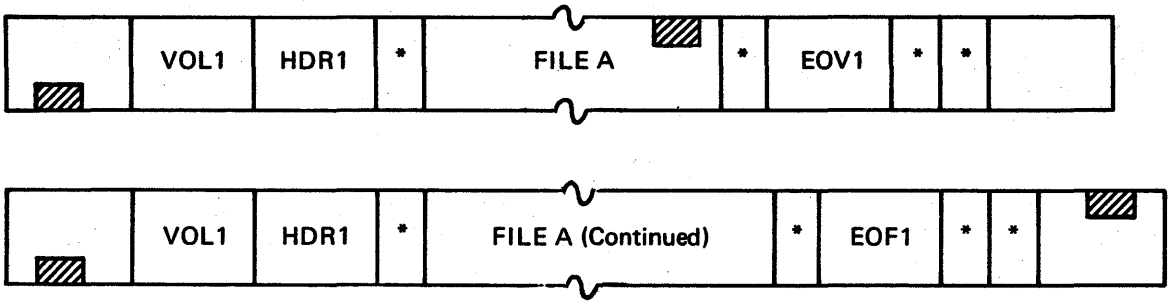
EOV1 — Required only if physical end-of-tape reflector is encountered before an EOF1 is written or if a multi-file set is continued on another volume. It is preceded by a single tape mark and followed by a double tape mark.

The structure of SI tapes that results from these required labels is shown below. The label identifier and number is used to denote the entire 80-character label in these figures.

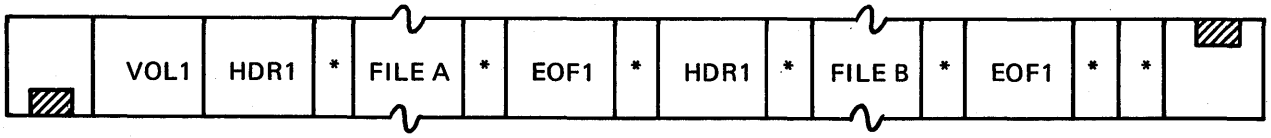
Single volume file:



Multi-reel file:



Multi-file volume structure that results from a request for a multi-file set is:



Multi-file multi-volume sets are also possible. Tape label configuration that occurs when EOF1 coincides with end-of-volume is defined in the ANSI standard.

LABELED MULTI-FILE SETS

A multi-file set consists of one or more files on one or more volumes of tape. Individual files can be accessed by name, even though their order is not known.

Labeled multi-file sets require the use of both REQUEST and LABEL statements. (LABEL statements are not required if the program can generate these fields internally.) REQUEST specifies the tape characteristics; LABEL produces the file header for individual files. LABEL must specify the set name as the M parameter. This set name is limited to six characters and must be different from any local file name. The utility routine, LISTMF, is available to list the labels of all files in an existing set. LABEL can be used to position within a set when a position number is used in the parameter list.

To create a labeled multi-file set, the following parameters should be used (parameters after the first can appear in any order). The label type must be U.

```
REQUEST(mfn,MF,U,RING,. . . )

LABEL(lfn1,M=mfn,W,. . . )        Program call to create lfn1
LABEL(lfn2,M=mfn,W,. . . )        Program call to create lfn2
```

The mfn parameter is the name of the multi-file set, 1-6 letters and digits beginning with a letter. This parameter associates the file with a particular set: all files in the set must reference it. Also, mfn cannot be used in any I/O request except as the M parameter in LABEL or POSMF requests.

RING/NORING parameters on REQUEST for the multi-file set determines the RING status for all processing of that set. RING/NORING parameters are ignored on LABEL used to position a multi-file set.

On REQUEST, the MF parameter designates the first parameter to be a multi-file name rather than a logical file name. The U parameter causes standard labels to be produced. Other parameters should establish tape density and format for the entire multi-file set. On LABEL, density and format parameters are ignored. REQUEST can include a VSN parameter.

LABEL is recommended for each file. In addition to required lfn and M parameters, optional parameters describing file header fields can appear. If a position number is not given with the P parameter, it is assumed to be one larger than that of the previous file; and the new file is written at the end of the current set. When an L parameter is used in creating a file header, future jobs can access the file by label name.

To access a labeled multi-file set, a REQUEST control statement is needed to attach the set to the job. A LABEL control statement (either U or Z) need appear only for the file to be accessed. For example, to access the third file on a volume:

```
REQUEST(MANY,MF,U,NORING ... )
LABEL(FILE3,R,M=MANY,P=3, ... )
```

When an R is specified on a LABEL statement, the set is positioned according to the P parameter, an OPEN function is issued to read the label, and the contents are checked against any corresponding parameters on the LABEL statement. Use of L instead of P causes the tape to be searched for a matching label name. If a match cannot be found, a message, FILE NAME NOT IN MULTI-FILE SET, is issued and processing stops. The same message appears also when neither P nor L is given and the end of the device set is encountered. When R is not specified, the next file in the set is opened when P and L are both omitted.

Writing on a multi-file can be done at the end of the existing set; or at some point prior to the end, existing files can be overwritten. For example, to create a new file LASTONE:

```
LABEL(LASTONE,W,M=MYSET,L=LAST)
```

Since P is omitted, the label is written at the end of existing files and given a position one greater than the last file.

If a position number is given when a label is to be written, the file is positioned as requested. If a label exists at that point, its expiration date is checked. A new label is not written over the existing one unless it is expired or the operator authorizes writing over an unexpired label. Since rewrite-in-place is not defined for tapes, rewriting a file label destroys access to the associated file and all files following it on the tape.

The assignment of a multi-file can proceed automatically with the use of a VSN under the following conditions:

A VSN statement or parameter equates the multi-file name to the physical volume of tape.

```
VSN(mfname=1234)        or        REQUEST(mfn, ..... ,VSN=1234)
```

A REQUEST statement is used to assign the multi-file name to the job.

```
REQUEST(mfname,MF)
```

A LABEL statement is used to identify the specific file by label name, equate the file to the logical file name, and identify the file as being a multi-file set member.

```
LABEL(lfn,M=mfname,L=lfn, ..... )
```

Once the multi-file name has been assigned to the job via the REQUEST statement, any file can be accessed individually via the LABEL statement. The execution of a new LABEL statement automatically prevents the preceding labeled file from being accessed.

USAGE SUMMARY

Magnetic tape files to be used or created by a job must be explicitly requested. Three control statements are involved: REQUEST, LABEL, and VSN.

The REQUEST statement can be used for all tape files – labeled, unlabeled, single file, or multi-file set. Parameters, in addition to specifying format and density, can specify processing for the file. Identifying the

tape as input or output and the type of label is sufficient to initiate label processing and checking when the file is opened. The installation default options for unloading, label processing, and parity error processing can be overridden. A volume serial number parameter for the volume (or first volume in multi-volume file) allows the system to assign the file automatically.

The LABEL statement can be used in place of a REQUEST statement for a labeled, single file volume and to write or check file header labels on single or multi-file volumes. Parameters establish label type and whether labels are to be read or written. Fields in file header (HDR1) labels are written or checked according to the values specified. If a multi-file volume is to be labeled, a REQUEST statement must first establish the multi-file name, then a LABEL statement can exist with the name and label field values for each file in the set. With LABEL, either a volume serial number or a label name can be given for identification for automatic tape assignment purposes. Automatic assignment by label name applies only when the read (R) parameter is specified by LABEL. The LABEL statement also can be used to position to a particular member of a multi-file set.

A LABEL statement can follow a REQUEST statement for the same file. Conflicts in parameters are resolved in favor of the REQUEST statement. Unresolvable conflicts are referred to the operator.

The VSN statement can be used to equate a file name with a volume serial number so that the system can assign a mounted tape automatically when it is requested by a REQUEST or LABEL statement or function. The VSN for multi-file set or for alternate volumes can be stated. Since the system accepts the first VSN equated to a file name, a VSN preceding a REQUEST or LABEL statement overrides any VSN value or supplies the omitted parameter. This VSN information is independent of label information. It is not written or checked against label fields.

Automatic tape assigning capabilities, which are selectable by installation options, speed job throughput when the programmer supplies information to allow assignment of mounted tapes without operator action. The system searches first for an eq parameter, then a VSN parameter, then a label name from among the control statements. If both the VSN and label name parameters are specified, only the VSN is used for automatic assignment. However, label verification proceeds separately and inconsistencies are brought to the attention of the operator for action. The operator has the option of assigning a VSN to a tape when it enters the system if such identification was not made by the programmer.

Only the VSN statement allows multi-volume file identifiers or alternate tape volumes to be specified. Use of the VSN statement is recommended when a job's tape file requirements change frequently.

If more than one VSN parameter is given for a single file, the first encountered is accepted. Therefore, deliberate duplication provides the programmer with the ability to override, for example, a REQUEST function specification within a program without changing the program.

The maximum number of tape drives a job uses at any time is specified by the MT (7-track) and NT (9-track) tape parameters on the job statement. Specifying more tapes than are needed can delay execution of a job. The greatest delay results from specifying a number of tapes when the job does not use any tapes. Specifying fewer tapes than needed causes the job to abort.

PRINT FILES

Print files are those with a disposition code indicating printer output. The file OUTPUT always is a print file.

Print files must have these characteristics:

1. Characters must be in 6-bit display code (IC=DIS) or 8-bit ASCII (IC=ASCII). IC is declared with the ROUTE control statement or macro. Default is DIS. Files to be printed with an extended print train (more than 64-character character set) must be in ASCII.

2. The end of a print line must be indicated by a zero byte in the lower 12 bits of the last central memory word of the line. Any other unused characters in the last word should be filled with binary zeros. For example, if the line has 137 characters (including the carriage control character), the last word would be aabbccddeeffgg000000 in octal; the letters represent the last seven characters to be printed in the line. No line should be longer than 137 characters.

3. Each line must start at the high order end of a central memory word.

4. The first character of a line is the carriage control, which specifies spacing as shown in the following table. It is never printed, and the second character in the line appears in the first position. A maximum of 137 characters can be specified for a line, but 136 is the number of characters that is printed. Table 3-4 shows carriage control characters.

   When the following characters are used for carriage control, no printing takes place. The remainder of the line is ignored.

Q	Clear auto page eject (JANUS default)
R	Select auto page eject
S	Clear 8 vertical lines per inch
T	Select 8 vertical lines per inch
PM(col 1-2)	Output remainder of line (up to 30 characters) on the B display and the dayfile and wait for the JANUS typein /OKuu
V	Specifies a new carriage control array to be loaded for a 580 printer

   The remaining carriage control characters do not inhibit printing. Only the carriage control character is not printed. Any pre-print skip operation of 1, 2, or 3 lines that follows a post-print skip operation is reduced to 0, 1, or 2 lines.

   The functions S and T should be given at the top of a page. In other positions S and T can cause spacing to be different from the stated spacing. Q and R need not be given at the top of a page as each causes a page eject before performing its functions.

The V function can be used when assigning output to a 580 printer with programmable format control (PFC). A PFC printer does not use carriage control format tapes; instead, it contains a microprocessor plus memory. PFC arrays are loaded into this memory, performing the same function as the format tape. System defined arrays are available for use (see the ROUTE control statement in section 4); however, the V function allows a user-specified array to be used. When V is the first character of the line, 6,8, or C may be specified as the second character. Other characters invalidate the function. If the second character is 6, 6-line per inch spacing is indicated. If the second character is 8 or C, 8-line per inch spacing is indicated. An 8 means that the entire array is contained on one line, and a C means that two lines are used. When two lines are used, both lines must begin with the characters VC. The data starting in column 3 defines the format array to be used in subsequent printing. The alphabetic characters A through L, the letter O, and blanks may be specified to indicate the following.

An A indicates a top of forms code. The array must begin with an A.

Letters B through K specify channels 2 through 11, respectively. Other carriage control characters contained in table 3-3 are used to skip to these channels; therefore, each of these letters should be specified at least once in the array.

An L indicates a bottom of forms code.

The letter O indicates the end of the array and must be specified as the last character in the array. It does not, however, correspond to any line on the form.

Any other characters are illegal and invalidate the array.

A maximum of 132 characters plus the end of array terminator is allowed in a 6-line per inch array, and a maximum of 176 characters plus the end of array terminator is allowed in an 8-line per inch array. An array may be less than the maximum length since the printer loops on what is specified, even if it is not a full page.

NOTE

Specifying a V (with 6, 8, or C) does not imply that 6- or 8-line per inch mode will be selected. If the user desires to change this mode, the S or T carriage controls must be used. If an array is indicated in a mode other than that previously specified by the S or T carriage controls, the array is ignored until the S or T carriage controls are used to change that mode.

If the V carriage control is specified and the printer is not a PFC printer, the printer page ejects and does not print the line(s).

The following examples illustrate typical carriage control output and its effect. In each example, the characters begin in column 1.

1. V6A B C D EFGHIJ O

   This causes the 6-line per inch buffer to be loaded with a 20-line array, implying a 20-line form.

2. V8ABCDEFGHIJO

   This causes the 8-line per inch buffer to be loaded with a 10-line array, implying a 10-line form.

3. VCA B D C
   VC E F G H I J O

   This causes the 8-line per inch buffer to be loaded with a 21-line array, A B D C E F G H I J O, implying a 21-line form.

4. V6BCDO

   This is invalid; it does not begin with an A.

5. VBA C DEO

   This is invalid; a 6, 8, or C is not the second character.

6. V8ABWCO

   This is invalid; it contains an illegal character.

TABLE 3-3. CARRIAGE CONTROL CHARACTERS

Character	Action Before Printing	Action After Printing
A	Space 1	Skip to top of next page†
B	Space 1	Skip to last line of page†
C	Space 1	Skip to channel 6
D	Space 1	Skip to channel 5
E	Space 1	Skip to channel 4
F	Space 1	Skip to channel 3
G	Space 1	Skip to channel 2
H	Space 1	Skip to channel 11
I	Space 1	Skip to channel 7
J	Space 1	Skip to channel 8
K	Space 1	Skip to channel 9
L	Space 1	Skip to channel 10
1	Skip to top of next page†	No space
2	Skip to last line on page	No space
3	Skip to channel 6	No space
4	Skip to channel 5	No space
5	Skip to channel 4	No space
6	Skip to channel 3	No space
7	Skip to channel 2	No space
8	Skip to channel 11	No space
9	Skip to channel 7	No space
X	Skip to channel 8	No space
Y	Skip to channel 9	No space
Z	Skip to channel 10	No space
+	No space	No space
0(zero)	Sapce 2	No space
-(minus)	Space 3	No space
blank	Space 1	No space

†The top of a page is indicated by a punch in channel 1 of the carriage control tape. The bottom of page is channel 7.

JOB CONTROL STATEMENTS 4

This section describes the control statements applicable to program execution and file manipulation. Utilities are also presented. The first statement described is the job statement that begins the job. Remaining control statements are in alphabetical order.

In the formats that follow, uppercase letters indicate constants and lowercase letters indicate values to be supplied by the user. Equal signs and slashes are required where they are shown within a parameter field.

CONTROL STATEMENT SYNTAX

All control statements, except the job statement that begins a job, have the same general format. They begin with a verb and are followed by parameters separated by separator characters. A terminator must follow the last parameter or the verb when no parameters are given. Blanks within the parameter list are ignored.

Verbs — 1-7 letters or digits that indicate the operation to be performed. Leading blanks can appear before the verb. The first character must be a letter. A blank immediately following the verb serves as a separator.

Separators — A separator is any character with a display code value greater than 44B, except *) . $ and blank. (A blank can be used to separate the verb from the first parameter.) The comma and left parenthesis are preferred separators. See appendix A for display code values.

Parameters — Parameter format and order depends on the individual control statements, as described below. Some parameters have more than one field; fields within parameters are separated by = / or commas.

If a parameter field includes characters other than letters, digits, or asterisks, it must be written as a literal. A literal is a character string delimited by dollar signs. Blanks within the literal are significant. If the literal is to contain the character $, two consecutive dollar signs must be written. The literal $A B$$41$ is interpreted as a B$41.

Terminators — Terminators are the characters period and right parenthesis.

Any characters after the terminator are treated as a comment. They appear on the job dayfile when the control statement is listed.

Certain control statements can be continued on one or more cards or lines. These statements are specifically noted in the following descriptions. (Refer to the appropriate product reference manual to determine which system programs allow continued control statements.) In general, the last nonblank character of the card or line to be continued must be a separator; the verb and parameter fields cannot be split between cards or lines. The final card or line must contain a terminator.

NOTE

In a system using the 64-character set, colons should not be used in a control statement except within a literal. (A single colon is permitted in a literal.) Two or more consecutive colons could give incorrect results because the operating system uses 12 zero-bits (equivalent to two consecutive colons) to signify the end of a control statement.

Control statement interpretation is described in section 6.

JOB STATEMENT

A job is identified, certain resources are requested and processing priority levels are established with the job statement. In addition, the installation might require accounting information on this statement. The first statement in a job deck or in a file to be submitted for batch execution must be the job statement. Any other statement in this position is presumed to have job statement parameters and is interpreted accordingly.

One parameter, the job name, is required on all job statements. Other parameters can be included to specify resources, priority levels, or processing time limitations. If these parameters are omitted, the operating system automatically assigns the system default values established when the operating system was installed. Parameters can be listed in any order following the job name.

All blanks and any unknown parameters that appear on the job statement are ignored. However, when improper characters are used as variables with valid parameters, the job is terminated. For example, parameters such as CM7FFF and DATA would cause job termination since CM must be followed by digits only and D followed by two letters and one or two digits.

All numbers on job statements are presumed to be octal values, unless changed by the system analyst when the operating system is installed at the user's installation.

The format of the job statement is:

xxxxx,Tt,IOt,CMfl,ECfl,Pp,Dym,MTk,NTk,CPp,STmmf.

After the terminator following the last parameter, general comments or installation defined material, such as accounting information, can appear.

xxxxx — Name the user assigns to the job to identify it to the operating system. Any combination of digits or letters can be used; the first character must be a letter. A name longer than five characters is truncated to five.

The operating system modifies the name of every job by assigning letters and digits that differ for each job as the sixth and seventh characters. This ensures unique identification if a job is entered with a name duplicating that of another job already in process. For example, if two jobs are named JOBNAME, one might be processed as JOBNA23 and the other as JOBNA34. If a job name contains fewer than five characters, all unused characters through the fifth are filled with zeros before unique sixth and seventh characters are added.

Tt

t is an octal value for the time, in seconds, which the user estimates his job requires the central processor. It must include the time required for assembly or compilation; it does not include time during which the job is in the input queue or in central memory but not using the central processor. If the job access to the central processor exceeds the value specified by t , the job is terminated abnormally. (Use of the RECOVR feature in a program allows results of execution to that point to be recovered before termination).

t cannot exceed five digits. An infinite time can be specified by 77777 or 0; the job proceeds until completed even if it exceeds the installation maximum value for t . An infinite time limit should not be used indiscriminately as certain kinds of program errors, such as an infinite loop, can result in great waste in such cases.

IOt

t is an octal value for the time, in seconds, which the user estimates his job requires for input/output. Although t cannot exceed five digits, an infinite time limit can be specified by 0. The default limit is infinite but can be changed by the installation. If the job input/output time exceeds the value specified by t , the job is terminated prematurely. (Use of the RECOVR feature in a program allows results of execution to that point to be recovered before termination.)

CMfl

fl is the maximum field length (octal number of central memory words) that the job requires.

When the CM parameter is specified, that amount of storage is allocated to the job throughout execution, unless the job itself requests a smaller amount by a REDUCE or RFL statement. If the CM parameter is not used, the system establishes field length requirements for each step of the job, expanding or contracting it as necessary. Since smaller field lengths are used whenever possible, more jobs can pass through the system in a given time period.

The system library programs, including the loader, compilers, and utilities, have an associated field length in the library tables. The field lengths are set by the installation to a judicious length for typical jobs, which should eliminate the need for the CM parameter on many job statements.

Any CM parameter on the job statement is rounded upward to a multiple of 100. The highest permissible value is defined by the installation for a given mainframe. An RFL control statement requesting a field length greater than the CM value on the job statement causes job termination. The RFL limit is the installation field length maximum if CM is not on the job statement.

ECfl

fl is the maximum amount (octal) of direct access ECS the job needs in multiples of 1000-word blocks. The highest value permitted is defined by the installation. An installation default amount (typically zero) is assigned if the parameter is omitted; and subsequent MEMORY requests from user programs are not allowed to exceed that amount.

The EC parameter is applicable only to user programs in which ECS is accessed through hardware block store instructions. It is not applicable to files stored on ECS or buffered through ECS.

Pp

p is the priority level (octal) requested for a job. The lowest executable priority level is 1. If zero is given for p , the system treats it as level 1. The installation determines the highest value permitted, but it never can exceed 7777 (octal). A value greater than the highest permitted value defaults to the installation default.

Dym

This parameter is used only in conjunction with a string of interdependent jobs. y is the dependency identifier (two alphabetic characters) assigned by the user to the entire string. m is the dependency count (number) of jobs (0-77 octal) upon which this particular job depends. Examples using the D parameter are presented in the discussion of the TRANSF statement.

MTk
NTk

MT specifies 7-track tape and NT 9-track. If both 7- and 9-track tapes are used, MTk and NTk should both be noted. The installation determines whether this parameter is necessary.

k is the maximum number of 7-track or 9-track tape units a job will require at any one time. k can range from 0 to 77 (octal), but cannot exceed the total number of tape units at the computer site. If more tape units are required at any time during job execution than are specified by k , the job will be terminated.

A job can use more than a total k tape units as long as their use is not simultaneous. For example, if k is 3 and 7-track tape units A, B, and C are assigned to the job, and an UNLOAD, but not a RETURN function is issued for the tape unit C, tape unit D can be requested for the job. This makes a total of 4 tape units used during the entire job.

CPp — This optional parameter is applicable only to systems having more than one central processor. Use of the CP parameter restricts the job to executing only on the specified processor. Omission of the parameter allows the system to select the processor for job execution; and, usually, both processors will be used during the execution of any program. p can be A or B.

On a CDC CYBER 174, CYBER 71-2x, 72-2x, CYBER 73-2x, or 6500 system, the parameter restricts job execution to one of the two identical central processors. In general, such a restriction serves no benefit. It is useful, however, for running CPU diagnostic programs.

On a CDC CYBER 74-2x or 6700 system, the two processors operate at different speeds. CPA restricts the job to the faster processor; CPB restricts it to the slower processor. When the parameter is omitted, the system chooses the faster processor when it is available.

STmmf — This optional parameter specifies a three-character identifier (mmf) of the system on which the job is to be run. For multi-mainframe environments, ST should be used to ensure that a string of interdependent jobs is executed in the same mainframe.

Examples of job statements:

```
J2,T400,CM45000,EC2,P1,DAB3,MT5,CPA.  THIS IS JOB 2 IN STRING B3

THIS JOB CARD GIVES ALL DEFAULT VALUES AND JOB NAME THISJ.

OSCAR.COBOL V4 USED

S3R2,MT1.  FIRST RUN.
```

ABS (ABSOLUTE CENTRAL MEMORY DUMP)

ABS dumps absolute addresses of central memory whether or not the addresses are within the field length assigned to the job. Both the DMP control statement and ABS can be used for an absolute dump, but DMP is limited to the first 131K of memory.

The format of ABS is:

```
ABS,from,thru.
```

When only one parameter appears, it is presumed to be the thru parameter and the dump starts at address 0. When both parameters are present, thru must be greater than first.

from — Address at which dump is to begin, 1-6 digits octal.

thru — Address at which dump is to end, 1-6 digits octal. If the value exceeds the size of memory, dumping stops at the end of memory.

The format of the output on file OUTPUT is the same as that produced by the DMP control statement. ABS can also be called using the SYSTEM macro described in section 6.

ACCOUNT (ACCOUNTING INFORMATION)

ACCOUNT supplies accounting information. The installation determines what accounting information is required and what can be optionally specified. Depending on the installation, the ACCOUNT control statement might be required immediately after the job statement; it might be allowed or disallowed elsewhere among the control statements.

The format of ACCOUNT is:

ACCOUNT,parameter list.

The dayfile message indicating the execution of ACCOUNT might be edited so that sensitive information is deleted. Illegal accounting information might cause job termination.

Some installations require accounting information on the job statement instead of the ACCOUNT control statement. Others might not require any such accounting.

ADDSET (ADD DEVICE TO DEVICE SET)

ADDSET adds members to a device set. It can be used to create a master device when parameters MP and VSN indicate the same volume serial number. Members being added must have the same device type as the master device (see LABELMS). ADDSET cannot be entered through INTERCOM.

A member device is added to an existing device set when parameters MP and VSN specify different volume serial numbers. A MOUNT statement for the master device must be issued before ADDSET can be used to add a member device.

The format of ADDSET is:

ADDSET,MP=vsn,VSN=vsn,SN=setname,NF=n,NM=m,RP=ddd,*PF,*Q,mode.

Parameters MP, VSN, SN are required; all others are optional. All parameters are order independent.

MP=vsn — Volume serial number of master device; 1-6 letters or digits, leading zeros assumed. Required.

VSN=vsn — Volume serial number of device being added; 1-6 letters or digits, leading zeros assumed. Required.

SN=setname — Name of device set created or device set to which a member is added; 1-7 letters or digits beginning with a letter. Required.

NF=n — Maximum number of permanent or queue files that can exist on the device set. Value of n cannot be less than one nor greater than 16000.

NF=n has meaning only for an ADDSET for a master device. Default is 300 (octal).

NM=m — Maximum number (decimal) of members allowed in the device set. NM=m is used by ADDSET to preallocate on the master device system tables for the member devices. For each member RBR, the system needs one PRU if the RBR is less than 62 words long, or two PRUs otherwise. For system tables ADDSET reserves a number of PRUs equal to twice NM. If each member device is to have several RBRs, NM=m should be specified as somewhat larger than the actual number of member devices. NM=m has meaning only for an ADDSET of a master device. Default is 50 (decimal).

RP=ddd — Retention period for the device set. ddd must be decimal (0 to 999) indicating the number of days before the device set expires. 999 indicates an infinite retention period. RP=ddd has meaning only for an ADDSET of a master device. Default is 31 days.

*PF — Permanent files can reside on this member of the device set. Although the master device need not be a permanent file device, at least one device in the device set must be a permanent file device.

*Q — Queue files can reside on the member device being added to the device set if it is a member of the public queue set.

mode — Recording mode for an 844 disk pack. Default is defined at installation time.

- HT Half tracking
- FT Full tracking

ALTER (CHANGE PERMANENT FILE LENGTH)

ALTER changes the end-of-information for an attached permanent file. End-of-information is set at the end of the PRU at which the file is currently positioned. ALTER is identical to the EXTEND control statement when new information has been written to the file and the current file position is at the end of the new information.

ALTER requires exclusive access to the file; an RW=0 parameter on the ATTACH control statement provides exclusive access. The permissions required depend on whether the file is being lengthened or shortened:

Extend permission is required to extend the file length;

Modify and extend permission are required to reduce the file length.

The format of ALTER is:

ALTER,lfn.

lfn — Logical file name of attached permanent file, 1-7 letters or digits beginning with a letter.

ATTACH (ATTACH PERMANENT FILE TO JOB)

ATTACH attaches a permanent file to a job, as long as parameters specified on the ATTACH control statement establish the right to use the file. Subsequent operations allowed on the file depend on the passwords submitted: turnkey, read, modify, extend, or control permission is granted only when the appropriate passwords are specified. In a multi-mainframe environment, the permanent file must reside on a device directly connected to the mainframe on which the job is executing.

When the file is attached to the job, its initial position is beginning-of-information.

The format of ATTACH is:

ATTACH,lfn,pfn,ID=name,AC=act,CY=cy,EC=ec,LC=n,MR=m,PW=pw,RW=p,SN=setname.

The first parameter establishes the logical file name. Parameters lfn and pfn are required in the order shown; all others are optional and order independent. ATTACH can be continued: if a period or right parenthesis does not appear at the end of the parameter list, column 1 of the next statement is considered to be a continuation of column 80.

lfn — Name by which file is to be known as a local file, 1-7 letters and digits beginning with a letter. If omitted or null, the first seven characters of the pfn establish lfn.

pfn — Permanent file name by which the file is known in the permanent file manager tables, 1-40 letters or digits. If omitted, lfn must be the same as the permanent file name.

ID=name — ID parameter by which the file was cataloged. Required unless the file was cataloged with ID=PUBLIC.

AC=act — Account parameter, 1-9 letters or digits.

CY=cy — Cycle number to be attached; 1-999. Default is highest existing numbered cycle.

EC=ec — Size of buffer for sequential public device set file (octal). EC is ignored when SN is specified.

EC=K	Installation standard number of blocks of ECS.
EC=nnnn	Number of 1000 (octal) word blocks to be allocated.
EC=nnnnK	Same as EC=nnnn.
EC=nnnnP	Number of ECS pages, with a page 1000 (octal) central memory words.

LC=n — Lowest cycle indicator; n must be any non-zero value. CY overrides LC except when CY=0.

MR=m — Read-only permission request; any single nonzero digit.

PW=pw — 1-5 passwords, separated by commas, for permissions required in this job. Passwords are defined by the CN, TK, RD, MD, EX parameters of the CATALOG control statement.

RW=p — Rewrite request

0	Job has exclusive file access if it has control, modify, or extend permission.
nonzero digit	Job retains modify and extend permission; any control permission is cancelled. Other jobs can attach the file with MR=1 to read the file, but cannot receive control permission.

SN=setname — Name of set on which file is cataloged, 1-7 letters or digits beginning with a letter. The master device of a private device set must be referenced on a MOUNT control statement before SN is used. If omitted the job's current permanent file default set is assumed (see SETNAME).

An ATTACH of an incomplete cycle must specify CY and any control password.

AUDIT (PERMANENT FILE SUMMARY)

AUDIT provides the status of permanent files. The user can restrict the AUDIT to an owner ID, permanent file name, or device set.

AUDIT can run in either full mode or partial mode. Items contained in the printed reports of each of these modes are listed in table 4-1.

The format of AUDIT is:

AUDIT,LF=lfn,MO=m,ID=name,PF=pfn,{AI=F / AI=P},SN=setname,VSN=vsn,AC=n.

All parameters are optional and order independent. If a terminator does not appear at the end of the parameter list, column 1 of the next card or line is considered to be a continuation of the AUDIT parameter list.

LF=lfn — Logical file name to receive the output listing created by AUDIT, 1-7 letters or digits beginning with a letter. Default is OUTPUT.

MO=m — AUDIT mode; only one of the following modes can be specified:

A	AUDIT all files (default)
X	AUDIT expired files
D	AUDIT inactive cycles
I	AUDIT incomplete files
P	AUDIT files with parity errors
R	AUDIT archived files

ID=name — Owner identification; audit all files with this identification.

PF=pfn — Permanent file name; audit all files with pfn. If PF=pfn is used, the ID=name parameter must also be used.

AI=F — Full two-line output for each file audited. Default.

AI=P — Partial one-line output for each file audited.

SN=setname — Name of device set to be audited, 1-7 letters or digits beginning with a letter. Master device for this device must have been previously mounted.

VSN=vsn — Volume serial number of device to be audited, 1-6 digits or letters beginning with a letter. All files residing on this device are audited. Master device for this device set must have been previously mounted. SN=setname parameter must also be specified.

AC=n — Account number; audit all files with this 1-9 character account number.

TABLE 4-1. ITEMS LISTED BY AUDIT

	All Files	Archived Files	Expired Files	Files of Same ID	Files on Specified Device	Partial Audit	Full Audit or Account
Account Parameter	X	X	X	X	X	X	X
Creation Date (ordinal)	X	X	X	X	X	X	X
Cycle Number	X	X	X	X	X	X	X
Date of Last Alteration (optional)	X	X	X	X	X	X	X
Date of Last Attach (optional)	X	X	X	X	X	X	X
Expiration Date (optional)	X	X	X	X	X	X	X
Flags†	X	X	X	X	X		X
Length Number of PRUs determined by Installation Parameter	X		X	X	X	X	X X
Length in RBs	X		X	X	X		X
Number of Attaches	X	X	X	X	X		X
Number of Extends	X	X	X	X	X		X
Number of Rewrites/Alters	X	X	X	X	X		X
Owner ID	X	X	X	X	X	X	X
Permanent File Name	X	X	X	X	X	X	X
Set Name	X	X	X	X	X	X	X
Subdirectory Number	X	X	X	X	X		X
Time of Last Alteration	X	X	X	X	X		X
Time of Last Attach	X	X	X	X	X		X
First VSN	X	X	X	X	X	X	X
VSN of Dump Tapes (first/last)	X	X	X	X	X		X

†Flags are:

A Archived file
C RB conflict file
E Parity error in file
N New version file
P Positioned file
R Random file
S CYBER Record Manager IS, DA, or AK file

BKSP (BACKSPACE SYSTEM-LOGICAL-RECORD)

BKSP backspaces one or more system-logical-records on rotating mass storage, ECS, or SI format tape. Backspacing terminates when beginning-of-information is encountered.

The format of BKSP is:

BKSP,lfn,n,C.

Parameters are positional; only lfn is required.

lfn	Name of file to be backspaced, 1-7 letters or digits beginning with a letter.
n	Number of system-logical-records to be backspaced, 1-262143 (decimal). Default is 1.
C	File to be backspaced is coded. Default is binary.

CATALOG (CREATE PERMANENT FILE)

CATALOG makes an existing local file a permanent file by creating entries in permanent file manager tables. A permanent file is known in these tables by a permanent file name unique within an owner ID. As many as five cycles can exist with the same permanent file name and ID but different cycle numbers.

The local file must have all permissions in order for a new permanent file name and ID to be entered in the permanent file manager tables. When the first cycle of a permanent file is created, the values for XR, EX, CN, MD, TK, and RD define the passwords which are to be used in future references to all cycles of this permanent file. Consequently, these parameters are ignored for a new cycle catalog. Any control password or turnkey password defined must be specified with the PW parameter to create a new cycle of a permanent file.

The local file must reside on a member of a public device set or on a member of a private device set designated for permanent files. A *PF parameter on a REQUEST control statement prior to file creation ensures proper file residence. An SN parameter on the REQUEST determines the device set for the file.

Once the file is cataloged, it remains available to the job as a local file with all permissions, unless the RW parameter or MR parameter cancels some permissions.

The format of CATALOG is:

CATALOG,lfn,pfn,ID=name,AC=act,CY=cy,CN=cn,EX=ex,FO=fo,MD=md,MR=m,PW=pw,RD=rd,RP=rp,
RW=p,TK=tk,XR=xr.

The first two parameters are required in the order shown. All other parameters are order independent. CATALOG can be continued: if a period or right parenthesis does not appear at the end of the parameter list, column 1 of the next statement is considered a continuation of column 80.

lfn — Logical file name by which file is presently known to the job, 1-7 letters or digits beginning with a letter. If omitted, the first 7 characters of pfn are assumed. This name does not become part of the permanent file identification.

pfn — Permanent file name by which the file is known in permanent file manager tables, 1-40 letters or digits. If omitted or null, lfn becomes the permanent file name.

ID=name — Owner or creator of file. Required unless the installation is cataloging the file with ID=PUBLIC.

AC=act — Account parameter, 1-9 letters or digits.

CY=cy — Cycle number of file with same pfn/ID combination, 1-999. If omitted, illegal, or not unique, cycle number is one greater than highest existing cycle number. If a cycle 999 exists, automatic cycle number assignment cannot take place.

CN=cn — Password for control permission (purge or catalog new cycle), 1-9 letters or digits.

EX=ex — Password for extend permission, 1-9 letters or digits.

FO=fo — File is CYBER Record Manager IS, DA, or AK organization. Permissions are defined in terms of Record Manager logic: extend is equated with adding new records, modify with deleting or replacing records. If the file is not IS, DA, or AK organization, this parameter is ignored.

MD=md — Password for modify permission, 1-9 letters or digits.

MR=m — Multi-read indicator

0	No other job can attach file while this job is in execution. Default.
nonzero digit	Other jobs can attach file immediately for read only.

PW=pw — Control password. Required to catalog a new cycle of the same pfn/ID or to catalog a file that has ID=PUBLIC.

RD=rd — Password for read permission, 1-9 letters or digits.

RP=rp — Retention period indicating the number of days file is to be retained, 0-999. Infinite retention is 999, although an installation might change this. Default is installation defined. Installation procedures determine the future of the file once the retention period expires.

RW=p — Rewrite request

0	Job has exclusive file access if it has control, modify, or extend permission.
nonzero digit	Job retains modify and extend permission; any control permission is cancelled. Other jobs can attach the file with MR=1 to read the file, but cannot receive control permission.

TK=tk — Password for turnkey required in addition to RD, MD, EX, or CN, 1-9 letters or digits.

XR=xr Password for modify, extend, and control permission, 1-9 letters or digits. Any MD, EX, or CN parameter overrides XR for the specified parameter only.

When a file is cataloged with a pfn unique to the ID, these parameters are applicable:

AC, CN, CY, EX, FO, MD, MR, PW, RD, RP, RW, TK

When a new cycle is cataloged with the same pfn and ID of an existing permanent file, the new cycle has the same set of passwords as the original file. Any control permission passwords must be specified on the CATALOG that establishes a new cycle. These parameters are applicable to a CATALOG for a new cycle:

AC, CY, FO, MR, PW, RP, RW

Any permanent file parameter not applicable to CATALOG is ignored.

CKP (CHECKPOINT REQUEST)

CKP requests a checkpoint dump be taken during job execution. Each time a checkpoint dump is taken during job execution, a file is written containing information needed to restart the job at that point. The system numbers each checkpoint dump in ascending order.

The format of CKP is:

CKP.

The checkpoint/restart system facility captures the total environment of a job on magnetic tape so the job can be restarted from a checkpoint, rather than from the beginning of the job. Total environment includes all files associated with the job. For mass storage files, the complete file is captured, including data from any ECS buffers and the relative position within that file. For magnetic tape files, only the relative position on the tape is captured so the tape can be properly repositioned during restart. (See the RESTART utility.)

Checkpoint/restart cannot handle the following items:

Rolled-out jobs
Random files (except random permanent files)
Multi-file volumes
ECS resident files

The job should request a dump tape with a REQUEST or LABEL control statement that indicates the tape is to be used for checkpoints. The tape must be SI data format and default density, but can be either 7-track or 9-track and labeled or unlabeled. Either a 7-track or 9-track tape can be assigned by the operator when an MN parameter appears in REQUEST. Only one tape can be defined for checkpoint dumps per job. If no tape is supplied, checkpoint defines an unlabeled tape for its use at the time the checkpoint occurs with the following request statement:

REQUEST,CCCCCCC,CK,MN,RING.

Checkpoint/restart defines the following files for its use:

CCCCCCC CCCCCCI CCCCCCM CCCCCCO

The user should refrain from using these file names. User system-logical-records should not have a level 16, since checkpoint uses level 16 for internal processing.

COMBINE (RECORD CONSOLIDATION)

COMBINE consolidates one or more consecutive system-logical-records in one file into one level 0 system-logical-record on a second file. COMBINE is applicable only to files with system-logical-record structure: files cannot be S or L tapes. COMBINE terminates at a partition boundary.

The format of COMBINE is:

COMBINE,lfn1,lfn2,n.

Parameters lfn1 and lfn 2 are required.

lfn1	File from which one or more system-logical-records is read, 1-7 letters or digits beginning with a letter.
lfn2	File to which one system-logical-record is written, 1-7 letters or digits beginning with a letter.
n	Number (decimal) of system-logical-records in lfn1 to be written onto lfn2. Default is 1.

The job is responsible for positioning of both files.

COMMENT (ADD COMMENT TO DAYFILE)

COMMENT inserts a formal comment into the job dayfile. The comment is displayed at the operator console as part of the job dayfile and the job continues, so the operator might not see the comment. The PAUSE control statement should be used instead of COMMENT when the comment is to be brought to the attention of the operator, since PAUSE stops the job until the operator acknowledges the PAUSE.

The format of COMMENT is:

COMMENT.comment

The period is required. The comment can begin in any column after the period; no ending punctuation is required.

comment	String of 72 characters. Any character can be specified, including those otherwise used as punctuation.

Only the comment appears in the dayfile; the word COMMENT does not. The first 40 characters of the comment, including any leading blanks, appear on the first line. Any additional characters appear on a second line in the dayfile.

COMPARE (COMPARE FILES)

COMPARE compares one or more consecutive system-logical-records in one partition with the same number of consecutive system-logical-records in a partition on another file. Comparison begins at the current position of each file and continues until the number of system-logical-records of the specified level or higher level has been processed from the first file. COMPARE terminates if a partition boundary is encountered.

Files to be compared can reside on rotating mass storage, ECS, or magnetic tape.

COMPARE can be used with an S or L tape when record size does not exceed PRU size for an SI tape. When a tape file is to be compared with a file not on tape, the tape file must be specified first in the COMPARE parameter list.

The format of COMPARE is:

COMPARE,lfn1,lfn2,n,lev,e,r.

Parameters lfn1 and lfn2 are required; all others are optional. All parameters are order dependent.

lfn1, lfn2	Names of files to be compared, 1-7 letters or digits beginning with a letter.
n	Number (decimal) of system-logical-records of level lev or higher in lfn1 to be compared to lfn2. Default is 1.
lev	Record level number (octal). Default is 0.
e	Number (decimal) of non-matching word pairs to be written to the OUTPUT file for each non-matching record. Default is 0.
r	Number (decimal) of non-matching records to be processed during the comparison. Included in non-matching record OUTPUT file if the e parameter is given. Default is 30000.

Both the contents of the record and the system-logical-record terminator must be identical for the utility to declare both files identical. When all pairs of records are identical, COMPARE writes the message GOOD COMPARE to the dayfile; otherwise the message is BAD COMPARE. A discrepancy between levels of corresponding records is listed on OUTPUT, and the comparison is abandoned, leaving the files positioned immediately after the unlike record terminators.

A bad compare produces a message on the file OUTPUT. When the e and r parameters are specified, information on OUTPUT can identify the non-matching records. The first record on each file is number 1.

COMPARE determines whether a tape file is binary or coded mode in the following way. File names are those of example 4 below. The first record of the first-named file (GREEN) is first read in binary mode. A redundancy check is produced; the file is backspaced and re-read in coded mode. If another redundancy check occurs, the fact is noted in file OUTPUT, the corresponding record of the second-named file (BLACK) is skipped over, and the process begins again. If the coded read is successful, the corresponding record of BLACK is read in coded mode. If this record of BLACK produces a redundancy check, the fact is noted in file OUTPUT, and nothing further is done with that record. Each record of file BLACK is read in the same

mode as that in which the corresponding record of GREEN was successfully read; but if the record GREEN was unsuccessfully read in both modes, the record of BLACK is read in the same mode as the preceding record of BLACK. Once a record of GREEN has been read without redundance, following records of GREEN are read in the same mode until a change is forced by a redundancy check.

Examples of COMPARE usage:

1. COMPARE(RED,BLUE)

   Compares next system-logical-record on file RED with next record on file BLUE.

2. COMPARE(RED,BLUE,6)

   Compares next six system-logical-records. Each record level on file RED must have the same level as the corresponding record on file BLUE for a good compare.

3. COMPARE(RED,BLUE,3,2)

   Compares two files from their current positions to and including the third following end-of-section mark having a level number of 2 or greater.

4. COMPARE(GREEN,BLACK,3,2,5,1000)

   Comparison is the same as the previous example; but the first five discrepancies between corresponding words in the files plus their positions in the record are listed on OUTPUT. Positions are indicated in octal, counting the first word as 0. The limit of pairs of discrepant records to be read is 1000. If two long files are compared, for instance, 20 might be used as the record parameter, so that a large number of discrepancies are described in detail; but if, through an error, the two files are completely different, an enormous and useless listing is not produced. Furthermore, the comparison is abandoned if this limit is exceeded, and the files are left positioned where they stand.

COPY (COPY TO END-OF-INFORMATION)

COPY copies one file onto a second file until a double end-of-partition (empty partition) or end-of-information is encountered on the first file. If end-of-information is encountered on the first file, enough end-of-partitions are written on the second file to ensure that it has a double end-of-partition.

Both files are backspaced past the last end-of-partition written unless a backspace is illegal on the device or end-of-information was encountered.

The format of COPY is:

COPY,lfn1,lfn2.

Parameters are order dependent and optional.

lfn1 File to be copied onto lfn2, 1-7 letters or digits beginning with a letter. Default is INPUT.

lfn2 File onto which lfn1 is copied, 1-7 letters or digits beginning with a letter. Default is OUTPUT.

COPY is intended for use with files residing on disk or on binary SI format tapes. COPY gives undefined results when used with S or L tapes or with labeled or coded tapes.

COPY can be used with any CYBER Record Manager file that resides on a PRU device. lfn1 is copied through end-of-information or a double end-of-partition. File format is not changed; FILE control statements are ignored (see the CYBER Record Manager Reference Manual).

COPYBCD (COPY LINE IMAGE FILE)

COPYBCD reformats files of line images. It is used most often to produce a tape file that can be listed off-line. Each line image of the input file is assumed to be terminated by a 12-bit byte of zeros in the low order position of the last word of the line image. COPYBCD writes each line image as a 148-character record, with the zero-byte line terminator converted to blanks on the output file.

When a partition boundary is encountered on the input file, a printer carriage control character for a skip to top of next page is written on the output file before an end-of-partition is written. Thus, the final printed output begins each partition at the top of a new page. Stray characters appear at the top of this page as a result of the skip and end-of-partition on the output file.

The format of any output tape is determined by the REQUEST or LABEL control statement in the job.

The format of COPYBCD is:

COPYBCD,lfn1,lfn2,n.

All parameters are positional and optional.

lfn1 Name of input file to be copied onto lfn2, 1-7 letters or digits beginning with a letter. Default is INPUT.

lfn2 Name of output file onto which lfn1 is to be copied, 1-7 letters or digits beginning with a letter. Default is OUTPUT.

n Number of partitions (decimal) to be copied, $0<n<2^{18}-1$. Default is 1.

COPYBF AND COPYCF (COPY BINARY AND CODED FILES)

COPYBF and COPYCF copy binary files and coded files, respectively, to other files. The minimum field length for these routines is 5000 (octal). When L tapes are copied, the minimum is 1000 (octal), plus twice the length of the largest physical record to be copied.

COPYBF and COPYCF copy partitions delimited by level 17 record terminators on PRU devices (SI tapes and mass storage) and by tape marks on S and L tapes. Copy continues until the specified number of partitions has been copied or end-of-information is encountered. An EOF label on a tape multi-file set is considered to be end-of-information. An informative message is entered in the job dayfile when the copy terminates.

These utilities produce a file with a specific structure. If an exact duplication of the input file is required, COPY should be used. Alternatively, some appropriate sequence of COPYBR/COPYCR/COPYBF/COPYCF with explicit record or file counts, or other file positioning utilities can be used.

The format of COPYBF is:

COPYBF,lfn1,lfn2,n.

All parameters are order dependent and optional.

lfn1	Name of file from which information is to be copied, 1-7 letters and digits beginning with a letter. Default is INPUT.
lfn2	Name of file to which information is to be copied, 1-7 letters and digits beginning with a letter. Default is OUTPUT.
n	Number of partitions to be copied, $0<n<2^{18}-1$ (decimal).

The format of COPYCF is:

COPYCF,lfn1,lfn2,n.

Parameters are discussed under COPYBF.

If an end-of-information is encountered on the input file before the number of partitions specified by the n parameter have been copied, the copy operation ceases (but not aborts) at that point. An end-of-partition is written on lfn2, and is not backspaced over. A dayfile message indicates the number of partitions copied before end-of-information was encountered.

When these utility routines detect an end-of-volume for a tape, the next volume is requested, label checking/writing is performed for labeled tapes, and the function continues normally on the next volume.

When a file with system-logical-records is copied to an S or L tape, each system-logical-record becomes a physical tape block. Each level 17 record delimits a partition. Similarly, when an S or L tape is copied to a PRU device, each physical record becomes a system-logical-record of level 0. A tape mark on an S or L tape delimits a partition. An informative message on the dayfile notes that levels 1 through 16 lose their level indicator on an S or L tape.

For the record and block types indicated below, CYBER Record Manager end-of-partition (EOP) is equivalent to a NOS/BE 1 end-of-partition. The routines COPYBF and COPYCF can be used to copy a specified number of partitions. All other considerations are the same as for copying system files.

Device	Block Type	Record Type
SI tapes and mass storage	C	F,D,R,T,U,S,Z
	K	F,D,R,T,U,Z
S and L tapes	C	F,D,R,T,U,Z
	K	F,D,R,T,U,Z†
	E	F,D,R,T,U,Z†

†A copy from an S/L device to a system device might add extraneous system CYBER Record Manager defined end-of-section terminators to a file.

Although not primarily implemented for that purpose, these routines are capable of limited format conversion. Table 4-2 shows format conversion copies that can be handled successfully.

TABLE 4-2. COPYxx FORMAT CONVERSION

INPUT	OUTPUT		
	SI Tapes and Mass Storage	S Tape	L Tape
SI Tapes and Mass Storage	Yes	Yes [1, 5]	Yes [2, 5]
S Tape	Yes [3, 4, 5, 7]	Yes [3, 6, 7]	Yes [3, 6, 7]
L Tape	Yes [3, 4, 5]	Yes [3, 6]	Yes [3, 6]

[1] If the system-logical-record or L tape physical record is greater than 512 words, the copy terminates with an error message.

[2] If the system-logical-record is greater than the copy buffer size, the copy terminates with an error message.

[3] If the S tape physical record is greater than 512 words or the L tape physical record is greater than the copy buffer size, the system aborts the copy with an error message.

[4] If the S or L tape record is not a multiple of 10 characters, the last word of the system-logical-record is filled with zero bits; and an informative message is issued when the copy finishes.

[5] If a 9-track coded S or L tape is used, character conversion takes place. Four 8-bit characters on input convert to four 6-bit characters in memory. Four 6-bit characters from memory convert to four 8-bit characters on tape. An informative message concerning this conversion is issued when the copy finishes.

[6] If a 9-track coded S or L tape is used, character conversion takes place between files; and an informative message concerning this conversion process is issued when the copy finishes.

[7] The largest 9-track tape record that can be copied by COPYBR or COPYBF is 3840 8-bit characters. A record of 5120 characters can be copied by COPYCR/COPYCF.

COPYBR AND COPYCR (COPY BINARY AND CODED RECORDS)

COPYBR and COPYCR copy binary logical records and coded logical records, respectively, to output files. The minimum field length for these routines is 5000 (octal). When L tapes are copied, the minimum is 1000 (octal), plus twice the length of the largest physical record to be copied.

COPYBR and COPYCR copy physical records on S or L tapes and system-logical-records on PRU devices (SI tapes and mass storage). Copy continues until the specified number of records has been copied or end-of-information or end-of-partition is encountered. An EOF label on a tape multi-file set is considered to be end-of-information. An informative message is entered in the job dayfile when the copy terminates.

These utilities produce a file with a specific structure: the last item on the output file is an end-of-partition that the utilities write, then backspace over. If an exact duplication of the input file is required, COPY should be used, as noted with the COPYBF and COPYCF utilities.

The format of COPYBR is:

COPYBR,lfn1,lfn2,n.

Parameters are order dependent and optional.

lfn1	Name of file from which information is to be copied, 1-7 letters and digits beginning with a letter. Default is INPUT.
lfn2	Name of file to which information is to be copied, 1-7 letters and digits beginning with a letter. Default is OUTPUT.
n	Number of records to be copied, $0<n<2^{18}-1$ (decimal). Default is 1.

The format of COPYCR is:

COPYCR,lfn1,lfn2,n.

Parameters are discussed under COPYBR.

If an end-of-partition is encountered on the input file before the number of records specified by the n parameter have been copied, the copy operation ceases (but not aborts) at that point. An end-of-partition is written on the output file, but it is not backspaced over. A dayfile message indicates the number of records copied before the partition boundary was encountered.

A formatted FORTRAN write to a PRU device can produce more than one line per logical record. When COPYCR is used to copy the file to an S tape, the line images are not detected as separate records.

When COPYBR or COPYCR is used to copy one S or L tape to another, each tape block copied is counted as a logical record and is converted to a system-logical-record level zero. Similarly, each system-logical-record of an input file becomes a physical record of an S or L format output file.

When these utility routines detect an end-of-volume on a tape, the next volume is requested, label checking/writing is performed for labeled tapes, and the function continues normally on the next volume.

If a partial logical record (a record not terminated with a system-logical-record mark) is encountered on the input file before an end-of-partition or end-of-information is encountered, information in the partial record is written to the output file as a logical record of level zero (or a physical tape block for an S or L tape).

For the record and block types indicated below, CYBER Record Manager end-of-section (EOS) is equivalent to a system-logical-record of level 0. The routines COPYBR and COPYCR can be used to copy a specified number of sections for these file structures.

Device	Block Type	Record Type
SI tapes and mass storage	C	F,D,R,T,U,S,Z
S and L tapes	none; EOS and EOR are not equivalent	

For CYBER Record Manger W type records, both end-of-section and end-of-partition are written as a system-logical-record of level 0. COPYBR or COPYCR can be used to copy a specified number of sections and partitions. In determining the number of records to be copied, the user should be aware that the operating system cannot distinguish between EOS and EOP defined for W type records. The copy terminates when the specified number of records has been copied, or when EOI is encountered on lfn1. For W type records, COPYBR and COPYCR copy to end-of-information.

Refer to table 4-2 with the COPYCF utility for a list of successful format conversions.

COPYN (CONSOLIDATE FILE)

COPYN consolidates or merges files. System-logical-records from up to ten binary input files can be extracted and written on one output file. Input can be from tape, card, or mass storage files. Output can be to a tape, card, or mass storage file.

Directive statements on file INPUT determine the order of the final file. Several tapes can be merged to create a composite tape. A routine can be selected from a composite tape, temporarily written on a scratch tape, and transmitted as input to a translator, assembler, or programmer routine, eliminating the need for tape manipulation by the second program. In its most basic form, COPYN can perform a tape copy.

The format of COPYN is:

COPYN,f,outlfn,inlfn1, ...

Parameters are order dependent and required. Up to 10 inlfn parameters can be specified.

f — Format of output record.

- 0 — Copy records verbatim.
- nonzero — Omit ID from record.

outlfn — Logical file name of output file, 1-7 letters or digits beginning with a letter.

inlfn — Logical file name of input file, 1-7 letters or digits beginning with a letter.

System-logical-records to be copied might or might not have an ID prefix table containing the name of the program or the name associated with the record. A record ID format consists of the first seven characters of the second word of each record. If records do not contain an ID, record identification directives must specify the record number (the position of the record from the current position of the file). Records without an ID are copied verbatim to the output file.

Format of the binary input files depends on the storage media. A binary tape file consists of the information between the load point and a double end-of-partition; this file can contain any number of single end-of-partition marks. A mass storage file ends at end-of-information; a card file must be terminated with a 7/8/9 card.

On the output file, a file mark for an output tape is written by using a WEOF statement in the desired sequence; or it can be copied in a range of records and counted as a record.

Deck structure for a COPYN job in which all input files are other than INPUT:

Job statement
REQUEST statements as necessary
COPYN call
7/8/9
COPYN directives
6/7/8/9

COPYN DIRECTIVE STATEMENTS

Directive statements for COPYN use are REWIND, SKIPF, SKIPR, WEOF, and record identification statements. These statements are read from INPUT when COPYN executes. The directive statements are free-field; they can contain blanks, but must include the separators indicated in each statement description. The ordering of the directive statements establishes the material written on the output file. Directive statements are written on the file OUTPUT as they are read and processed. When an error occurs, the abort flag is set; and a message is printed on OUTPUT followed by the statement in error. This statement is not processed, but an attempt is made to process the next directive statement. When the last directive statement is processed, the abort flag is checked; if set, the job is terminated. Otherwise, control is given to the next control statement.

REWIND (REWIND FILE)

REWIND generates a rewind of the named file. This file must not be one of the input or output file names given on the COPYN control statement nor the system INPUT file.

The format of the REWIND directive is:

REWIND,lfn.

lfn Name of file to be rewound, 1-7 letters or digits beginning with a letter.

SKIPF (SKIP FILE)

SKIPF skips forward or backward a designated number of partitions on a tape file. Requests for other types of files are ignored. No indication is given when SKIPF causes a tape to go beyond the double end-of-partition or when the tape is at load point.

The format of the SKIPF directive is:

SKIPF,lfn,n.

lfn	Name of tape file to be skipped, 1-7 letters or digits beginning with a letter.
n	Number (decimal) of file marks to be skipped. +n skips forward n marks; -n skips backward n marks.

SKIPR (SKIP RECORD)

SKIPR skips forward or backward a designated number of records. With the exception of zero-length records and tape marks which must be included, requests for other types of files are ignored.

The format of the SKIPR directive is:

SKIPR,lfn,n.

lfn	Name of tape file in which records are skipped, 1-7 letters or digits beginning with a letter.
n	Number (decimal) of records to be skipped. Zero-length records and file marks must be included in parameter n.

WEOF (WRITE FILE MARK)

WEOF writes a partition boundary on the named file. This file must be an output file named on the COPYN control statement.

The format of the WEOF directive is:

WEOF,lfn.

lfn	Logical file name of file, 1-7 letters or digits beginning with a letter.

RECORD IDENTIFICATION STATEMENT

The record identification statement contains the parameters which identify a system-logical-record or set of records to be copied from a given file.

The format of the record identification statement is:

p1,p2,p3.

p1 — First record to be copied or the beginning record of a set. Name associated with the record or a number giving the position in the file can be specified.

p2 — Last record to be copied in a set of records:

name	System-logical-records p1 through p2 are copied. p2 must be located between p1 and end-of-information.
decimal integer	Number of records to be copied, beginning with p1. Zero-length records and file marks are counted. Copying stops when the file end is encountered, even if the count has not been reached.
*	p1 through an end-of-partition are copied.
**	p1 through a double end-of-partition are copied.
/	p1 through a zero-length record are copied.
0 or blank	Only p1 is copied.

p3 — Input file to be searched. If p1 is a name, and p3 is omitted, all input files declared on the COPYN statement are searched until the p1 record is found. If it is not located, a message is issued. If p1 is a number and p3 is omitted, the last input file referenced is assumed. If this is the first directive statement, the first input file on the COPYN statement is used.

Examples of record identification statements:

SIN,TAN,INPUTA	Copies all system-logical-records from SIN through TAN from file INPUTA.
SIN,10,INPUTA	Copies 10 system-logical-records from file INPUTA, from SIN through SIN+9.
SIN,TAN	Searches all input files beginning with current file or first input file. When SIN is encountered, all system-logical-records are copied from SIN through TAN or until an end-of-partition is encountered.
SIN,,INPUTA	Copies system-logical-record SIN from file INPUTA.
1,TAN,INPUTA	Copies the current system-logical-record through TAN from INPUTA.
1,10,INPUTA	Copies 10 system-logical-records, beginning with the current system-logical-record on file INPUTA.
1,*,INPUTA	Copies the current system-logical-record through the first file mark encountered on INPUTA.

FILE POSITIONING FOR COPYN

Files manipulated during a COPYN operation are left in the position indicated by the previously executed directive. The file containing p1 is positioned at the record following p2. Other files remain effectively in the same position.

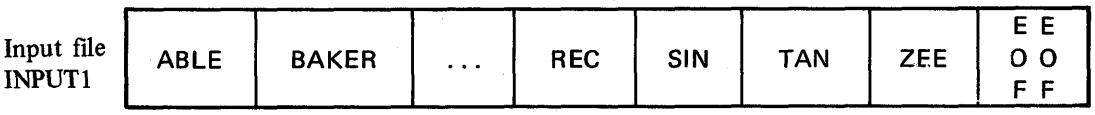
When COPYN is searching for a named record and p3 has been omitted, each input file is searched in turn until either the named record is found or the original position of the file is reached. The job INPUT file, however, is not searched end-around.

In contrast to the end-around search, a copy operation does not rewind files. An end-of-partition terminates a copy even if the record named in p2 has not been encountered. Since the output file is not repositioned after a search, COPYN can be re-entered. Therefore, the programmer is responsible for any REWIND, SKIP, or WEOF requests referencing the output file.

COPYN does not check for records duplicating names on other files. If such records exist, the programmer is responsible for them. COPYN uses the first record encountered that matches the name on a directive statement.

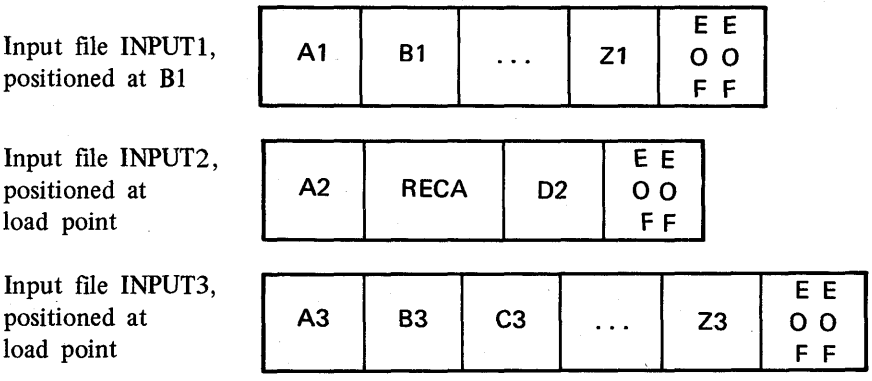
Examples of file positioning:

1. Record identification statement: REC,,INPUT1



If INPUTA were positioned at TAN, TAN and ZEE would be examined for REC. The double EOP would cause ABLE to be the next system-logical-record examined, continuing until REC is read and copied to the output file. INPUT1 would then be positioned at SIN.

2. Record identification statement: RECA



All records from B1 through A1 are searched to find RECA; this repositions INPUT1 to B1. A2 is searched, and when RECA is found, it is copied to the output file. INPUT2 remains positioned at D2. INPUT3 is not searched.

3. Record identification statements and binary records on INPUT file. Directive statements are:

```
REC,,INPUT
JOB1,JOB3,INPUT
ABLE,,IN2
7/8/9
REC (binary)
7/8/9
JOB1 (binary)
7/8/9
JOB2 (binary)
7/8/9
JOB3 (binary)
7/8/9
```

Because the INPUT file is not searched end-around, REC and JOB1 through JOB3 must directly follow the requesting record identification statements in the order specified by them. An incorrect request for an INPUT record terminates the job.

COPYSBF (COPY SHIFTED BINARY FILE)

COPYSBF adds a carriage control character to the beginning of each line during a copy to a second file. It is used with files to be printed when the existing first character is not a carriage control character. COPYSBF inserts a page eject character at the beginning of the first line. A blank is inserted at the beginning of subsequent lines to cause single spacing. A minimum field length of 10000 (octal) is required for COPYSBF.

A tape input file must be binary. Each line must be terminated by a 12-bit byte of zeros in the low order position of the last central memory word of the record.

The format of COPYSBF is:

```
COPYSBF,lfn1,lfn2.
```

Parameters are order dependent and optional.

lfn1	Name of input file to be copied onto lfn2, 1-7 letters or digits beginning with a letter. Default is INPUT.
lfn2	Name of output file onto which lfn1 is to be copied, 1-7 letters or digits beginning with a letter. Default is OUTPUT.

COPYXS (COPY X TAPE TO SI TAPE)

COPYXS converts a binary tape in X format to SI format. X tapes exist as a result of operating systems that are predecessors to NOS/BE 1. The binary X tape logical structure contains 512-word PRUs with short PRUs of sizes that are variable multiples of central memory words or 136 character PRUs.

The format of COPYXS is:

COPYXS,xlfn,scplfn,n.

Parameters xlfn and scplfn are required.

xlfn	Logical file name of input X tape, 1-7 letters or digits beginning with a letter.
scplfn	Logical file name of output SI tape, 1-7 letters or digits beginning with a letter.
n	Number (decimal) of partitions to be copied. Default is 1.

COPYXS is used in the following manner. Both files must be requested as S format.

```
REQUEST(xlfn,S)
REQUEST(scplfn,S)
COPYXS(xlfn,scplfn,n)
```

The output tape is produced in SI format, but is flagged in the system tables as S format. To read the output tape in the same job, the following control statements are needed:

```
UNLOAD(scplfn)
REQUEST(scplfn,MT)
```

COPYXS cannot determine when end-of-information occurs on an X tape; therefore, at least n partitions to be copied must exist on the X tape. Neither the input nor the output tape is rewound after conversion. After the requested number of partitions has been copied, the output tape is backspaced and positioned directly in front of the first tape mark preceding the EOF trailer label. Subsequent files can be copied to the output tape; however, the block count in the trailer label is then incorrect.

DELSET (DELETE MEMBER)

DELSET deletes devices from a device set. It cannot be executed while a device set is being shared. All member devices must be deleted before a DELSET is issued for the master device. The master device must be mounted before DELSET is issued. Permanent files, queue files, and local files residing on the device must be removed before DELSET is issued; if any portion of a local file or permanent file resides on the device to be deleted, the DELSET request is aborted.

The format of DELSET is:

DELSET,SN=setname,MP=vsn1,VSN=vsn2.

All parameters are required and are order independent.

SN=setname	Name of set from which member is to be deleted, 1-7 letters or digits beginning with a letter.
MP=vsn1	Volume serial number of master device for the device set, 1-6 letters or digits with leading zeros assumed.

VSN=vsn2 Volume serial number of member to be deleted from the device set, 1-6 letters or digits with leading zeros assumed.

DISPOSE (RELEASE FILE)

DISPOSE releases a file for end-of-job processing or specified disposition immediately or at the true end-of-job. DISPOSE can be used to:

Assign a disposition code for an output file, including a forms code

Send a file to a central site or remote site device

Evict a file.

The file referenced with DISPOSE must reside on a public queue device or on ECS and must not be a permanent file.

When a special-name file is to be evicted such that all file data and references are destroyed, the DISPOSE control statement should be used in preference to an UNLOAD or RETURN control statement. UNLOAD and RETURN cause the implicit disposition of the file to occur. Only DISPOSE or ROUTE can evict a file without causing special-name file output.

The format of DISPOSE is:

DISPOSE,lfn, { *dc / *dc=C / dc=Cff / dc=Iid }.

The only required parameter is lfn. The asterisk is optional before the dc parameter.

lfn Name of file to be disposed, 1-7 letters or digits beginning with a letter. If only lfn is specified, the file is evicted.

* Defer disposition until end-of-job. Must be used if DISPOSE control statement appears before the file is created. In the absence of *, disposition occurs when the DISPOSE control statement is encountered in the job stream. The * cannot be used when disposing a file to an INTERCOM terminal or to a forms code.

dc Disposition code. If dc is not specified, the file is evicted.

PR	Print on any available printer	P8	Punch free-form binary format
P2	Print on 512 printer	FR†	Print on microfilm recorder
LR	Print on 580-12 printer	PT†	Plot on any available plotter
LS	Print on 580-16 printer	HR†	Print on hardcopy device
LT	Print on 580-20 printer	HL†	Plot on hardcopy device
PB	Punch standard binary format	FL†	Plot on microfilm recorder
PU	Punch Hollerith format		

†Supporting drivers must be supplied by the installation.

C	File is to be routed to the central site
Cff	Code for special card or paper form ff. Codes are defined by the installation.
Iid	File is to be routed to the INTERCOM terminal specified by id.

Identification on the printout or punch output file is the name of the job that executed DISPOSE.

Examples of DISPOSE usage:

1.
```
JOB.
COBOL.
LGO.
DISPOSE,OUTPUT,PR.          Prints OUTPUT on any available printer
REWIND(LGO)
FTN.
LGO.
7/8/9
COBOL program               Creates print file on OUTPUT
7/8/9
data for COBOL program
7/8/9
FORTRAN program             Creates unrelated print file on OUTPUT
7/8/9
data for FORTRAN program
6/7/8/9
```

This example creates two unrelated print files. The use of DISPOSE allows the files to be printed separately. The job dayfile is attached to the second OUTPUT file.

2.
```
JOB.
DISPOSE,HERON,*PR=C.        File HERON to be printed at central site at end of job
COBOL.
LGO.
7/8/9
COBOL program               Creates file HERON and file OUTPUT
7/8/9
data for COBOL program
6/7/8/9
```

This job creates a file named HERON and prints it at central site. If this job is submitted from an INTERCOM terminal, the OUTPUT file and the dayfile are returned to that terminal.

DMP (DUMP CENTRAL MEMORY)

DMP prints the contents of selected areas of central memory. Four types of dumps are possible, depending on the relative values of the parameters on the DMP control statement:

Exchange package dump	Parameters omitted or all parameters specified are 0.
Control point area dump	Parameters equal in value and not 0.
Relative dump	Parameters specify address within field length.
Absolute dump	Parameters are six digits in length and begin with a 4, 5, 6, or 7. Absolute dumps might be inhibited at some installations.

DMP output appears on the file OUTPUT. Each output line contains the contents, in octal, of up to four central memory words, with the address of the first word at the beginning of the line.

When the content of a word is identical to the last word printed, printing of that word is suppressed. Printing resumes with the next word having a different content; the address of the word at which printing resumes is printed and marked by a right arrow.

When the content of a word is the address of that word, printing is suppressed. Printing resumes with the next word that does not have its address as its content; the address of the word at which printing resumes is printed and marked by a greater-than sign.

EXCHANGE PACKAGE DUMP

The format of DMP that produces an exchange package dump is:

DMP,0,0. or DMP.

Either or both of the parameters can be omitted.

Output from the dump includes:

The contents of the exchange jump package as noted below.

The contents of the communication area of the job field length, addresses RA through RA+100.

The contents of the first 100 octal words before and after the address to which the P register points, provided the addresses are with the field length. If the P register is 0, the P address in bits 30-47 of RA+0 determine the locations to be dumped. If the P register or the P address in RA+0 is less than 200 (octal), the first address dumped is 100. If both the P register and the P address are 0, only the communications area and the exchange package are dumped.

The 16-word exchange package includes the following information:

P	Program register contents
RA	Central memory address of beginning of user field length

FL	Central memory address of field length limit
EM	Error mode register divided by 100 (octal)
RE	ECS reference address divided by 1000 (octal)
FE	ECS field length divided by 1000 (octal)
MA	Monitor address applicable only to machines with monitor exchange jump instructions
A0-A7	Contents of A registers 0-7
B1-B7	Contents of B registers 1-7 (B0 is always zero)
X0-X7	Contents of X registers 0-7

When the exchange jump package is dumped, the following information is given also if addresses are within the field length. A message **OUT OF RANGE** appears if they are outside the field length.

C(A1)-C(A7)	Contents of addresses listed in registers A1-A7
C(B1)-C(B7)	Contents of addresses listed in registers B1-B7

CONTROL POINT AREA DUMP

The format of DMP that produces a control point area dump is:

DMP,x,x.

x	Any octal value except 0 can be specified.

This control statement dumps the entire (200 octal word) control point area.

RELATIVE DUMP

The format of DMP that produces a relative dump of locations with the job field length is:

DMP,from,thru.

When only one parameter appears, it is presumed to be the thru parameter and dump begins at RA.

from	Address at which dump is to begin after RA, octal.
thru	Address at which dump is to end, octal. If address exceeds FL, FL is substituted.

ABSOLUTE DUMP

The format of DMP that produces a dump of absolute address in memory is:

DMP,from,thru.

When only one parameter appears, it is presumed to be the thru parameter and dump begins as if 400000 (octal) were specified.

from — Absolute address to be dumped, expressed as six octal digits address+n00000 where n is 4 through 7 as noted below. If the value exceeds memory size, no dump occurs.

thru — Absolute address at which dump is to end, expressed as six octal digits address+n00000 where n is 4 through 7 as noted below. If value exceeds the size of memory, dump stops at the end of memory.

If the value specified is less than 400000, it is treated as address+400000.

Absolute addresses are expressed in six octal digits. The first digit must be 4 through 7, which sets the upper bit in the address to indicate an absolute dump. DMP subtracts 400000 from the addresses specified to determine the absolute address to be dumped. The maximum address that can be dumped is:

thru Parameter	Maximum CM Address Dumped
4xxxxx	77 777 (32K)
5xxxxx	177 777 (65K)
6xxxxx	277 777 (98K)
7xxxxx	377 777 (131K)

Only the first 131K words of memory can be dumped with DMP. See the ABS control statement if more than 131K of memory is to be dumped. DMP can also be called using the SYSTEM macro described in section 6.

DMPECS (DUMP EXTENDED CORE STORAGE)

DMPECS prints the contents of selected areas of Extended Core Storage. The file on which information appears and the format of the dump are both selected by control statement parameters. Only the field length assigned to the job can be dumped. All addresses are between RE and FE, the reference address and field length of assigned ECS.

The format of DMPECS is:

DMPECS,from,thru,format,lfn.

Parameters are positional; from and thru are required.

from — Address (octal) at which dump is to begin after RE.

thru — Address (octal) at which dump is to end. If address exceeds FE, FE is substituted.

format — Format of each output line

0 or 1	4 words in octal and in display code; default
2	2 words in 5 octal digit groups and in display code
3	2 words in 4 octal digit groups and in display code
4	2 words in octal and in display code

lfn — Name of file on which printout is to appear, 1-7 letters or digits beginning with a letter. If omitted or 0, OUTPUT is assumed.

The dump begins at the closest multiple of 10 (octal) less than or equal to the value of the from parameter; the dump ends at the closest multiple of 10 (octal) greater than the value of the thru parameter minus 1.

DSMOUNT (DISASSOCIATE DEVICE)

DSMOUNT disassociates a private device from the job. DSMOUNT is a logical operation. When DSMOUNT specifies the master device of a private device set, the entire set is disassociated from the job. A CLOSE/UNLOAD function is issued for each open file on the set before each mounted member device is dismounted. Finally, the master device is logically dismounted from the job.

The format of DSMOUNT is:

```
DSMOUNT,VSN=vsn,SN=setname.
```

Both parameters are required and order independent.

VSN=vsn — Volume serial number of device to be dismounted, 1-6 letters or digits with leading zeros assumed. Can be a member device or a master device.

SN=setname — Name of device set to which this device belongs, 1-7 letters or digits beginning with a letter.

DUMPF (DUMP PERMANENT FILE TO TAPE)

DUMPF dumps permanent files to a tape. It can be used to clear permanent files from a mass storage device or to maintain back-up copies of files selected by parameters on the DUMPF control statement. Parameters on the DUMPF can identify a single file by name or specify the criteria by which the permanent file system selects files for dumping.

The dump tape must be S tape format with the logical file name DUMTAPE. A REQUEST statement must appear in the job before DUMPF is called.

Three dumps are possible:

Mode 1 — Back-up dump. The original copy of the file remains on mass storage ready for immediate access by an executing job.

Mode 2 — Archive dump. The file remains a permanent file, but with archive status. The only copy of the file resides on the dump tape; it can be accessed by an executing job if the operator makes the archive tape available so that the file can be reloaded to mass storage.

Mode 3 — Destructive dump. The file is no longer a permanent file. The only copy of the file resides on the dump tape; it cannot be accessed unless the LOADPF utility is executed to restore the file to permanent file status.

DUMPF execution causes an implicit attach of a file having the permanent file name DUM. The device set from which files are being dumped must contain a copy of DUM cataloged with an ID of PUBLIC, a TK password of DUMPF, and installation defined passwords for RD, MD, and CN. Passwords to access DUM must be submitted as part of the DUMPF call.

For each cycle dumped, DUMPF makes an output listing entry that contains the permanent file name, owner ID, cycle number, volume serial number of the dump tape, date of dump, a comment, and the flagging of any parity errors.

The format of DUMPF is:

DUMPF,PW=pw,MO=n, {I / I=lfn1} ,LF=lfn2,CL,DP=a,ID=name,PF=pfn,CY=cy,SN=sn,VSN=vsn,IN=ddd,JN=yyddd,
LA=mmddyy,DA=yyddd,CD=mmddyy,TI=hhmm.

Only PW is required; all other parameters are optional and order independent. Only one CD, DA, JN, LA, or IN parameter can appear. If a terminator does not appear at the end of the parameter list, column 1 of the next card or line is considered to be a continuation of the DUMPF parameter list.

PW=pw — RD, MD, or CN password for DUM, depending on mode of dump. See CATALOG control statement for password definitions.

MO=n — Dump mode:

- 1 — Back-up mode. Permanent file tables and all associated mass storage space are intact. RD password required. Default.
- 2 — Archive dump. Mass storage space is released, but permanent file tables remain with the files marked as being on an archive tape. MO password required.
- 3 — Destructive dump. All permanent file tables and mass storage spaces are released as the files are dumped. CN password required. The central site operator receives notification when a mode 3 dump is attempted and must authorize continuance of the dump.

I=lfn1 — Logical file name of directive file for MO=1 dump; 1-7 letters or digits beginning with a letter. All other parameters except MO, SN, CL, and PW are ignored.

I — Directives for MO=1 are on INPUT.

LF=lfn2 — Output listing file. Default is OUTPUT.

CL — Complete list option selected. All files in the permanent file directory are listed. If CL is omitted, information is listed only for files which are dumped.

DP=a — Dump type:

- A — All files meeting criteria of other parameters. Default.
- X — All files meeting criteria of other parameters only if their expiration dates are equal or less than current date.
- C — All files meeting criteria of other parameters only if they have been modified, renamed, created, or extended since the last DP=C dump.

ID=name — Dump files with this owner.

PF=pfn — Dump files with this permanent file name. ID is also required.

CY=cy — Dump cycle cy of file identified by PF and ID. CY is ignored and the dump continues if this cycle is not found or if PF and ID have not also been specified.

SN=sn — Dump files from device set with this name; 1-7 letters or digits beginning with a letter.

VSN=vsn — Dump files from this device of device set specified by SN; 1-6 letters or digits with leading zeros assumed. VSN is ignored if SN is omitted.

IN=ddd — Dump files inactive this number of days; 1-3 digits. Can be qualified by a TI parameter.

JN=yyddd — Dump files inactive since this ordinal date; 5 digit ordinal date format. Can be qualified by TI parameter.

LA=mmddyy — Dump files not attached on or after this date; 6 digit month-day-year format. Can be qualified by TI parameter.

DA=yyddd — Dump files created, modified, renamed, or extended after this date; 5 digit year-and-day-of-year format. Can be qualified by TI parameter.

CD=mmddyy — Dump files created, modified, renamed, or extended after this date; 6 digit month-day-year format. Can be qualified by TI parameter.

TI=hhmm — Time qualifier for date parameters; 4 digit 24 hour clock format. If date parameters are not specified, TI is ignored.

Several copies of DUMPF can execute at the same time on the same set as long as all copies running have identical parameters. If an attempt is made to run a DUMPF with different parameters than one already running, all except the first DUMPF aborts.

If a group of files is to be dumped for back-up purposes, they can be identified by name and owner in a directive record. The I parameter is required to specify the name of the file containing directives. Directive format is as follows. Parameters are order independent and ending punctuation is not required. The CY and ID parameters are optional.

```
PF=pfn,CY=cy,ID=name
```

EDITLIB (CONSTRUCT USER LIBRARY)

EDITLIB constructs user libraries from a group of central processor routines or overlays. That library is available to the system loader by specific direction in the loader control statements for a job. It can also create and maintain system libraries and create deadstart tapes. With EDITLIB a user library can be modified by the addition, deletion, or replacement of routines; and statistics about library contents can be listed.

The user library must contain assembled central processor routines, programs, or text records produced by the COMPASS assembler, one of the system compilers, or loader generated overlays. Library records can be independent programs, subroutines, or overlays. Binary output from SEGLOAD cannot be made part of a library. Unassembled text records in BCD format, peripheral processor programs, and source language programs cannot be made part of user libraries.

EDITLIB considers each program on the user library to be a single unit occupying a system-logical-record. It extracts the name, entry points, and external references from tables output with the program assembly and uses them to construct tables describing the library file. Library tables are used by the loader to locate programs on the file. EDITLIB changes the tables when the user library is modified. Format of user library tables is the same as that for system libraries. A user library file created by EDITLIB contains:

Assembled programs

Tables referring to
- Entry points
- External references
- Program numbers
- Program names

The program number table is used to link external references, entry points, and program names.

A user library can contain at most 2047 programs, 2047 external references, and 2047 entry points. A particular program in the library can have at most 124 entry points and 124 external references.

The user library file generated by EDITLIB can be on mass storage or magnetic tape. If the library file name is assigned to a tape file before EDITLIB is called, the library is in sequential format on that tape, with the library tables preceding the programs. Otherwise, the library is in random format on mass storage. When the random library file is to be retained as a permanent file, the library file name should be associated with a permanent file device before EDITLIB is called.

If a user library is to be copied from mass storage to tape, the EDITLIB directive RANTOSEQ should be used rather than a COPY utility. Likewise, SEQTORAN should be used to copy a library from tape to disk. The COPY utilities cannot copy a library file from mass storage correctly.

The user is responsible for cataloging and attaching any permanent files that are used by EDITLIB while performing the task specified on each directive, and for extending permanent files that have been changed.

EDITLIB CONTROL STATEMENT FORMAT

The EDITLIB utility is called by an EDITLIB statement in the control statement section. If encountered during job processing, EDITLIB accesses the next unprocessed section of the INPUT file; unless the I parameter names another source of directives. A parameter on this statement specifies the file that contains EDITLIB directives. These directives provide details for creating or manipulating the user library.

The format of EDITLIB is:

EDITLIB(USER,I=lfndir,L=lfnlist)

All parameters are optional.

USER	Distinguishes user library definition from system library. Default is USER.
lfndir	Logical file name containing directives, 1-7 letters or digits beginning with a letter. Default is INPUT. I is identical to I=INPUT.
lfnlist	Logical file name to receive listable output, 1-7 letters or digits beginning with a letter. Default is OUTPUT. L is identical to L=OUTPUT.

The following deck structure assembles two programs and adds them to an existing library:

```
job statement
COMPASS.
FTN.
ATTACH(ALIB,ID=SMITH)
EDITLIB(USER)
EXTEND(ALIB)
7/8/9
   COMPASS program to be assembled
7/8/9
   FORTRAN Extended program to be compiled
7/8/9
   Directives instructing EDITLIB to add programs to user library ALIB from LGO file
6/7/8/9
```

EDITLIB DIRECTIVE FORMAT

The directive section for EDITLIB must contain only valid directives. EDITLIB considers the first 72 columns of each 80 column card or 90 column card image to contain a separate directive. Blanks can be used freely; EDITLIB removes them except in a literal or comment field. Required format for directives is similar to system control statement format.

The format of EDITLIB directives is:

keyword. or keyword(parameter list)

Parentheses are required around parameter lists. Optional parameters have the format parameter=value; all others are required. Required parameters must appear in the order given; optional parameters can appear in any order after the required parameters.

Directive format and use is summarized below:

LIBRARY(libname, $\left\{\begin{matrix}\text{OLD}\\\text{NEW}\end{matrix}\right\}$) Defines library to be created or modified

FINISH.	Terminates library manipulation
ENDRUN.	Stops execution of directives
ADD(prog,from,AL=level,FL=fl,FLO=0,LIB)	Adds new program to library
REPLACE(prog,from,AL=level,FL=fl,FLO=0,LIB)	Replaces program on library
DELETE(prog)	Deletes program in library
SETAL(prog,level)	Changes access level
SETFL(prog,fl)	Changes field length requirements
SETFLO(prog, {0 / 1})	Sets FL override bit for INTERCOM
LISTLIB(prog,lfn)	Lists program data from library file
REWIND(lfn)	Rewinds file
CONTENT(prog,lfn)	Lists program data from file
SKIPF({n / prog},lfn)	Skips ahead n records or to prog
SKIPF(n,lfn,F)	Skips n files forward
SKIPB({n / prog},lfn)	Skips back n records or to prog start
SKIPB(n,lfn,F)	Skip n files backward
*/	Inserts comments in output
RANTOSEQ(rlfn,slfn)	Rewrites random library as sequential library
SEQTORAN(slfn,rlfn)	Rewrites sequential library as random library

The prog parameter in these directives can take several forms:

A single program name can be stated. EDITLIB searches the entire file specified to find the named program.

An asterisk can replace the program name. EDITLIB processes all programs from the current file position to end-of-file.

A range of programs to be included in directive execution can be specified with a + between the first and last programs to be processed. In a file with records A,B,C,D,E, the range B+D represents B,C,D.

A range of programs to be excluded from directive execution can be specified with a - between the first and last programs to be considered. In a file with records A,B,C,D,E, the range B-D represents A and E.

An asterisk can replace either the first or last program named in a range. For the first named program, it is equated with the current file position; for the last, it is equivalent to end-of-partition.

For the ADD, REPLACE, and REWIND directives only, several individual programs can be stated. In a file with records A,B,C,D,E, the parameter D/B/E represents D and B and E. EDITLIB searches the entire file specified to find the named program.

A single program to be excluded from directive execution can be specified with a dash (–) preceding the program name or with the program name appearing at both ends of the range of programs to be excluded.

Program names must not exceed 7 characters. Any character supported by the system is legal. If characters EDITLIB uses for delimiters are in a name, the entire name must be written as a literal between dollar signs. These characters are:

$ ( ) - + = . , / blank

Any dollar sign to be included in the program name must be prefixed by a second dollar sign.

If the prog parameter is a single program name, EDITLIB searches the entire file for that program. If the prog parameter is a range, EDITLIB searches the entire file for the first program in the range, but does not search end-around for the second program. Thus, a range goes from the first program through either the second program or end-of-partition whichever occurs first. The file INPUT is not searched.

The interpretation of the * depends on file format. The current position of a library file is always defined to be the beginning of the file. Current position of other files is simply the beginning of the next record on the file, which can be controlled by the user with file manipulation directives. An * replacing the last program is equivalent to stating end-of-partition.

Examples of names acceptable to EDITLIB:

Parameter Format	Resulting Program Name
PROG12	PROG12
$PROG12$$$	PROG12$
$I-O$	I-O
AA BB	AABB
$AA BB$	AA BB
3AB	3AB

Library file names should not begin with ZZZZZ since these are reserved for system names.

MANIPULATION OF LIBRARY FILES

A library is created by identifying the library in a LIBRARY directive followed by file manipulation statements and ending with the FINISH directive. Multiple LIBRARY/FINISH sequences are permitted within an EDITLIB directive set. An ENDRUN should follow the last FINISH in the EDITLIB directive set. If ENDRUN is not supplied by the user, EDITLIB inserts it.

Existing user libraries in random file format are modified by the ADD, REPLACE, and DELETE directives that change programs in the library. The SETAL, SETFL, and SETFLO directives change parameters in the program name table of entries for existing libraries. These directives must be issued between the LIBRARY (lfn,OLD) and FINISH directives.

The format of library files can be changed by the RANTOSEQ function and the SEQTORAN function.

File positioning statements can appear anywhere in the directive record. EDITLIB rewinds all files except INPUT before executing any directives. After a random library is written, it is rewound. When a new sequential library is written, it is left-positioned after the end-of-partition.

A list of information about any or all programs on a library file or a file of assembled information is obtained by the LISTLIB and CONTENT directives. Information listed comes from the program tables output with every assembled record. It includes:

Program name

Date, time, and compilation or assembly machine

Entry points

External references

AL and FL values

Length of object deck in central memory words

Type of program: relocatable or absolute

ADD (ADD PROGRAM DURING LIBRARY CREATION)

ADD directives between LIBRARY(lfn,NEW) and FINISH directives create a user library. Programs (other than peripheral processor programs) can be added from any file attached to the job, as long as the program contains the necessary prefix table material at the beginning of the assembled information. If the directive is in error, a message is issued, the programs are not added, and processing continues.

The format of ADD is:

$$\mathrm{ADD}\left(\mathrm{prog,from,AL{=}level,FL{=}fl,FLO{=}}\left\{\begin{matrix}0\\1\end{matrix}\right\}\mathrm{,LIB}\right)$$

Parameters prog and from are required; all others are optional.

prog — Name of program or range of programs to be added.

from — Logical file name where assembled program currently resides, 1-7 letters or digits beginning with a letter.

AL=level — Access level of 1-4 (octal) digits used to determine whether or not a given INTERCOM user can attach and use the program named. Also used to mark programs for access by control statements; level must be an odd number. Program is available only to internal calls unless AL is odd. Default is 0.

FL=fl — Maximum field length (0 to 377777 (octal)) required for program loading and execution. If FL=0, the field length specified on the job statement or the last RFL statement encountered is used. Default is 0.

FLO=$\left\{\begin{matrix}0\\1\end{matrix}\right\}$ — Field length override bit. If FLO=1, then the field length from the job control statement CM parameter or from the RFL control statement or from the EFL INTERCOM command, overrides FL. If FLO=0, no override is allowed. Default is 0.

LIB — Indicates the parameter from is a user library name. Allows programs to be added from an existing user library. It directs EDITLIB to search the directory of a file in library format.

If AL, FL, or FLO values are wanted in the new library tables, they must be explicitly stated in the directive, even if the addition is to be made from an existing library. To change the values of these parameters in an existing library, use the SETAL, SETFL, and SETFLO directives.

Examples of valid ADD formats and their results:

Parameter Format	Result
ADD(*,TREES)	All programs between current position and the end-of-file TREES is added.
ADD(RAINIER,MTS,FL=14400)	All of file MTS is searched for program RAINIER; field length of 14400 (octal) is required to execute RAINIER.
ADD(REDWOOD–SEQUOIA,TIMBER)	All programs on file TIMBER, except REDWOOD, SEQUOIA, and all those between, are added.
ADD(*+ASPEN,YELLOW)	All programs from the current position of YELLOW through program ASPEN are added.
ADD(BIG/SHARP,LEAF)	File LEAF is searched as needed, and programs BIG and SHARP are added.
ADD(ALP,LIBR,LIB)	The program name table of library LIBR is searched for program ALP which, when located, is added to the current library.

CONTENT (LIST FILE)

CONTENT lists any file of assembled programs, whether in library format or not.

The format of CONTENT is:

CONTENT(prog,lfn)

prog	Program or range of programs to be listed.
lfn	Logical file name containing prog, 1-7 letters or digits beginning with a letter.

DELETE (DELETE PROGRAM FROM LIBRARY)

DELETE logically deletes all references to the named program from library tables.

The format of DELETE is:

DELETE(prog)

prog	Name of program or range of programs to be deleted.

Examples of valid DELETE formats and their results:

Parameter Format	Result
DELETE(BIRCH+ASH)	Programs BIRCH through ASH on library being modified are deleted.
DELETE(LAUREL–MADRONE)	All programs on existing library except LAUREL, MADRONE, and those between, are deleted.

Programs named in a DELETE or REPLACE directive are logically deleted from the library file. Records in the file are not overwritten; but in the case of a REPLACE, the file is extended with the addition of a new program. To completely eliminate programs from the library, it is necessary to construct a new library using the old one as the source.

ENDRUN (STOP EXECUTION)

During directive processing, EDITLIB first interprets each directive in the record excluding comment statements. Execution begins after all directives are interpreted.

When an ENDRUN is encountered during execution phase, execution stops. In most instances, ENDRUN is the last directive in the record. By placing it earlier in the record, syntax of succeeding directives can be checked without an error producing premature termination.

The format of ENDRUN is:

ENDRUN.

FINISH (STOP FILE MANIPULATION)

FINISH indicates the end of library construction.

The format of FINISH is:

FINISH.

LIBRARY (DELIMIT LIBRARY)

LIBRARY identifies the library to be manipulated. This directive must precede all other directives except comments or file manipulation directives. Every directive set calling for library creation or modification must have at least one such directive. A FINISH directive is required to mark the end of library construction. File manipulation statements can appear between LIBRARY and FINISH.

The format of LIBRARY is:

LIBRARY(libname,{OLD / NEW})

libname	Library name and name of file containing library during this job.
OLD	Used when existing library to be modified is referenced by libname.
NEW	Used when libname refers to new library or directory to be created.

LISTLIB (LIST LIBRARY FILE)

LISTLIB lists a library file. Part or all of the library can be listed depending on the number of programs indicated by the prog parameter. The LISTLIB directive cannot appear between a LIBRARY and a FINISH.

The format of LISTLIB is:

LISTLIB(prog,lfn)

prog	Program or range of programs to be listed.
lfn	Logical file name containing prog, 1-7 letters or digits beginning with a letter.

RANTOSEQ (CONVERT RANDOM FILE TO SEQUENTIAL FILE)

RANTOSEQ takes a disk resident library file in random format and creates a sequential library file containing the same programs. This directive cannot appear between a LIBRARY and FINISH.

The format of RANTOSEQ is:

RANTOSEQ(rlfn,slfn)

rlfn Disk resident random library that is to be converted.

slfn Sequential library created from rlfn.

REPLACE (DELETE AND REPLACE PROGRAM)

REPLACE differs from the ADD directive in that it causes a program with an identical name to be deleted from the library before the new program is added. If a program with that name does not exist, an informative message is issued and the new program is added to the library.

The format of REPLACE is:

REPLACE(prog,from,AL=level,FL=fl,FLO=0,LIB)

Parameters have the same meaning as those of the ADD directive. AL, FL, and FLO values must be stated explicitly if values other than the defaults are wanted. Current values in source library or existing library tables are not preserved when ADD or REPLACE is used. See ADD for parameter definitions.

Examples of valid REPLACE formats and their results:

Parameter Format	Result
REPLACE(MAPLE,TREES,FLO=0)	Existing program MAPLE is deleted; program MAPLE is added from file TREES: FLO is set to 1; FL and AL are set to default values.
REPLACE(OAK,TREES)	Existing program OAK is deleted and replaced; FL, FLO, and AL receive default values.
REPLACE(ACORN,TREE,LIB)	Program name table for library TREE is searched for program ACORN. The named program is deleted from the current library and the new program ACORN is added from library TREE.

REWIND (REWIND FILE)

The format of REWIND is:

REWIND(lfn) or REWIND(lfn/lfn/ . . . lfn)

lfn Logical file name of file or files to be rewound.

SEQTORAN (CONVERT SEQUENTIAL FILE TO RANDOM FILE)

SEQTORAN takes a tape resident library file in sequential format and creates a disk resident library file containing the same programs. The directive cannot appear between a LIBRARY and a FINISH.

The format of SEQTORAN is:

SEQTORAN(slfn,rlfn)

slfn	Tape file in sequential format that is to be converted.
rlfn	Random library file created from slfn.

SETAL (CHANGE ACCESS LEVEL)

SETAL assigns a new access level to the named program.

The format of SETAL is:

SETAL(prog,level)

prog	Name of program or range of programs.
level	New access level of 1-4 (octal) digits.

SETFL (CHANGE FIELD LENGTH)

SETFL assigns a new field length to the named program.

The format of SETFL is:

SETFL(prog,fl)

prog	Name of program or range of programs.
fl	New field length of 0 to 377777 (octal).

SETFLO (SET FIELD LENGTH OVERRIDE BIT)

SETFLO sets the field length override bit for INTERCOM.

The format of SETFLO is:

SETFLO(prog, $\left\{\begin{matrix}0\\1\end{matrix}\right\}$)

prog	Name of program or range of programs.
0 1	New field length override parameter. 1 allows override. 0 is the default value and does not allow override.

SKIPB (SKIP BACKWARD)

SKIPB repositions a library backward one or more records or files. The library is positioned at the beginning of a record or file. When beginning-of-information or end-of-information is encountered, a skip by count is terminated. For a skip by name, the entire file is searched, if necessary, in the direction stated. Skip by program name is applicable to sequential files only.

The format of SKIPB for records is:

SKIPB($\left\{\begin{matrix} n \\ prog \end{matrix}\right\}$,lfn)

n	Number (decimal) of records to be skipped backward; cannot be zero.
prog	Program name to which instruction skips.
lfn	Logical file name containing prog, 1-7 letters or digits beginning with a letter.

The format of SKIPB for files is:

SKIPB(n,lfn,F)

n	Number (decimal) of files to be skipped backward; cannot be zero.
lfn	Logical file name of multi-file, 1-7 letters or digits beginning with a letter.
F	Indicates files are to be skipped, not records.

SKIPF (SKIP FORWARD)

SKIPF repositions a library forward one or more records or files. The library is positioned at the beginning of a record or file. When beginning-of-information or end-of-information is encountered, a skip by count is terminated. For a skip by name, the entire file is searched, if necessary, in the direction stated. Skip by program name is applicable to sequential files only.

The format of SKIPF for records is:

SKIPF($\left\{\begin{matrix} n \\ prog \end{matrix}\right\}$,lfn)

n	Number (decimal) of records to be skipped forward; cannot be zero.
prog	Program name to which instruction skips.
lfn	Logical file name containing prog, 1-7 letters or digits beginning with a letter.

The format of SKIPF for files is:

SKIPF(n,lfn,F)

n	Number (decimal) of files to be skipped forward; cannot be zero.

lfn Logical file name of multi-file, 1-7 letters or digits beginning with a letter.

F Indicates files are skipped, not records.

USER EDITLIB EXAMPLES

1. MTCREAT.
   REQUEST(MTLIB,LO,VSN=14444) Requests 7-track tape to hold new library.
   REQUEST(SORCEFL,MT,VSN=14445) Requests tape with previously assembled source programs.
   FTN.
   EDITLIB(USER)
   7/8/9
   FORTRAN Extended program to be compiled, program name HOOD.
   7/8/9
   LIBRARY(MTLIB,NEW) Initiates construction of new library MTLIB.
   REWIND(SORCEFL) Rewinds binary input file.
   REWIND(LGO) Rewind binary output from FORTRAN Extended program.
   ADD(*+SHASTA,SORCEFL) Adds programs from beginning of file through SHASTA.
   SKIPF(3,SORCEFL) Skips 3 programs on file.
   ADD(HOOD,LGO) Adds program from LGO file.
   ADD(*,SORCEFL) Adds all remaining programs on SORCEFL.
   FINISH. Terminates library construction.
   ENDRUN. Stops execution.
   6/7/8/9

   Job MTCREAT creates a sequential user library on a tape.

2. MTCHNGE.
   REQUEST(MTLIB,LO,VSN=14444)
   REQUEST(DIRECT,MNT,VSN=12000)
   EDITLIB(I=DIRECT)
   6/7/8/9

   Job MTCHNGE modifies the library created above. Directives for EDITLIB are on tape 12000.

3. BIRDS. Job statement.
   REQUEST(BIRDLIB,*PF) Requests permanent file device for library.
   ATTACH(GULLS,GULLSPF,ID=PETERSON) Attaches permanent file as lfn GULLS.
   ATTACH(WRENS,WRENSPF,ID=PETERSON) Attaches permanent file as lfn WRENS.
   EDITLIB(USER) Calls EDITLIB.
   CATALOG(BIRDLIB,BIRDLIBRARY,ID=PETERSON) Catalogs library as permanent file.
   7/8/9
   LIBRARY(BIRDLIB,NEW) Establishes library name.
   ADD(*,GULLS) Adds all files from GULLS.
   ADD(CACTUS–HOUSE,WRENS) Adds all files from WRENS except CACTUS through HOUSE.
   FINISH. Terminates library.
   ENDRUN. Stops execution.
   6/7/8/9

Job BIRDS creates a random format library file and makes it permanent. Binary input files exist on permanent files GULLSPF and WRENSPF.

4.
```
CHECK.
EDITLIB(USER)
7/8/9
ENDRUN.                          Stops execution here.
LIBRARY(OLDLIB,OLD)
DELETE(SPARROW)
REPLACE(HAWK,INPUT,FLO=0)
SETAL(SHRIKE,777)
SETFLO(ROBIN,1)
SETFL(CREEPER,55000)
.
.
.
FINISH.
6/7/8/9
```

Job CHECK uses EDITLIB to check syntax of all directives, but does not execute.

EXECUTE (INITIATE EXECUTION)

EXECUTE causes execution of a loaded program. It is a loader control statement. See the LOADER reference manual for additional information. EXECUTE terminates a load sequence.

The format of EXECUTE is:

```
EXECUTE.
```

EXECUTE normally follows a LOAD control statement.

EXIT (PROCESS AFTER FATAL ERROR)

EXIT establishes processing to occur if a fatal error is terminating the job. In the absence of an EXIT control statement, job termination occurs as described in section 2. When EXIT is present, processing might resume, depending on the type of error and the parameter on the EXIT control statement.

Certain conditions always cause abrupt termination of a job:

Request from the operating system or computer operator to terminate job and inhibit all output (KILL command)

Request from operator to transfer job from central memory back into input queue (RERUN command)

Error on job statement

Checksum error during job input

When other types of otherwise fatal errors occur, the operating system searches the control statements for EXIT. The following terminating conditions result in this search.

Job uses all execution time allotted

Arithmetic error unless negated by a MODE control statement

Peripheral processor encounters improper input/output request

Central processor program requests job termination

Operator request to drop job (DROP command)

ECS parity error occurs

Control statement error, other than on job statement

With the exception of control statement format errors, the above conditions can be reprieved within COMPASS or FORTRAN Extended programs, as indicated by the RECOVR macro.

The format of EXIT is:

$$\text{EXIT,}\left\{\begin{matrix}\text{C}\\\text{U}\\\text{S}\end{matrix}\right\}.$$

All parameters are optional.

omitted	Execute the following control statements when a non-special fatal error occurs.
C	Execute the following control statements unless a non-special error has occurred.
U	Execute the following control statements if a special fatal error has not occurred.
S	Execute the following control statements when either a special or non-special error occurs.

More than one EXIT control statement can appear in the job stream.

System action after an error occurs depends on the next EXIT control statement in the job stream, if any, as noted in the chart below. The chart also shows system action when an EXIT control statement is encountered during normal job step advancement.

	Next EXIT Control Statement				
Error Condition	**EXIT.**	**EXIT,C.**	**EXIT,U.**	**EXIT,S.**	**No EXIT**
Special error occurs (see list below)	Skip to EXIT,S or end job	Skip to EXIT,S or end job	Skip to EXIT,S or end job	Resume processing after EXIT,S.	End job
Other error occurs	Resume processing after EXIT	End job	Resume processing after EXIT,U.	Resume processing after EXIT,S.	End job
No error, but EXIT encountered	End job	Resume processing after EXIT,C.	Resume processing after EXIT,U.	End job	End job

The special error conditions are:

ABORT macro within central processor program that specifies NODUMP or S.

Control statement format errors.

Attempt to load an object program which resulted from containing assembly or compilation errors.

If necessary, the system increases the CP time limit, IO time limit or mass storage limit to provide an installation defined minimum of time and mass storage for EXIT processing. No limit is increased more than once in a job.

EXTEND (PERMANENT FILE EXTENSION)

EXTEND makes permanent information written at the end of an existing permanent file. Information can be written at the end of any attached permanent file; in the absence of an EXTEND or ALTER control statement, however, the added information disappears when the job terminates. EXTEND can be issued with the file at any position.

EXTEND can be issued by any job that attaches the file with extend permission or by the job that catalogs the file. The newly added information acquires the privacy controls of the existing permanent file. No boundary exists between the original information and the new information.

The format of EXTEND is:

EXTEND,lfn.

lfn — Logical file name of permanent file attached with extend permission, 1-7 letters or digits beginning with a letter.

GETPF (ATTACH PERMANENT FILE FROM LINKED MAINFRAME)

GETPF attaches a permanent file to a job, as long as parameters specified on the GETPF control statement establish the right to use the file. GETPF differs from the ATTACH control statement in that:

GETPF creates a local copy of a file; ATTACH manipulates the file itself.

GETPF can obtain a copy of any permanent file residing in a permanent file default set. ATTACH can access only permanent files which reside on a device directly connected to the mainframe on which the job is executing.

The format of GETPF is:

GETPF,lfn,pfn,ID=name,AC=act,EC=ec,{LC=n / CY=cy},MR=m,PW=pw,RW=p,ST=mmf.

The first parameter establishes the logical file name. Parameters lfn and pfn are required in the order shown; all other parameters are order independent. ID and ST are required. GETPF can be continued: if a period or right parenthesis does not appear at the end of the parameter list, column 1 of the next statement is considered a continuation of column 80.

lfn	Logical file name, 1-7 letters or digits beginning with a letter. If omitted, the first seven characters of pfn establish lfn.
pfn	Permanent file name by which the file is known in the permanent file catalog, 1-40 letters or digits. Required.
ID=name	ID parameter by which the file was cataloged. Required unless the file was cataloged with ID=PUBLIC.
ST=mmf	System on which file is cataloged, 3 characters.

See the ATTACH control statement for the remaining parameters.

GETPF always sets MR=1.

The file referenced by a GETPF must reside on the permanent file default set of the mainframe specified. A copy of the file is transmitted to the mainframe on which the job is executing at the time the file is opened.

Any modifications made to the file during the job are a part of the local file copy, not of the original permanent file.

LABEL (TAPE LABEL SPECIFICATION)

LABEL writes or checks VOL1 and HDR1 labels on tapes. In addition to substituting for a REQUEST control statement for a single file labeled tape, LABEL can be used to position within a multi-file set.

In most instances, LABEL is the first reference to a file in a job, unless it is preceded by a VSN statement indicating the volume serial number of the resident volume. For a single file volume, a REQUEST is not needed, although a REQUEST followed by LABEL is valid and does not create an error condition. If a

REQUEST statement follows the LABEL statement, duplicate file names are generated; and the job terminates since the LABEL program issues a REQUEST function to obtain the equipment. For labeled multi-file volumes, a REQUEST establishing the multi-file set must precede the LABEL statements that write the header labels for various files in the set.

The label program issues an OPEN function to read or write the file label. Contents of the label are copied to both the system and job dayfiles. When label fields are not consistent with the information supplied on the LABEL control statement, the operator is notified; the operator then can mount another tape and have its label checked or can authorize the job to continue with the existing tape.

The format of LABEL is:

LABEL,lfn,{W / R},{Z / Y},{RING / NORING}, IB,D=d,F=f,N=n,X=x,L=z,V=v,E=e,T=t,C=c,M=m,P=p,VSN=vsn.

The first parameter must be the logical file name. An R or W parameter is required. The remaining optional parameters are order independent. LABEL can be continued; if a terminator does not appear on the first statement, the next is assumed to be a continuation of the first.

Default parameters cause a single file header in ANSI format for a 7-track tape in SI format. Any other label or data format to be written, or a tape to be read, must be declared explicitly.

9-track tape can be selected only by giving either a 9-track density parameter (HD, PE, or GE) or a code conversion parameter (US or EB).

Read or write:

R	Label is to be read and compared with parameters on the LABEL control statement. When R is used, the tape can be a candidate for auto-assignment by label name.
W	Label is to be written.

Label type:

Y	3000 series label.
Z	Label conforms to standard label of previous operating system. Character 12 of the VOL1 label specifies data density; otherwise Z labels are identical to U labels.
absent	Standard label conforming to ANSI.

Write ring:

RING	Write-enabled ring required in tape.
NORING	Write-enabled ring prohibited in tape..
absent	Parameter is set to installation-defined value.

Noise brackets:

IB — Inhibits system noise brackets. Recommended if tape is to be read on another system. Noise brackets always are inhibited on phase encoded tapes.

Tape characteristics:

D=d — Density. If omitted, density declared or implied by REQUEST prevails. For 7-track tapes:

LO†	200 bpi
HI	556 bpi
HY	800 bpi

For 9-track tapes, the d parameter determines density for writing only; data is always read at the recording density.

HD	800 bpi
PE	1600 cpi, phase encoded
GE††	6250 cpi, group encoded

F=f — Format of the file data. Default is SI format.

S	S tape format
L†††	L tape format

N=n — Code for conversion of 9-track tapes only. Default is installation defined.

US	ASCII code
EB	EBCDIC code

X=x — Disposition of tape.

IU	Inhibit physical unload
SV	Unload tape at end of job; notify operator to save
CK	Checkpoint dump written on tape
CI	Checkpoint dump and inhibit physical unload
CS	Checkpoint dump and save

Label fields:

L=z — Label name: 1-17 characters for ANSI or Z labels; 1-14 characters for Y labels. Default value is spaces.

V=v — Label field. Volume number specifying volume sequence in volume set. 1-4 digits for ANSI or Z labels; 1-2 digits for Y labels. Default is 0001 for ANSI or Z labels, 01 for Y labels.

---

†200 bpi can be read but not written by CDC 667 Tape Drives.

††6250 cpi density is supported only on CDC 679 GCR Tape Drives.

†††L format is supported only on 7-track tape drives and CDC 669/679 9-track Tape Drives.

E=e — Label field. Edition number specifying version of file. 1-2 digits. Default is 00.

T=t — Label field. Number of days file is to be retained, 1-3 digits. Default determined by installation. 999 is permanent retention. A retention period greater than 364 days results in the assignment of T=999.

C=c — Label field. Creation date, in format of 2 digits for year, 3 digits for day. Default is current date.

M=m — Label field. The operating system uses this parameter to establish that the current LABEL function applies to a member of a multifile set; m is the logical multifile set name as it appears on the REQUEST statement for this set, and it must be present for all LABEL statements referencing members of this multifile set. When the label is written on tape, the multifile field does not contain the logical set name; it contains the VSN for the first volume of the multifile set.

P=p — Label field. Position number indicating file within multifile set, 1-4 digits. Default is 0001. Not defined for 3000 series labels.

VSN=vsn — Volume serial number of 1-6 characters used to identify the tape for automatic assignment. Parameter can appear on VSN statement rather than LABEL statement. A VSN of SCRATCH or 0 specifies a scratch tape.

LABELMS (DEVICE SET LABELING)

LABELMS labels a device before it is used in a device set, places the volume serial number in the label, and establishes the type of access to the device. In addition, LABELMS can be used optionally to inhibit preallocation of space for customer engineering diagnostics, to specify information for subsequent access to the device, and to record known flaws on a device so that such areas are not accessed.

The format of LABELMS is:

LABELMS,DT=eq,mode,I=lfn.

All parameters are optional.

DT=eq — Device type. If DT is omitted, the operator can assign any device type. The value of eq is a device mnemonic; for example, AY for 844-21. (See section 6 for list of device types.) Member devices subsequently added by the ADDSET statement must have the same device type as the master device.

mode — Recording mode for an 844 disk pack. Default is defined at installation time.

HT	Half tracking
FT	Full tracking

I=lfn — Logical file name for input directives containing allocation and flaw information. If I is specified but not equivalenced, file INPUT is used; otherwise, no directives are expected. Consequently, default allocation information is used and the disk is presumed to be free of flaws. If this parameter is specified, DT must also be specified.

Input directive formats are as follows:

All values in the directives are assumed to be octal unless suffixed with a D.

Each directive must begin in column 1 and end with a valid terminator. Valid control separators must appear between the elements of a directive. Successive allocation directives must refer to successive portions of a device; allocation directives can be intermixed with flaw directives. A maximum of eight allocation directives is permitted.

Allocation directives: Aas,Rpru,Nrbs.

Flaw directives: { Ttn,Ccn,Ssn. / Ttn,Ccn,Sfsn–lsn.

as — Allocation style with limits of 0 to 77 (octal); default is as = 0. The user can request a specific allocation feature, such as directing a file to a specific portion of a device having a particular record block size and/or recording technique.

pru — Number of PRUs per record block. The pru value must be greater than or equal to 1/32 of the PB (physical block) size and less than or equal to 32 times the PB size. PB size depends on the device, as shown below. For compatibility with 844 density packs, the following pru values are listed. These values apply, regardless of whether or not the installation is using the 844 double density feature.

(RB size) $\leq$ 70B: 2, 4, 7, 10, 16, 34, 70
(RB size) > 70B: (2n-1)*70+1 $\leq$ pru value $\leq$ 2n*70, where n=1,2...20B

rbs — Number of record blocks in the RBR for this device or portion of device. The RBR, maintained by the operating system in central memory, contains information indicating its allocation style and the status (available for assignment) of all record blocks governed by this RBR. The limits of rbs are 1 to 7777 (octal). Default depends on the device as shown below.

tn	Track number	Limits depend on device as shown below
cn	Cylinder number	
sn	Sector number	
fsn	First sector number	Indicates several contiguous flaw sectors
lsn	Last sector number	

Device	PB Size (PRUs)	RB Size Default (PRUs)	rbs Default	tn Limits	cn Limits	sn Limits
841	70	70	1750	0 to 23	0 to 307	0 to 15
844-21	160	70	6240	0 to 22	0 to 632	0 to 27
844-41	160	70†	6240†	0 to 22	0 to 1466	0 to 27

All values listed above are octal.

†To create an 844-41 (double-density) pack with an RB size of 70B, two allocation directives must be input to LABELMS. 844-41's require two RBRs when the RB size is 70B.

NOTE

User packs cannot have the number of record blocks (RBs) greater than the installation-defined maximum number of record blocks to be used for private devices. All members of a user device set must have identical allocation directives specified when the devices are labeled.

For 844-21 (AY) and 844-41 (AZ), the flaws recorded on the device in the utility flaw map (UFM) are read by LABELMS (except during deadstart) and added to the flaws supplied in the input file. If the pack does not contain the flaw map, the following informative message is written to the job dayfile:

ERROR IN READING UFM

During deadstart. LABELMS obtains a complete set of flaws from IRCP through CMR–including the flaws from the utility flaw map read by IRCP.

LIMIT (LIMIT MASS STORAGE)

LIMIT limits the amount of rotating mass storage that is assigned to a job. Normally, a job is assigned as much mass storage as it needs; however, a user might want to limit the maximum mass storage that should be assigned, for example, during a debug phase when large amounts of output would indicate program errors. Any time mass storage in excess of the specified limit is required, the job terminates.

The format of LIMIT is:

LIMIT,n.

n — Number (octal) indicating the maximum number of blocks that can be allocated to the job. Blocks are 4096 60-bit words. The n parameter is required.

The value of the LIMIT parameter should anticipate both the number and size of files that exist at one time. The information in the mass storage accounting message in the dayfile might be helpful in determining a limit for the LIMIT control statement. Note that the dayfile message is in decimal words, but the LIMIT argument is in blocks of 4096 words. The mass storage statistic is issued only if a LIMIT control statement has been executed by the job or if the installation has set a non-zero default mass storage limit. Generally, very small limits should be avoided, since the system allocation of one record block, at minimum, for each file can exceed the limit established even though each file is small.

Record blocks are defined at each installation, usually with different sizes of blocks for different mass storage devices. A disk, for example, might have record blocks of 3200 words. A statement specifying LIMIT(2) would, in this instance, cause job termination when a third file is opened, since 3 times the record block size is more than the stated limit of 8192 words.

Mass storage occupied by the INPUT file or attached permanent files is not involved in the total mass storage allocation for LIMIT calculations. Any file evicted from mass storage decreases the count of words allocated.

LISTMF (LIST LABELED TAPE)

LISTMF lists the HDR1 labels of files in a multi-file set. The utility is valid only for tape files with ANSI standard labels. All volumes in the set are processed with a single utility call. The listing appears on the file OUTPUT.

A REQUEST control statement defining the multi-file set is required before LISTMF is called.

The format of LISTMF is:

LISTMF,M=mfn,P=p.

M=mfn — Multi-file name of the set, as declared on the REQUEST control statement. Required.

P=p — Position of file at which listing is to begin; 1-3 digits. The first file in the set is position 1. Default is 1.

The multi-file set is rewound at the beginning of LISTMF execution, then positioned to the beginning of the file indicated by the P parameter. Listing of header labels stops when the end of the set (EOF label followed by multiple tape marks) is reached. No further positioning occurs.

LOAD (LOAD PROGRAM)

LOAD loads a file into memory in anticipation of a call for execution of loaded programs. LOAD can initiate a load sequence or be part of an existing load sequence; it does not terminate a load sequence. An EXECUTE control statement, or, in the case of overlay preparation, a NOGO control statement, would normally terminate the load sequence.

LOAD is defined by the loader, not the operating system. See the LOADER reference manual for further details.

The format of LOAD is:

LOAD,lfn1/r,lfn2/r, ...

More than one parameter can be specified when all files contain relocatable programs. Only one parameter can be specified when the file contains an absolute program.

lfn — Logical file name of file containing binary executable code, 1-7 letters or digits beginning with a letter.

r — Rewind indicator:

- R — Rewind file prior to loading. Rewind of the file INPUT rewinds to the beginning of the control statements; no skipping of control statements occurs.
- NR — Inhibits rewind prior to loading.

Loading from the file terminates when a partition boundary, or end-of-information is encountered, or when two consecutive 7/8/9 cards are encountered in an image of a job deck.

LOADPF (LOAD PERMANENT FILE TO TAPE)

LOADPF loads permanent files that have been dumped to tape. All (or a selected portion of) files on the tape can be loaded. An optional directive file specifies individual files to be loaded. Multiple copies of LOADPF can execute at the same time. A job can access a file as soon as it is entered into the permanent file tables. For each cycle loaded, LOADPF makes an output listing entry that contains the permanent file name, owner ID, cycle number, date of last dump, and a comment.

Before LOADPF is called, a REQUEST or LABEL control statement must define a tape file named DUMTAPE in S format with an existing label. If the dump tape for a file to be loaded contains more than one file with the same permanent file name, cycle number, and ID name, a message is sent to the operator and the file is ignored. New cycles of a permanent file will not be loaded if the passwords of the tape cycle disagree with the existing cycle.

NOTE

Files purged between a full DUMPF and several change dumps (DUMPF,DP=C) are reloaded when both the change and full dumps are reloaded.

The format of LOADPF is:

LOADPF,LP=x,LF=lfn,CL,SN=setname,VSN=vsn,ID=name,PF=pfn,CY=cy, { I=lfn / I } .

All parameters are optional and order independent. Only one LP parameter can be specified. If a terminator does not appear at the end of the parameter list, column 1 of the next card or line is considered to be a continuation of the LOADPF parameter list.

LP=x

A	Load all files. Existing files are not replaced unless the file is incomplete or not disk resident. Default.
R	Replace existing files. Both X and R can be specified in the form LP=X,R.
P	Load archived files (files with entries in permanent file tables but file residence on tape).
X	Do not load expired files.
O	Permanent file dump tape is in SCOPE 3.2 or 3.3 format. If LP=O is not specified, the tape is assumed to be a SCOPE 3.4 permanent file dump tape. The O option can be used with other LP parameters in the form LP=R,O,X.
LF=lfn	Name of file on which listing is to appear, 1-7 letters or digits beginning with a letter. Default is OUTPUT.
CL	Complete list option selected. All files on the dump tape are listed. If CL is omitted, only loaded files are listed.
SN=setname	Name of device set to which files are loaded, 1-7 letters or digits beginning with a letter. Master device of this set must be previously mounted.
VSN=vsn	Volume serial number of the device onto which permanent files are loaded, 1-6 letters or digits with leading zeros assumed. Parameter SN must also be included, and the master device of the set must be previously mounted.
ID=name	Load files with this owner.

PF=pfn — Load files with this permanent file name. ID=owner is also required.

CY=cy — Load cycle cy of file specified by PF and ID. CY is ignored and the load continued if this cycle is not found, or if PF and ID are not specified.

I=lfn — Logical file name of directive file, 1-7 letters or digits beginning with a letter. If I is specified but not equivalenced, file INPUT is used.

A group of files to be loaded can be identified by name and owner in a directive record. When input directives are selected, only parameters SN and CL are valid on the LOADPF call. Parameters on directives are order independent. CY is optional. Directive format is:

```
PF=pfn,CY=cy,ID=name
```

LOADPF EXAMPLES

1.
```
JOB1.
REQUEST(DUMTAPE,HY,S,E)
LOADPF.
6/7/8/9
```

   This job loads all files on the tape unless LOADPF finds the owner ID, permanent file name, and cycle number combination already in the system; such files are skipped.

2.
```
JOB2.
REQUEST(DUMTAPE,HY,S,E)
LOADPF(LP=X)
6/7/8/9
```

   This job loads all non-expired permanent files from tape.

3.
```
JOB3.
REQUEST(DUMTAPE,HY,S,E)
LOADPF(PF=STARTREK,ID=SPOCK)
6/7/8/9
```

   All cycles of the permanent file STARTREK with owner ID SPOCK are loaded unless one of the following conditions arises:

   The permanent file name/owner ID combination already exists in the system with different passwords.

   A duplicate cycle number is encountered.

   The permanent file name/owner ID combination already has five cycles cataloged.

4. JOB4.
   REQUEST(DUMTAPE, . . .)
   LOADPF(I)
   7/8/9
   PF=PASSERIFORMES,CY=21,ID=VEERY
   PF=ANATINAE,ID=GADWELL
   PF=PROCELLARIIFORMES,ID=FULMAR
   6/7/8/9

   This job loads the specified permanent files from tape.

MAP (PRODUCE LOAD MAP)

MAP determines the extent of the load map produced for all subsequent programs loaded in central memory. When MAP is omitted, an installation default determines the type of map.

Output from a load map appears on the file OUTPUT. It includes items such as the type of load, location of programs, common blocks and tables, and entry points. Load maps of programs on the system library, such as compilers or assemblers, are never produced. See the LOADER reference manual for an explanation of all items in the load map.

The MAP option selected remains in effect until another MAP control statement changes the option or the job ends.

The format of MAP is:

$$\text{MAP,}\left\{\begin{matrix}\text{OFF}\\\text{FULL}\\\text{PART}\end{matrix}\right\}.$$

OFF	No map is produced.
FULL	Full map is produced.
PART	Map has all items except entry point addresses.

The effect of a MAP can be overridden for a particular load sequence by the MAP option of the loader statement LDSET (see the CYBER Loader Reference Manual).

MODE (SUSPEND ERROR EXIT)

MODE specifies the error conditions that abnormally terminate the job. Normally, a job terminates when any of the following CPU program errors are detected:

Reference to an operand (any number used in a calculation) that has an infinite value.

Reference to an address outside the field length of the job in central memory or ECS; such an address can be generated during assembly if a non-existent location is referenced or inadequate field length is set.

Reference to an operand for floating point arithmetic which has an indefinite value

When a selected error condition is detected, the job terminates. When an error condition not selected by MODE is detected, job processing continues and no error message is issued. A MODE selection remains in effect until another MODE control statement is executed or the job ends.

The format of MODE is:

MODE,m.

m	CPU program error exit conditions 0-7 (octal). If omitted, 7 is assumed.	
	0	Disable CPU program error exit; all errors allow job to continue
	1	Address is out of range
	2	Operand is infinite
	3	Both 1 and 2 remain in effect
	4	Floating point number of indefinite value
	5	Both 1 and 4 remain in effect
	6	Both 2 and 4 remain in effect
	7	1 and 2 and 4 remain in effect

For example, a MODE, 5. statement directs the system to continue processing even if an infinite operand is encountered. If an address is out of range or a floating point number of indefinite value is encountered, the job terminates. A control statement MODE,7. is equivalent to a job without a MODE control statement.

MOUNT (ASSOCIATE DEVICE SET)

MOUNT associates a device set and its members with a job. MOUNT is a logical operation; if the device is physically available, no operator intervention is required. If the device is not physically available, the device name is placed in an operator display, and the job is swapped out until the device is mounted.

When the master device is mounted, the device set tables are read into the system and all files and member devices become logically accessible to the job. The master device must remain mounted while the associated device set is in use. When the master is mounted, the system issues a MOUNT for other member devices as needed. The user also can issue a MOUNT for a member device.

The format of MOUNT is:

MOUNT,VSN=vsn,SN=setname,mode.

Parameters VSN and SN are required; mode is optional. All parameters are order independent.

VSN=vsn	Volume serial number of device to be mounted, 1-6 letters or digits with leading zeros assumed.
SN=setname	Name of device set to which this device belongs, 1-7 letters or digits beginning with a letter.
mode	Recording mode for an 844 disk pack. Default is defined at installation time. HT Half tracking FT Full tracking

PAUSE (OPERATOR INTERFACE)

PAUSE inserts a formal comment into the job dayfile and stops the job until the operator acknowledges the comment. PAUSE should not be used unless communication with the operator is essential. The COMMENT control statement allows messages to be inserted into the dayfile without the need for operator response.

The format of PAUSE is:

PAUSE. comment

Ending punctuation is not required.

comment	String of 74 characters to be displayed for the operator. Any characters can be specified, including those otherwise used as punctuation. Characters with display code values greater than 57 are displayed as blanks.

All eighty characters (PAUSE plus message) are displayed for the operator. A message longer than 74 characters can be sent by using a second PAUSE control statement, but each statement requires operator action.

The operator acknowledges the PAUSE message by a GO, DROP, or KILL command that continues, drops, or aborts the job, respectively.

PURGE (REMOVE PERMANENT FILE)

PURGE removes the permanent status of a file. The file remains as a local file for the job if the file is being accessed on the mainframe at which the job is executing, if the file is not archived, and if the RB parameter is not specified. Control permission is required to purge a file.

PURGE affects only one cycle of a permanent file. If it is the only cycle of the file, the permanent file name is removed from the permanent file tables. A subsequent CATALOG with the same permanent file name and ID would be an initial CATALOG.

The format of the control statement and subsequent file permissions depends on whether the file is already attached to the job. If the full format is specified when the file is attached, all parameters except lfn and RB are ignored.

If the file is attached to the job, the format of PURGE is:

PURGE,lfn,RB=1.

If the file is not attached to the job, the format of PURGE is:

PURGE,lfn,pfn,ID=name,AC=act, {LC=n / CY=cy} ,EC=ec,MR=m,PW=pw,RB=1,RW=p,SN=setname,ST=mmf.

Only lfn is required as long as the file is attached to the job. Parameters other than lfn and pfn are order independent. PURGE can be continued: if the parameter list is not terminated by a period or right parenthesis, column 1 of the next statement is considered to be a continuation of column 80.

lfn — Logical file name by which file is attached to the job, 1-7 letters or digits beginning with a letter.

RB=1 — Record block conflict. Applicable only when the record block conflict flag is set in system tables to indicate that storage allocation for the file is in conflict with mass storage allocation elsewhere. If this parameter is used when the conflict flag is set, the local file has all permissions removed except control permission and the mass storage associated with the file is not released when the file is released to the system. The AUDIT utility reveals the presence of files with storage conflict.

ST=mmf — System on which file is cataloged, 3 characters. If the file is not cataloged on the mainframe at which the job is executing, a job is generated on the specified mainframe to purge the file. If this parameter is specified, any SN parameter is ignored.

See the ATTACH control statement for the meaning of remaining parameters.

RECOVER (DEVICE SET MAINTENANCE)

RECOVER validates a device set and reconstructs tables whenever the integrity of a device set is in question. It scans critical disk tables of a device set to verify and recreate each. Any errors encountered during the recovery process are noted in the OUTPUT file. The RECOVER control statement is not executed if this job or any other job has issued instructions to mount the device set.

The format of RECOVER is:

RECOVER,SN=setname,VSN=vsn.

Parameters are required and order independent.

SN=setname — Name of device set to be validated or reconstructed, 1-7 letters or digits beginning with a letter.

VSN=vsn — Volume serial number of device set master device, 1-6 letters or digits with leading zeros assumed.

In a multi-mainframe environment the permanent file could be destroyed if RECOVER is executed when one of the mainframes has the master mounted. Therefore, the system aborts the request unless called from the console by an operator type-in.

REDUCE (REDUCE FIELD LENGTH)

REDUCE decreases the central memory field length assigned to a job to the amount of memory needed by the program currently loaded. It also restores dynamic field length management by the operating system that the job previously inhibited through execution of an RFL control statement or through use of a CM parameter on the job statement. REDUCE should be used whenever the job no longer requires special field length handling.

The format of REDUCE is:

REDUCE.

RENAME (CHANGE PERMANENT FILE TABLE)

RENAME changes values of parameters in the permanent file manager tables. Parameter values originating from a prior RENAME or original file catalog can be deleted or changed to different values and new parameters can be added. RENAME affects only the parameters specified on the control statement; other parameters remain as they were.

Prior to issuing RENAME, the job must attach the file with read, extend, modify, and control permission.

The format of RENAME is:

RENAME,lfn,pfn,ID=name,AC=act,CN=cn,CY=cy,EX=ex,MD=md,RD=rd,RP=rp,TK=tk,XR=xr.

Only the lfn parameter is required; it must be the first parameter. All other parameters are optional and order independent. RENAME can be continued: if the parameter list is not terminated by a period or right parenthesis, column 1 of the next statement is considered to be a continuation of column 80. Two commas can follow lfn when pfn is not changed.

Specifying the parameter name and an equals sign without a following parameter value removes the existing value for that parameter.

lfn — Logical file name of attached permanent file, 1-7 letters or digits beginning with a letter. Required.

RP — Retention period, 0-999. Applies to date of original CATALOG, not to date of RENAME.

See the CATALOG control statement for the meaning of remaining parameters.

Any change to the permanent file name, ID, or passwords of any cycle of a file causes the same change to be made for all cycles of the file. Consequently, RENAME cannot change the permanent file name, ID, or passwords if any cycle of the file has been dumped or archived to tape. If the PFN/ID are being changed and a file already exists with the proposed PFN/ID, the PFN/ID change will not occur; a nonfatal error message is issued.

REQUEST (ASSIGN FILE TO DEVICE)

REQUEST requests assignment of a file to a particular device. Since control statements are processed in order of appearance, the REQUEST statement for a particular file must precede the control statement that executes the program referencing that file. Otherwise, the file is sought or written on a public scratch device when it is referenced.

REQUEST is most commonly used with permanent files, magnetic tapes, and private device sets, but it can be used to cause file assignment to any public device or unit record equipment. Files are assigned to public disk packs by a REQUEST or by system default; but to ensure that a file is assigned to a permanent file device, a REQUEST statement with a *PF parameter should be used.

When a REQUEST control statement is encountered, job processing might halt for operator action or continue with operating system action, depending on the form of the parameter specifying device type; and, for magnetic tape, the installation tape assigning options.

The general form of REQUEST is:

REQUEST,lfn,dt,parameters.

Parameter lfn is required and must be the first defined; all other parameters are optional and order independent.

lfn — Logical file name by which file will be known throughout the job, 1-7 letters or digits beginning with a letter. lfn beginning with ZZZZZ is reserved for the system.

dt — Device type mnemonic plus other dt parameters to further describe equipment requested. If the user specifies an optional device type parameter which is unique to a device type (for example, the GE parameter for a 9-track tape), the device type mnemonic need not be specified. A preceding asterisk allows assignment of devices without operator action if possible. An asterisk is implied for mass storage devices.

parameters — Optional parameters.

The optional device type descriptors depend on the category of equipment involved. Details of parameters for REQUEST are discussed separately in relation to files on the following devices:

Magnetic tapes (7- and 9-track) including multi-file sets

Unit record devices such as card reader and line printer

ECS

Public devices including those used for permanent files

An asterisk preceding the device type mnemonic causes the operating system to attempt to assign the device without operator action. Automatic assignment is attempted on mass storage devices regardless of whether the asterisk is specified. The tape assigning options available make the * redundant for magnetic tape requests, but it can be used; however, * cannot be used if two units are requested with the same control statement or a multi-file set is involved. If * is used for unit record devices, the REQUEST control statement appears on the operator display for manual assignment. The operator must then make the unit physically ready and logically assign it to the job by entering a command on the console keyboard. See Unit Record Device Request description which follows in this section.

When sufficient information is given on the REQUEST control statement, the operating system assigns the device to the job without operator action. For rotating mass storage devices, automatic assignment is attempted whether or not the asterisk precedes the dt parameter. For other device requests, operator action is required if an asterisk does not precede the dt parameter. If dt is not declared, the operator can assign any device. For tape request, a VSN parameter is used to locate and to assign the tape if it is mounted.

The operating system compares the device assigned by the operator with the request; any discrepancy is reported to the operator. An additional operator command must be given if the dt parameter on the control statement is to be overridden by manual assignment. Conflicts must be resolved by the operator.

TAPE FILE REQUEST

The REQUEST control statement can describe both physical and logical characteristics for magnetic tape files. When only the logical file name and magnetic tape device type MT are specified, the file, by default, becomes a 7-track, unlabeled tape with SI format written at installation density, or read at written density; and installation declarations for automatic unloading are honored. Any other use, such as for checkpoints or multi-file sets, or characteristics of the file must be specifically declared.

The MT or NT device type parameter can be prefixed by an asterisk or a 2. The asterisk is applicable only when compatibility with previous operating systems is considered. The asterisk prefix results in assignment of a scratch tape to the file. However, if a non-scratch VSN has been specified also, it overrides the scratch designation. If REQUEST includes parameter E, a scratch tape is not assigned. Depending upon the selection of installation options, the operating system attempts to assign the tape to a job automatically using a VSN, or label name parameter. Operator assignment is necessary only when automatic assignment attempts are unsuccessful.

If either a 7- or 9-track tape is acceptable, an MN parameter can be used in place of MT or NT. The resulting tape has default density. If the request includes at least one device type descriptor which is unique to magnetic tapes (such as the RING parameter), neither the device type nor the density need be specified.

A 2 prefix to MT or NT causes two tape units to be requested from the operator; they are used in the order assigned. Tape requests using the 2 prefix cannot be auto-assigned. When the tape on the first unit reaches end-of-volume, the system begins processing the tape on the second unit while the tape on the first unit is rewound and unloaded. When the tape on the second unit reaches end-of-volume, the system returns to the first unit, which should have been mounted in the interim with a new tape. The tape on the second unit is rewound and unloaded. This alternating process is repeated as long as the file is referenced. The operator must ensure the proper tape mounting sequence.

7-TRACK TAPE PARAMETERS:

REQUEST,lfn,MT,{LO / HI / HY},{CK / MF},{S / L},{U / Y / Z},{E / N / NS},{IU / SV},{RING / NORING},IB,NR,VSN=vsn.

Logical file name:

If the MF parameter is not specified, lfn is the logical file name of 1-7 letters or digits beginning with a letter.

If the MF parameter is specified, this parameter is a multi-file set name of 1-6 letters or digits beginning with a letter.

The multi-file set name cannot be used in any input/output statement except as the M parameter in a LABEL statement or POSMF macro.

7-track identification:

A declaration of LO, HI, or HY is sufficient to define the device type as MT. If MT is absent, LO, HI or HY can be prefixed by a 2 if two units are required.

Density:

LO†	200 bpi density
HI	556 bpi density
HY	800 bpi density
absent	Density is set to an installation defined value if initial use is output. If initial use of a label tape is input, the density of the label is determined automatically; however, it is recommended that density be specified whenever known, and that density be used to read both the label and the data, except as indicated under Z below. If initial use of an unlabeled tape is input, the density is set to an installation declared value.

File disposition:

IU	Any physical unload of the tape file in a context other than reel swapping is inhibited. The IU parameter does not inhibit logical actions associated with UNLOAD or RETURN. IU is recommended when a scratch tape or input tape is requested that is to remain mounted and ready.
SV	The tape file is unloaded at job termination, and the operator is notified that the tape is to be saved.
absent	Action performed at end-of-job is option of the installation.

†CDC 667 Tape Units can read but not write at 200 bpi. If the installation has both 667 and 607/657 units, jobs writing at 200 bpi must not be assigned 667 tape units.

Tape security:

RING — Write-enable ring required in tape.

NORING — Write-enable ring prohibited in tape.

absent — RING/NORING is set to an installation defined value.

Volume serial number identification:

VSN=vsn — Volume serial number of the tape volume, 1-6 letters or digits with leading zeros assumed. The VSN appears on the previewing display for the operator's information before the job is assigned to a control point. Once the tape is mounted and the unit made ready, the operating system can locate the volume without further operator action. Once the tape is assigned, the VSN is verified against the standard or Z format label, if present. VSN also is verified against operator-supplied VSN for an unlabeled tape.

If a scratch tape is desired, a VSN of SCRATCH or 0 can be used. The * prefix can be used for a scratch tape also.

If a VSN parameter is declared for a file on a REQUEST, and a VSN control statement or a VSN parameter on a LABEL control statement also appears, the first declaration is effective.

absent — Any VSN declaration is used; otherwise, file header label fields are used for assignment and verification. If neither VSN nor file header label field declaration is made, any tape volume is accepted; but the assignment must be made manually unless * prefix is used.

Parity error recovery procedure:

NR — The NR parameter can be used to inhibit normal parity error recovery procedures. Data containing the parity error is returned to the user.

Special tape use:

CK — Checkpoint dumps are written on the tape.

MF — The tape is a valid U or Z labeled multi-file set.

absent — Neither of the above is assumed.

Inhibit system noise brackets:

IB — The IB parameter inhibits system noise brackets. Use of this parameter is recommended if the tape is to be read on another system.

Data format:

S	Data format is S.
L	Data format is L.
absent	Data format is SI format.

Input or output use (apply only to labeled tapes):

E	Existing label. Initial use of the tape is input; only the expiration date is checked in the label.
N	New label. Initial use of the tape is output; tape label is written.
absent	If file is to be labeled (U, Z or Y is declared), a tape label is written.

Label characteristics:

U	Tape label format is ANSI (standard label)
Y	Tape label format is Y (3000 series label).
Z	Tape label format is ANSI, except character 12, of the VOL1 label is used to indicate data density. These labels were standard for SCOPE 3.3.
absent	Tape is unlabeled unless either E or N is declared; in which case, ANSI (U) label format is assumed.

Label processing:

NS	The NS parameter can be used to indicate a tape has non-standard labels and is to be processed as unlabeled even though the tape is labeled; existing labels appear to the system as data and are passed to the user as such. The user can then process the labels or ignore them. Non-standard labels are not supported on SI tapes.

9-TRACK TAPE PARAMETERS:

A declaration of NT or a 9-track density for a tape to be written is required to identify a 9-track tape. Definitions and conditions for all except the density and data format parameters are the same as those for 7-track tape.

$$\text{REQUEST,lfn,NT,}\left\{\begin{matrix}\text{PE}\\\text{HD}\\\text{GE}\end{matrix}\right\},\left\{\begin{matrix}\text{S}\\\text{L}\end{matrix}\right\},\left\{\begin{matrix}\text{CK}\\\text{MF}\end{matrix}\right\},\left\{\begin{matrix}\text{U}\\\text{Y}\\\text{Z}\end{matrix}\right\},\left\{\begin{matrix}\text{E}\\\text{N}\\\text{NS}\end{matrix}\right\},\left\{\begin{matrix}\text{US}\\\text{EB}\end{matrix}\right\},\left\{\begin{matrix}\text{IU}\\\text{SV}\end{matrix}\right\},\left\{\begin{matrix}\text{RING}\\\text{NORING}\end{matrix}\right\},\text{IB,NR,VSN=vsn.}$$

Density:

A density specification is effective only when the tape is to be written: density setting is a hardware function when the tape is read.

PE	1600 cpi
HD	800 cpi
GE	6250 cpi
absent	Tape is written at installation-declared density.

Data format:

S	Data format is S.
L†	Data format is L.
absent	Data format is SI format.

Inhibit system noise brackets:

IB	The IB parameter inhibits system noise brackets. Use of this parameter is recommended if the tape is to be read on another system. Noise brackets always are inhibited on phase encoded tapes.

Coded data conversion codes for 9-track S or L tapes (refer to conversion tables in appendix A):

US	Coded data on tape is to be converted from ASCII on input or to ASCII on output.
EB	Coded data on tape is to be converted from EBCDIC on input or to EBCDIC on output.
absent	Coded data conversion is defined by the installation.

Examples of REQUEST statements for tapes:

1. `REQUEST(FILE1,NT,U,E,NORING)` or `REQUEST(FILE1,NT,E,NORING)`

   The operator must assign an ANSI labeled, 9-track tape. The label is checked when the first function is issued on the tape. Because density is not specified, it is assumed that both the label and data are written at the same density.

2. `REQUEST(FILE,*MT,RING)`

   Depending on installation option, the system automatically assigns FILE1 to a scratch tape on a 7-track tape unit. The file is unlabeled and written in SI data format at an installation-declared density.

---

†Currently L tapes are supported only on 7-track tape devices and CDC 669/679 9-track Tape Drives.

3. REQUEST(STANF27,LO,VSN=OHIO17,U,S,SV,RING)

Depending on installation option, file STANF27 is assigned automatically to a unit containing volume OHIO17. An ANSI label is written; both label and data are written at 200 bpi. Data format is S. The volume is saved at job completion.

UNIT RECORD DEVICE REQUEST

When a file is input from a card reader or output to a printer or card punch, devices are assigned automatically; REQUEST is not necessary. There are no standard drivers for the unit record equipment; request and assignment of such devices is only valid for on-line diagnostic packages or for devices for which the installation has provided drivers. If the installation has provided drivers, the following devices can be requested. Assignment is not automatic; the operator must assign the requested device to the job.

REQUEST,lfn,dt.

lfn — Logical file name of 1-7 letters or digits beginning with a letter.

dt — Device type. The following device types are recognized, but not supported by the standard system. If an installation provides software drivers for these devices, they can be specified.

LP	Any available line printer	GC	252-2 Graphics Console
LQ	512 Line Printer	HC	253-2 Hardcopy Recorder
LR	580-12 Line Printer	FM	254-2 Microfilm Recorder
LS	580-16 Line Printer	TR	Paper tape reader
LT	580-20 Line Printer	TP	Paper tape punch
CR	405 Card Reader	PL	Plotter
CP	415 Card Punch		

ECS FILE REQUEST

Files that are to reside on ECS are requested by the following control statement; this statement is not to be used for files that are buffered through ECS.

REQUEST,lfn,AX,EC.

lfn — Logical file name of 1-7 letters or digits beginning with a letter.

AX — ECS device type mnemonic. Required.

EC — Maximum file size. If omitted, default buffer size is the maximum file size.

- EC — Default buffer size maximum.
- ECnnnn / ECnnnnK — Maximum size nnnn words multiplied by 1000 (octal).
- ECnnnnP — Maximum size nnnn ECS pages, where page size is 1000 (octal) 60-bit words.

If ECS is turned OFF, the files requested on ECS are allocated on rotating mass storage devices.

MASS STORAGE FILE REQUEST

Mass storage files on either public device sets or private device sets are requested as follows. The EC parameter is valid only for files on public device sets.

For private device sets, a MOUNT control statement must assign the master device to the job before REQUEST assigns a file to the device set.

REQUEST,lfn,dtaa,OV,EC,*PF,*Q, { SN / SN=setname } ,VSN=vsn.

The first parameter must be lfn. Other parameters are optional and order independent.

lfn — Logical file name of 1-7 letters or digits beginning with a letter.

dtaa — Device type mnemonic and allocation style. An asterisk can appear before dt, but its function is redundant.

dt — Device type mnemonic for a mass storage device:

AM	841 Multiple Disk Drive	AZ	844-41 Disk Drive
AY	844-21 Disk Drive	A*	Any mass storage device
AH	819 Disk Drive (CYBER 170 Model 176 only)		

aa — Octal allocation style defined by the installation for public sets; by LABELMS for user device sets. Can be null.

OV — Overflow to any other mass storage device is allowed when device dtaa or a device specified by SN and VSN parameters is unavailable or full. Permanent files are assigned only to permanent file devices and queue devices, respectively; otherwise, the file might be assigned to any mass storage. If all mass storage of any type becomes unavailable, a device capacity exceeded status is returned to a COMPASS program when the EP bit is set in the FET. When OV is omitted, and requested device is not available or full, all parameters are ignored except *PF, *Q, and SN as the system selects the device on which to continue.

EC — Buffer file through ECS. Valid only for sequential files on public devices.† If ECS is OFF, this parameter is ignored for this job.

The EC parameter can also be used on a CYBER 170 Model 176 to request a specific number of LCM buffers for buffering data to the 819 disk. If the user specifies AH (819 Disk Drive) without specifying the EC parameter, the default number of LCM buffers is assigned.

EC — Default buffer size.

ECnn — Number of 512_{10}-word buffers (nn) to be assigned.

†All file types will be buffered for device type AH (CYBER 170 Model 176 only).

	ECnnnn	Buffer size of nnnn 60-bit words multiplied by 1000 (octal).
	ECnnnnK	
	ECnnnnP	Buffer size of nnnn ECS pages, where page size is 1000 (octal) 60-bit words.
*PF	Assign file to a permanent file device. If SN and VSN specify a permanent file device, *PF is not required. If SN is not specified, the file is assigned to the default *PF set.	
*Q	File is to be assigned to a queue device. If SN is a private device set, *Q is not allowed. If SN is not specified, the file is assigned to the queue set.	
SN	Assign file to setname. If omitted, file is assigned to a public device set.	
	SN	Setname specified by SETNAME control statement; if SETNAME has not been specified previously, file is assigned to a public device.
	setname	Name of set. 1-7 letters or digits beginning with a letter.
VSN=vsn	Volume serial number of device within set specified by SN, 1-6 letters or digits with leading zeros assumed. VSN cannot be used without the SN parameter.	

Allocation style aa is an optional appendage to the device type mnemonic. Two digit octal codes representing allocation style must be defined at each installation; they can be used to identify sub-areas of a device. For example, an installation can divide 844 disk packs into two sub-areas – default and large space allocation. If the large space allocation area is identified as allocation style aa=55, files residing in the large space allocation sub-area are assigned more units of disk storage than similar files residing in the default sub-area. At this example installation, a file is assigned large space allocation sub-area by REQUEST(lfn,AY55).

RESTART (RESTART JOB FROM CHECKPOINT TAPE)

RESTART restarts a job from a checkpoint tape. After locating the proper dump on the checkpoint tape, the restart program requests all tape files defined at checkpoint time, and repositions these files. Then a request is made for all mass storage files and ECS buffer length where applicable. Files are copied from the checkpoint tape and repositioned. RESTART also restores the central memory field length of the job and restarts the user's program. If a permanent file was attached to the job when a checkpoint was called, it is attached and positioned as it was at the time of the checkpoint.

A restart job requires only a control statement to request the checkpoint tape (REQUEST or LABEL) and the RESTART control statement. If a checkpoint tape is not requested, the restart program requests an unlabeled 7-track or 9-track tape (for the file named on the RESTART control statement) as follows:

```
REQUEST(lfn,CK,MN)
```

Since RESTART recreates all files used for the checkpointed job, the user should not create any files before the call to RESTART. If any of those files are recreated by the user before the call to RESTART, a duplicate file error might occur.

If a device set was mounted when the checkpoint was taken, the job issuing the RESTART must execute a MOUNT control statement for the device set before calling RESTART; RESTART does not mount device sets. Files on device sets are attached and positioned by RESTART.

Any ECS direct access user area attached to the job is copied in its entirety to the checkpoint tape. At restart time, it is recopied to ECS from the checkpoint file. On the job statement for the restart job, the user must request at least as much ECS as was attached to the original job. If reconfiguration results in insufficient ECS available to the user, restart is not possible. The RESTART statement should not be used within a CCL procedure (see section 5).

The format of RESTART is:

RESTART,name,n,S=xxx.

All parameters are optional and order independent.

name	Name of checkpoint file as defined at checkpoint time. Default is CCCCCCC.
n	Number (decimal) of checkpoint to be restarted. If n is greater than the number of the last checkpoint taken, the restart attempt is terminated. Default is 1.
S=xxx	Ignored by RESTART: allowed for compatibility with previous systems.

A checkpoint dump cannot be restarted in the following cases:

A tape file necessary for restarting the program was overwritten after the checkpoint dump was taken.

A machine error propagated bad results but did not cause abnormal termination until after another checkpoint dump.

RETURN (EVICT FILE)

RETURN performs an operating system CLOSE/RETURN function. It differs from the UNLOAD control statement only in that RETURN reduces the maximum number of tapes that can be held by the job, but UNLOAD does not. RETURN deletes all references to the files specified, except as noted below, and destroys the file contents of local files.

The format of RETURN is:

RETURN,lfn1,lfn2, ...

More than one file or multi-file set can be specified; only one is required.

lfn	Name of file to be returned, 1-7 letters or digits beginning with a letter. lfn cannot be INPUT.
	Name of multi-file set tape to be returned, 1-6 letters or digits beginning with a letter.

For magnetic tape output files, RETURN causes trailer labels to be written, the file to be rewound, and then to be unloaded. With the exception of members of a multi-file set, the tape units on which the file resides is disassociated from the job and made available to the system for new assignment. The count of the number of tape units logically required by the job, as set by a tape parameter on the job statement, is then decreased.

For multi-file set names, the tape units assigned to the set are disassociated from the job and made available to the system for new assignment. The count of the number of tape drives required is then decreased.

For mass storage files, RETURN causes the file to be returned. Special-named files on queue devices are released to the output queue associated with their dispositions: if any of the special-named files are to be evicted, the DISPOSE or ROUTE control statement should be used instead of RETURN. Permanent files return to permanent file manager jurisdiction. Other mass storage files are evicted.

REWIND (REWIND FILE)

REWIND positions a file at the beginning-of-information.

For a labeled magnetic tape, this position is the start of the user's data after label information.

For unlabeled multi-volume tapes, a REWIND causes the current volume to be rewound.

For labeled multi-volume, single-file tapes, a REWIND causes the current volume to be rewound and the volume number in the system tables to be set to 1. A subsequent forward motion causes the label to be read and compared with the system tables, and the operator is notified if the current volume is not number 1.

For labeled multi-file tapes, a REWIND issued for a file causes positioning to the beginning of that file. If necessary, the operator is instructed to mount the previous volume.

The format of REWIND is:

REWIND,lfn1,lfn2, ...

More than one file can be specified; only one is required.

lfn Name of file to be rewound, 1-7 letters or digits beginning with a letter.

A REWIND that references a multi-file set name is illegal; the job terminates.

In most cases, when a file is requested for a job, that file is positioned automatically at beginning-of-information. However, because of variations in installation parameters and procedures, automatic positioning can not always occur with every file requested. Therefore, it is best to follow the REQUEST statement with a REWIND statement to ensure that the file is positioned at its beginning when first referenced.

RFL (REQUEST FIELD LENGTH)

RFL requests a specific central memory field length and inhibits dynamic field length management by the operating system. RFL should not be used unless the job has special requirements. A REDUCE control statement should immediately follow the operation that requires RFL use, so that dynamic field length management is restored.

For most jobs, the amount of central memory required varies with each job step. For example, a FORTRAN compilation might require 45000 words and a COPY routine might require 5000 words. System usage can be

improved when memory not currently needed is freed for other jobs. The system automatically increases or decreases the field length assigned to a job to optimize use of system storage.

If a job step needs more storage than would be assigned normally, RFL can be used to specify the maximum field length required. RFL can increase or decrease field length.

The format of RFL is:

RFL,fl.

fl	New field length (octal). Maximum value is established by the value of the CM parameter on the job statement, if any, or by an installation-determined value. The fl parameter must be specified; there is no default.

ROUTE (FILE DISPOSITION)

ROUTE directs a file to an input or output queue. Both file destination and type of further processing can be specified by control statement parameters. ROUTE is concerned with handling a file after it is released from the job, so it is not applicable to files with a fixed residence such as permanent files, private device set files, or files residing on other non-allocatable equipment. Unless deferred routing is requested, the file is released from the job immediately.

The file must be resident on a queue device. This can be assured by specifying *Q on a REQUEST statement.

The characteristics of a file that can be specified by ROUTE are:

Disposition code	Print, punch, etc.
Deferred routing	Do not release the file immediately
External characteristics	Punch card format or print train
Forms code	Particular paper or card forms to use
File ID	Name identifying the file while it is in the output queue, this name is printed on the banner page of a printout or punched on the lace card of a punch card deck
Internal characteristics	Data is in display code, ASCII, or binary format
Priority	Priority of file to be output at originating INTERCOM terminal
Repeat count	Number of extra copies for output files
Spacing code	Octal number of the array to be used with the 580 PFC printer
Station ID	Logical identifier of the computer to process the file
Terminal ID	Central site or identifier of the INTERCOM terminal to receive the file

Unlike DISPOSE, deferred routing can be used with INTERCOM terminal ID and forms code on a ROUTE control statement.

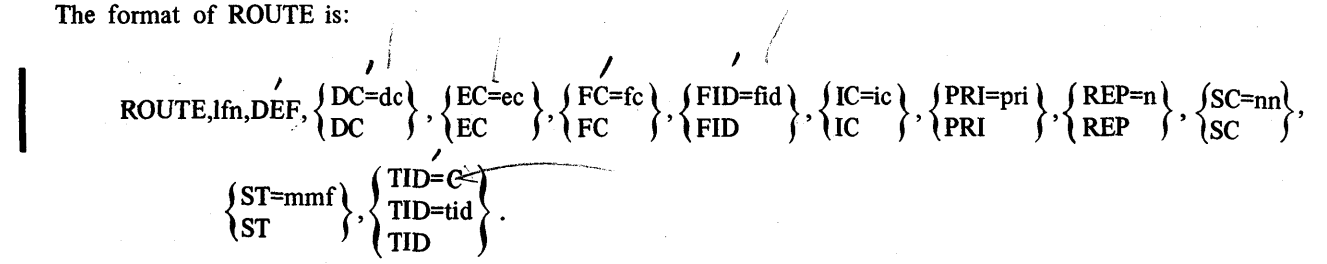
Files on public mass storage devices, except those with the special names listed below, receive a disposition code of scratch when they are created. At end-of-job or when the file is returned, such a file is discarded.

Files with special names receive specific disposition and external and internal characteristic codes when they are created. These files are sent to the predetermined destination at end-of-job or when returned. If a special-named file is to be discarded, DISPOSE or ROUTE must be used. The file names with special codes are listed below:

Special File Name	Destination	Default DC	Default EC	Default IC
OUTPUT	Print on any available printer with standard print train	PR	A6 or B6††	DIS
PUNCH	Punch in Hollerith format	PU	026 or 029††	DIS
PUNCHB	Punch in standard binary format	PU	SB	BIN
P80C	Punch in free-form binary format	PU	80COL	BIN
FILMPR†	Print on microfilm recorder	FR†	– – –	– – –
FILMFL†	Plot on microfilm recorder	FL†	– – –	– – –
HARDPR†	Print on hardcopy device	HR†	– – –	– – –
HARDFL†	Plot on hardcopy device	HL†	– – –	– – –
PLOT†	Plot on any available plotter	PT†	– – –	– – –

Format of files routed to the input queue can be dictated by operating system convention. If keywords FID, IC, EC, or FC are used in conjunction with DC=IN, they are ignored and no warning message is issued.

The format of ROUTE is:



Only lfn is required. All other parameters are optional and order independent.

lfn — Logical file name of the file to be routed, 1-7 letters or digits beginning with a letter. lfn cannot be INPUT.

DEF — Defer file disposition. The system stores the information about the file and disposes it as requested when the file is released. Files are released by RETURN and UNLOAD control statements, ROUTE or DISPOSE statements that specify immediate release, or at end-of-job. Routing of files to the input queue cannot be deferred. With deferred routing, the user can redefine the same file with subsequent ROUTE statements or specify characteristics of a file before the file is created.

---

†Supporting software must be supplied by the installation.
††Depends on installation parameter.

DEF used with DC=IN causes the ROUTE statement to be ignored. If omitted, file is released at ROUTE execution. DEF used with DC=SC, or DC not equivalenced, causes all user generated output to be discarded. The dayfile is not discarded.

DC=dc — File disposition:

SC	Evict (scratch) the file (default)	FR†	Print on microfilm recorder
PR	Print on any available printer	FL†	Plot on microfilm recorder
P2	Print on 512 printer	HR†	Print on hardcopy device
LR	Print on 580-12 printer	HL†	Plot on hardcopy device
LS	Print on 580-16 printer	PT†	Plot on any available plotter
LT	Print on 580-20 printer	IN	Place file in the input queue
PU	Punch		

Use of DC=IN can be restricted by the installation.

DC — Equivalent to DC=SC.

EC=ec — External characteristics of the print or punch file.

Print Files:

B4	Print format BCD 48 character print train
B6	Print format BCD 64 character print train
A4	Print format ASCII 48 character print train
A6	Print format ASCII 64 character print train
A9	Print format ASCII 95 character print train

Default value for JANUS print files is B6 or A6 depending on installation option. If EC=A9 is specified, JANUS will not print the file unless IC=ASCII is also specified. For all other print EC values, JANUS requires IC=DIS. INTERCOM print files with a default EC code are printed on any available train other than B4 or A4. The print trains normally mounted for output from INTERCOM terminals are:

BCD 200UT	B6
ASCII 200UT	A6
730 Series batch terminal	A6
711 and 714 terminal	A9

Punch Files:

026	Punch format 026
029	Punch format 029
ASCII	Punch format ASCII
SB	Punch format binary
80COL	Punch format 80 column binary

Default value for JANUS punch files is 026 or 029 depending on installation option. The only INTERCOM terminal with a punch is the 733; the default is determined when the terminal is auto-loaded.

†Supporting software must be supplied by the installation.

EC — Use default value for external characteristics.

FC=fc — Forms code, where fc can be any two letters or digits. This parameter indicates special card or paper forms are to be used for output. The operator should be informed of the meaning of the codes so that the proper forms are mounted. Each installation, typically, establishes procedures for using forms codes.

FC — Use standard forms. Default.

FID=fid — File name while the file is in the output queue.

- `*` — First five characters of the file name are the same as the first five characters of the job name. Two unique sequence numbers, different from the job sequence numbers, are added in the sixth and seventh positions.
- `fffff` — First five characters of the file name are fffff. This name is printed on the banner page of a printout or punched on the lace card of a punch card deck. Any combination of one to five letters or digits can be specified, with the first character a letter. The two unique job sequence characters added by the system to the job name are used as the sixth and seventh characters of the file name. If fffff is less than five characters, the name is filled with display code zero through the fifth position.
- `*fffff` — Equivalent to FID=fffff except two unique sequence numbers, other than the job sequence numbers, are added in the sixth and seventh positions.

FID — File name while the file is in the output queue is the same as the job name. Default.

IC=ic — Internal characteristics of the file:

DIS	File format is display code. Default
ASCII	File format is ASCII
BIN	File format is binary

IC=DIS is required by JANUS for all print files except where EC=A9, in which case, IC=ASCII is required. Files can be printed at INTERCOM terminals when IC=DIS or IC=ASCII is specified with any EC parameter. IC=BIN is required for binary punch files.

IC — Equivalent to IC=DIS.

PRI=pri — Priority level for a file to be output at originating INTERCOM terminal, 1-4 (octal) digits. PRI can be used to enter a priority for a file to be entered into the remote output queue. In any other instance, the parameter is ignored.

PRI — File receives standard priority. Default.

REP=n — Repeat count for output files, $n \leq 37B$.

REP — Default value, zero extra copies.

SC=nn — Spacing code for output sent to a 580 PFC printer. nn is an octal value, 0 to 77B, indicating an installation-defined spacing code array. Zero indicates the default array. All other values of nn are defined at the installation. See a site analyst for valid nn values.

SC — Equivalent to SC=0.

ST=mmf — The logical identifier of the system responsible for processing the file. If DC=IN, mmf is the logical identifier of the system where the job is executed. The ST parameter on the ROUTE control statement overrides an ST parameter on the job statement of the routed file. If the DC parameter specifies an output queue, mmf is the system where the file is output.

ST — Process the file on the system where it originated.

TID=tid — INTERCOM terminal identification. File is to be returned to terminal identified.

TID=C — File is to be output at central site.

TID — File is to be returned to the site or terminal where the job originated. Default.

Examples of ROUTE usage:

1.
```
job statement
ROUTE(LOON,DEF,DC=PR,EC=A9,IC=ASCII)
.
.
.
EXIT.
ROUTE(LOON,DC=SC)
7/8/9
.
.
.
6/7/8/9
```

This job creates a long file in ASCII format for a printer with an ASCII 95-character print train. If job aborts, the file is scratched. If job terminates normally, file LOON is printed after operator mounts 95-character print train. Note that the file is referenced before it is created. The routing information is saved and used when the file is sent to the output queue.

2.
```
job statement
COPY(INPUT,SWALLOW)
ROUTE(SWALLOW,DC=IN)
.
.
.
7/8/9
SWALLOW,STABC.
.
.
.
```

```
7/8/9
.
.
.
6/7/8/9
```

The job file SWALLOW is executed on system ABC.

3.
```
job statement
COPYBF(INPUT,FALCON)
ROUTE(FALCON,DC=IN,ST=ABC)
7/8/9
HAWK,T100.
COPYBF(INPUT,OWL)
REWIND(OWL)
COPYBF(OWL,EAGLE)
ROUTE(OWL,DC=PR)
ROUTE(EAGLE,DC=PR,ST=DOG)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

This job creates a file FALCON which is all but the control statements of the job. File FALCON is sent to the input queue of system ABC where it is known as job HAWK. Job HAWK produces file OWL to be printed on system ABC and file EAGLE to be printed on system DOG.

4.
```
job statement
COPY(INPUT,SWIFT)
OURTE(SWIFT,DC=IN,ST=DOG)
.
.
.
7/8/9
SWIFT,STABC.
.
.
.
7/8/9
.
.
.
6/7/8/9
```

When the ST parameter is specified on ROUTE and on the job statement of the file being routed, the ROUTE control statement overrules the job statement. Job SWIFT is executed on system DOG.

5. job statement

```
.
.
.
ROUTE(PIPIT,DEF,DC=PR)
.
.
.
RETURN(PIPIT)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

When the control statement RETURN(PIPIT) is executed, the file PIPIT is sent to the output queue to be printed. PIPIT is not scratched.

6. job statement

```
.
.
.
ROUTE(GREBE,DEF,EC=A6,IC-ASCII)
.
.
.
ROUTE(GREBE,DEF,EC=A9)
.
.
.
ROUTE(GREBE,DC=PR)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

The file named GREBE is printed on a printer with a 96-character ASCII print train. When the first ROUTE is executed, an EC of A6 and IC of ASCII are recorded. When the second ROUTE is executed, the EC is changed to A9. Since the IC parameter does not appear, its value does not change. When the third ROUTE is executed, the file GREBE is sent to the output queue to be printed. Subsequent references to an lfn of GREBE refer to a new file with the same name.

7. MURRE.

```
.
.
.
ROUTE(ALCID,FID=*,DC=PR)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

Suppose the two unique sequence characters added to the job name by the system are 3F. The job is then known as MURRE3F. If the next sequence characters were 3Z when ROUTE is executed, the file ALCID would be given the name MURRE3Z when it is printed.

8. BIRDS.

```
.
.
.
ROUTE(TERN,FID=*TERN)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

Suppose the sequence characters are as in example 7. Then the file TERN is printed as TERN03Z.

9. BIRDS.

```
.
.
.
ROUTE(TERN,FID=TERN)
.
.
.
7/8/9
.
.
.
6/7/8/9
```

Suppose the sequence characters are as in examples 7 and 8. Then the file TERN is printed as TERN03F.

SAVEPF (CATALOG PERMANENT FILE ON LINKED MAINFRAME)

SAVEPF makes an existing local file a permanent file on the mainframe specified. SAVEPF differs from the CATALOG control statement in that SAVEPF can catalog a file at a mainframe other than that where the job is executing; CATALOG cannot.

The format of SAVEPF is:

SAVEPF,lfn,pfn,ID=name,AC=act,CN=cn,CY=cy,EX=ex,FO=fo,MD=md,MR=m,PW=pw,RD=rd,RP=rp,
RW=p,ST=mmf,TK=tk,XR=xr.

The lfn and ID parameters are required in the order shown. All other parameters are order independent. The ST parameter is required; other parameters might be required, as noted with CATALOG. Any SN parameter is ignored. If a terminator does not appear at the end of the parameter list, column 1 of the next card or line is considered to be a continuation of the SAVEPF parameter list.

lfn	Logical file name by which the file is presently known to the job, 1-7 letters or digits beginning with a letter. This name does not become part of the permanent file identification.
pfn	Permanent file name by which the file is known in permanent file manager tables, 1-40 letters or digits. If pfn is omitted, lfn is used.
ID=name	Owner or creator of file.
ST=mmf	System on which file is to be cataloged, 3 characters. The values for mmf are established at installation time.

When the ST parameter designates a mainframe running SCOPE 2, the file structure must adhere to SCOPE 2 Record Manager defaults; otherwise a FILE statement must be used. For example, the SCOPE 2 FORTRAN and COBOL compilers expect the source program to be in W type record format. A program created under the NOS/BE INTERCOM Editor consists of Z type records and cannot be compiled directly by SCOPE 2 compilers.

Example:

A user writes a program under the Editor CREATE command and makes the file local to the job with a SAVE,ZZZ command. The user than enters the following statement to make the file permanent under SCOPE 2: SAVEPF,ZZZ,ID=XX,ST=MFZ., where MFZ is the mainframe running SCOPE 2. The system responds with WAITING FOR MMF SAVEPF. This message appears even if the SCOPE 2 mainframe is down or not available. When INTERCOM responds with .., the file has been transferred and made permanent.

To compile and execute the program made permanent on SCOPE 2, the user creates the following file under the Editor CREATE command.

```
SCOPE 2 job statement.
SCOPE 2 account statement.
FILE,ZZZ,RT=Z,BT=C,FL=80.
ATTACH,ZZZ,ID=XX.
FTN,I=ZZZ.
LGO.
```

With the SAVE and BATCH commands, the user makes the file local and then submits the job. The program on file ZZZ is attached, compiled, and executed. The job aborts if the FILE statement is not included, since the FORTRAN compiler would expect W type records.

See the CYBER Record Manager Reference Manual and the NOS/BE Station Operator's Guide /Reference Manual for additional details on file conversion requirements.

See the CATALOG control statement for the remaining parameters.

SETNAME (ESTABLISH IMPLICIT SETNAME)

SETNAME indicates the device set to be referenced implicitly by subsequent ATTACH, PURGE, and REQUEST control statements. When SETNAME is not used, these control statements implicitly reference a system device set.

The format of SETNAME is:

SETNAME,setname.

The parameter can be omitted.

setname — Name of device set to be referenced implicitly, 1-7 letters or digits beginning with a letter. If omitted, public device sets are assumed.

A second SETNAME control statement overrides the first.

SETNAME is explicitly overridden by an SN=setname parameter on a REQUEST, ATTACH, or PURGE control statement. An SN that does not specify a setname on a REQUEST control statement does not override the SETNAME control statement. A rotating mass storage REQUEST which does not have an SN parameter will always reference public device sets.

SKIPB (SKIP BACKWARD SYSTEM-LOGICAL-RECORDS)

SKIPB bypasses one or more system-logical-records in a reverse direction. Current file position can be any point within a record when the control statement is executed. The file must have system-logical-record structure. SKIPB cannot be used with tapes in S or L format and should not be used with CYBER Record Manager file organizations unless RT=S.

The format of SKIPB is:

SKIPB,lfn,n,lev,mode.

Parameters are positional; only lfn is required.

lfn — Logical file name, 1-7 letters or digits beginning with a letter.

n — Number of system-logical-records of level lev or greater to be skipped, 1-262142 (decimal). Default is 1. A value greater than 262142 is treated as a rewind request.

lev — Level number, 0-17 (octal). Default is 0.

mode — File mode applicable to tape files only:

- B — Binary. Default.
- C — Coded

Skipping stops when the specified number of terminators containing the specified level have been bypassed or beginning-of-information is encountered. At the end of SKIPB, the file is positioned immediately following the system-logical-record terminator examined last. When the file is positioned immediately following a system-logical-record terminator, that terminator is not counted in the execution of n skips.

SKIPF (SKIP FORWARD SYSTEM-LOGICAL-RECORDS)

SKIPF bypasses one or more system-logical-records in a forward direction. Current file position can be any point within a record when the control statement is issued. The file must have system-logical-record structure. SKIPF cannot be used with tapes in S or L format and should not be used with CYBER Record Manager file organizations unless RT=S.

The format of SKIPF is:

SKIPF,lfn,n,lev,mode.

Parameters are positional; only lfn is required.

lfn — Logical file name, 1-7 letters or digits beginning with a letter.

n — Number of system-logical-records of level lev or greater to be skipped, 1-262142 (decimal). Default is 1.

lev — Level number, 0-17 (octal). Default is 0.

mode — File mode applicable to tape files only:

- B — Binary. Default.
- C — Coded

Skipping stops when the specified number of terminators containing the specified level have been bypassed or end-of-information is reached. At the end of SKIPF, the file is positioned immediately following the system-logical-record last examined.

A value greater than 262142 for the number of records to be skipped causes a rotating mass storage file to be positioned at end-of-information. For a tape file, a similar parameter causes the file to remain at its current position.

SUMMARY (ACCOUNT SUMMARY)

SUMMARY obtains an accounting summary up to the point in the job where the statement is encountered. The accounting summary, which appears in the job dayfile, lists resources used to this point in the job. The resources used by a job step can be determined by executing a SUMMARY statement before and after the job step and subtracting the resulting values. The summary output is the same as the accounting summary generated at end-of-job.

The format of SUMMARY is:

```
SUMMARY.
```

The discussion of the dayfile in section 2 gives details of summary output.

SWITCH (SET SOFTWARE SWITCH)

SWITCH sets one of the six software switches available for each job. At the start of job execution, all switches are zero. Execution of SWITCH changes the current setting to its opposite mode.

In program branching, where two alternate processing routes are provided, the software sense switch is frequently used to determine the path taken. This switch is a bit in central memory that a user's program can reference. A program might contain a request to take one path if the bit is set to one (on) and another if it is zero (off).

The format of SWITCH is:

```
SWITCH,n.
```

n	Number of switch to be changed, 1-6. The n parameter must be specified; there is no default.

Switches also can be set by the central site operator, a terminal user, or a program in a language that supports switch operations.

The following example changes switch 4 to ON, then OFF, then ON again.

```
SWITCH,4.    Set switch to 1.
SWITCH,4.    Resets switch to 0.
SWITCH,4.    Resets switch to 1.
```

SYSBULL (ACCESS SYSTEM BULLETIN)

SYSBULL copies requested system bulletins to the OUTPUT file.

The format of SYSBULL is:

```
SYSBULL,p1,p2, . . . ,pn.
```

Parameters are all optional.

pi Bulletin names, ALL, or INDEX:

ALL	Lists all bulletins. Any other parameters are ignored.
INDEX	Lists index of all bulletins available. Default.

INTERCOM makes a call to SYSBULL whenever a user logs in. The calls are:

SYSBULL(LOGIN)	If SUP is not specified.
SYSBULL(SUP)	If SUP is specified.

SYSBULL automatically attempts to find the bulletin named LOGIN or SUP. If found, the bulletin is immediately displayed. If SYSBULL does not find the system bulletin permanent file or the specific bulletin LOGIN or SUP, processing continues.

The operating system calls SYSBULL for each batch job entered in the system.

The call is:

SYSBULL(BATCH)

SYSBULL automatically attempts to find the bulletin named BATCH. If found, it is the first item printed on OUTPUT. If SYSBULL does not find the system bulletin permanent file or the specific bulletin BATCH, processing continues.

TRANSF (DECREMENT DEPENDENCY COUNT)

TRANSF decrements the dependency count for jobs in an interdependent job string. The user can submit a string of interdependent jobs to the computer, specifying the order in which they are to be executed. In such a string, jobs can be input in any order and from central site or remote card readers. A job is not executed until all prerequisite jobs in the string have been executed. Whenever possible, the operating system schedules interdependent jobs for execution in parallel (multiprogramming).

As each job is input, dependency identifier and dependency count on the job statement are noted. The dependency count is decremented by TRANSF control statements in prerequisite jobs. When the count of a dependent job becomes zero, it executes.

The Dym parameter on the job statement establishes job interdependency. y is the dependency identifier that names the string to which the job belongs. m is the dependency count (number) of prerequisite jobs on which the job depends.

TRANSF must appear after the control statements that execute the prerequisite programs. In multi-mainframe configurations, a string of interdependent jobs must execute on the same mainframe. TRANSF should not appear in the last job in the string since no jobs can depend on it.

The format of TRANSF is:

TRANSF,job1,job2, ...

Multiple job names or multiple TRANSF control statements can be used.

job — Name of job whose dependency count is to be decremented. Only the first five characters of each job name are used, with the dependency string identifier maintaining proper identification.

If a job containing a TRANSF control statement is terminated before that control statement is processed, the dependency count of other jobs is not decreased. Instead, all succeeding jobs that depend on this job remain in the input queue. No error message indicates that a job in a dependent string has terminated abnormally: operator alertness is needed to know the remaining jobs should be evicted or forced into execution. A message instructing the operator can be placed in a routine after a RECOVR function, or on a PAUSE statement following an EXIT statement.

An example of an interdependent job string JS follows. Consider jobs with names JOBA through JOBF:

JOBB is dependent on successful execution of JOBA
JOBC on JOBA
JOBD on JOBB and JOBC
JOBE on JOBC
JOBF on JOBB, JOBD, and JOBE

The control statements should appear with:

```
JOBA,DJS00.                    JOBB,DJS01.
execution call                 execution call
TRANSF(JOBB,JOBC)              TRANSF(JOBD,JOBF)
.                              .
.                              .
.                              .
7/8/9                          7/8/9

JOBC,DJS01.                    JOBD,DJS02.
execution call                 execution call
TRANSF(JOBD,JOBE)              TRANSF(JOBF)
7/8/9                          7/8/9

JOBE,DJS01.                    JOBF,DJS03.
execution call                 execution call
TRANSF(JOBF)                   7/8/9
7/8/9
```

JOBF, which can execute only if all other jobs in the string are successful, has a dependency count of 3, the number of jobs containing TRANSF references to JOBF.

TRANSPF (TRANSFER PERMANENT FILE)

TRANSPF changes the residence of permanent files and permanent file tables within a device set so that all permanent file information can be removed from a device. It also copies files from one device set to another. These operations are known as a single device set transfer and a dual device set transfer, respectively.

Before TRANSPF can be executed, a permanent file with name of DUM and ID of PUBLIC must be cataloged on the device set specified by the FS parameter. If this is not done, TRANSPF aborts. TRANSPF issues an internal ATTACH of the permanent file DUM; the passwords submitted in this ATTACH are those submitted via the PW parameter on the TRANSPF request. If TRANSPF is unable to attach the permanent file DUM, the function aborts.

Before TRANSPF is called, a MOUNT control statement must be executed for the master devices of the device sets specified by the FS and TS parameters. TRANSPF cannot be run on a shared device set.

The format of TRANSPF is:

TRANSPF,PW=pw,FS=setname1,TS=setname2,FM=vsn1,TM=vsn2,LF=lfn.

Parameters FS and TS are required; PW is required if passwords have been defined for file DUM. Remaining parameters are optional. All parameters are order independent. If a terminator does not appear at the end of the parameter list, column 1 of the next card or line is considered to be a continuation of the TRANSPF parameter list.

PW=pw — Specifies read, control, and modify passwords if defined for permanent file DUM. The turnkey password is assembled into the TRANSPF utility and need not be specified. If passwords have been defined for file DUM, at least one must be specified with this parameter or the utility aborts. No default exists.

FS=setname1 — Name of device set from which permanent file information is to be transferred; 1-7 letters or digits beginning with a letter. Default is the permanent file default set.

TS=setname2 — Name of device set to which permanent file information is to be transferred; 1-7 letters or digits beginning with a letter. Default is the permanent file default set.

FM=vsn1 — Volume serial number of member device from which permanent file information is to be transferred; 1-6 letters or digits with leading zeros assumed. Required when TS and FS specify the same setname. When TS and FS specify different setnames, all devices in the set are assumed and the FM parameter cannot be specified.

TM=vsn2 — Volume serial number of member device to which permanent file information is to be transferred; 1-6 letters or digits with leading zeros assumed. Data that cannot be contained on this device overflows to another member of device set specified by TS, except that files do not overflow to the member specified by FM when TS and FS specify the same setname. Required when TS and FS specify the same setname and FM specifies a master device. When TS and FS specify different setnames, TM cannot be specified. Default is all devices in device set specified by the TS parameter.

LF=lfn — Name of file on which output listing is written; 1-7 letters or digits beginning with a letter. Default is OUTPUT.

SINGLE DEVICE SET TRANSPF

A single device set TRANSPF is requested if the device set specified by the FS parameter is the same as the device set specified by the TS parameter.

TRANSFERRING FROM A MEMBER

If the FM parameter does not specify a master device, permanent files residing on the FM device are moved to the TM device. A file is moved if any part resides on the FM device. Once the file has been transferred, the disk space associated with the old copy is released. If the file cannot be completely contained on the TM device, the file overflows to any other device in the set except the FM device. If the transfer of a file is unsuccessful, that file is skipped, but TRANSPF is not aborted. A file transfer can be unsuccessful because of uncorrectable parity errors, not enough space in the device set to accommodate two copies of the file simultaneously, or permanent file catalog full. When all permanent file information is successfully transferred from the FM device, that device is no longer a permanent file device.

TRANSFERRING FROM A MASTER

When the FM parameter specifies a master device, the device set tables are moved to the device specified by TM, and the device labels for both devices are updated to reflect the new organization of the device set. If the tables cannot be successfully moved, the device set is not changed by the TRANSPF utility. Table transfers can fail because of uncorrectable parity errors, or not enough space on the TM device to completely contain the disk tables. The system must be idle before TRANSPF is executed for table transfer.

After the master device is successfullly changed, permanent files residing on the FM device are moved to the TM device as described above. When all permanent file information is transferred from the FM device, that device is no longer a permanent file device.

Examples of single device set transfer are:

1. FIRST.
   MOUNT(SN=TEST,VSN=999) Mount master.
   TRANSPF(FS=TEST,TS=TEST,FM=999,TM=111,PW=A)
   6/7/8/9

   This job transfers all permanent files and permanent file tables from the master device with VSN of 999 to the member device with VSN of 111. Both devices belong to device set TEST. Note that the member device with VSN=111 was not explicitly mounted. The system initiates the mount of the member when actual I/O is requested by TRANSPF. If this job runs successfully, device 111 is the master device of set TEST.

   If the tables do not fit on the device with VSN=111, the set is not changed, and the job ends. If the tables are successfully transferred, but the permanent files do not fit on the device with VSN=111, the files overflow to any devices in the set TEST except the device with VSN=999.

   The permanent file DUM is assumed to have been previously cataloged with password A on device set TEST.

2. SECOND.
   MOUNT(SN=TEST1,VSN=555) Mount master.
   TRANSPF(FS=TEST1,TS=TEST1,FM=888,TM=222,PW=Q)
   6/7/8/9

   This job transfers all permanent files from the member device with VSN=888 to the member device with VSN=222. Both members belong to the device set TEST1. Note that the members with VSNs of 888 and 222 were not explicitly mounted. The system initiates the mount of these members when actual I/O is requested by TRANSPF.

DUAL DEVICE SET TRANSPF

A dual device set TRANSPF is requested if the device set specified by the FS parameter is different from the device set specified by the TS parameter. TRANSPF transfers permanent files by simulating the following sequence of control statements:

```
REQUEST(lfn2,SN=setname2)
ATTACH(lfn1,pfn,ID=owner,SN=setname1)
COPY(lfn1,lfn2)
CATALOG(lfn2,pfn,ID=owner)
RETURN(lfn1,lfn2)
```

All files residing on the device set specified by the FS parameter are transferred to the device set specified by the TS parameter. The FM and TM parameters cannot be used; no member devices can be specified. After a successful transfer of a file, two copies of the file exist, one in the FS device set and one in the TS device set.

A permanent file transfer might be unsuccessful if lfn has an uncorrectable parity error, CATALOG is unsuccessful for reasons such as unavailable table space, or if not enough disk space is available on the TS device set to contain the file.

In a dual device set transfer, the disk tables are not moved as a separate entity; critical tables are only moved within a device set and never from one device set to another.

Example of dual device set transfer:

```
JOB.
MOUNT(SN=BOB,VSN=1944)                Mount master.
MOUNT(SN=TOM,VSN=1984)                Mount master.
TRANSPF(FS=BOB,TS=TOM,PW=YES)
6/7/8/9
```

This job moves permanent files from the device set BOB to the device set TOM.

UNLOAD (EVICT FILE)

UNLOAD performs an operating system CLOSE/UNLOAD function. It differs from RETURN only in that RETURN reduces the maximum number of tapes that can be held by the job, but UNLOAD does not affect the tape count. UNLOAD deletes all references to the files specified, except as noted below.

The format of UNLOAD is:

UNLOAD,lfn1,lfn2, ...

More than one file or multi-file set can be specified; only one is required.

lfn — Name of file to be unloaded, 1-7 letters or digits beginning with a letter. Can be a member of a tape multi-file set; cannot be INPUT.

Name of multi-file set of tape to be unloaded, 1-6 letters or digits beginning with a letter.

For tape files, tapes are rewound and unloaded after any necessary labels are written. The tape drive is then made available for new assignment. However, UNLOAD cannot override an IU (inhibit unload) parameter on the REQUEST control statement for the file. When the IU parameter exists, a subsequent unload rewinds, but does not unload, the tape.

For mass storage files, UNLOAD causes the file to be returned. Special-named files on queue devices are released to the output queue associated wtih their disposition; if any of the special-named files is to be evicted, the DISPOSE or ROUTE control statement should be used rather than UNLOAD. Permanent files return to permanent file manager jurisdiction. Other mass storage files are evicted.

VSN (TAPE VOLUME IDENTIFICATION)

VSN has two functions for tape files:

It relates the external sticker (volume serial number) for a tape to the logical file name.

It provides information for the tape prescheduling display at the operator console. Since the operator is then aware of upcoming tape requests, he can mount the required tapes so the system can access them without further operator action.

The VSN control statement can be used in place of a VSN parameter on a REQUEST or LABEL control statement. VSN execution does not affect either the checking or writing of tape labels. It can be specified for labeled or unlabeled tapes.

The format of VSN is:

VSN,lfn1=vsn1,lfn2=vsn2, ...

One statement can be used for any number of files. Multiple VSN control statements can be used. VSN can be continued: if the parameter list does not end with a terminator, column 1 of the next control statement is considered a continuation of column 80.

lfn — For a single file, the logical file name of 1-7 letters or digits beginning with a letter.

For a multi-file set, the multi-file set name of 1-6 letters or digits beginning with a letter.

vsn — Volume serial number of 1-6 letters or digits with leading zeros assumed. A vsn of 0 or SCRATCH, or omission of =vsn, results in scratch tape assignment.

If any of several alternate volumes suffice, equals signs should separate identifiers, as in: FILE=1234=1235.

If the file is to be assigned to a multi-volume set, VSNs should appear, separated by slashes, in the order that volumes are to be accessed as in: BIGFILE=1ST/2ND/.../LAST.

If conflicting volume serial numbers are given for a single tape file, the first encountered is used. However, duplicate specifications on the same control statement produce a fatal error.

VSN statements can be placed anywhere in the control statements as long as they precede the REQUEST or LABEL control statement that associates the file with the job. If a logical file name is to be re-used during a job, such as OLDPL for two UPDATE operations, the first file should be released by an UNLOAD or RETURN control statement before a VSN is given for the second file.

Examples of VSN use:

1.
```
JOB5,MT1.
VSN(TAPE1=1234)
REWIND,TAPE1.
```

The VSN control statement has no effect because no REQUEST or LABEL control statement appears for file TAPE1. File TAPE1 is opened as a disk file.

2.
```
JOB6,MT1.
VSN(TAPE1=1234)
REQUEST(TAPE1,MT,E,NORING)
```

```
JOB7,MT1.
REQUEST(TAPE1,VSN=1234,MT,E,NORING)
```

To have a specific magnetic tape assigned to the job, either of the above requests would suffice.

CYBER CONTROL LANGUAGE (CCL) 5

INTRODUCTION

The CYBER Control Language (CCL) provides control statement manipulation. Various verbs allow selection of CCL capabilities. The user may employ CCL to conditionally skip or process control statements and to process and reprocess a group of control statements. Other CCL verbs control processing of control statements in a file other than the job file. A CCL verb appears at the beginning of a CCL control statement, preceding any separators or terminators; a verb is only part of a CCL statement.

The following verbs cause control statements (including CCL control statements) to be skipped or processed conditionally.

IFE	If the IFE expression (for example, A=B) is true, processes following statements; if it is not true, skips until a terminating statement is encountered
SKIP	Skips until a terminating statement is found
ELSE	Terminates and initiates IFE skipping
ENDIF	Terminates IFE, SKIP, and ELSE skipping

The following pair of CCL verbs allows the user to process and reprocess a group of control statements (including CCL statements).

WHILE	Brackets a group of control statements and establishes the beginning of the group [The group is processed as long as the WHILE expression (for example, R1<5) is true.]
ENDW	Brackets a group of control statements and establishes the end of the group

The user can manipulate CCL symbolic names. A symbolic name is an alphanumeric character string recognized by CCL. CCL associates a constant or variable numeric value with a symbolic name.

SET	Allows the user to alter the values of variable symbolic names
DISPLAY	Evaluates an expression (for example, 2 + 3) and prints the result in the dayfile in both octal and decimal

CCL also provides the following functions to be used within an expression.

FILE	Determines the attributes of a file
DT	Determines the type of device on which a file resides
NUM	Determines if a parameter has a numeric value

CCL provides verbs and commands to control processing of control statements (including CCL statements) in a file other than the original job file. A group of these separate control statements is called a procedure. The following CCL verbs deal with procedures.

BEGIN	Initiates processing of a procedure
REVERT	Returns processing from a procedure to the control statement sequence that called it

A procedure is identified to CCL and to the NOS/BE operating system by the following header statement.

.PROC	Identifies the statements that follow as a procedure

A command is similar to a directive, controlling processing of data within a procedure. Commands are as follows:

. DATA	Allows data needed by a procedure to be stored within that procedure
. EOR	Causes an end of record to be written on a data file
. EOF	Causes an end of partition to be written on a data file
. *	Allows the user to include comments in a procedure; these comments are not printed in the dayfile

CCL statements have a syntax similar to, but not identical to, the operating system's control statement syntax. The asterisk (*) and all system separators are valid CCL separators. The separator following the verb in a CCL statement must be a comma or a left parenthesis. A CCL statement must be terminated by a period or a right parenthesis. The separator between parameters must be a comma. Literals ($-delimited character strings) are recognized within all CCL statements but might not be evaluated during substitution within a procedure (covered later in this section). CCL ignores all blanks in CCL statements, except blanks within a literal. Blanks cannot be used within a verb. Any CCL statement may continue over more than one card or line if the last character of a continued card or line is a valid CCL separator.

EXPRESSIONS

A CCL expression consists of operators and operands. It may include other expressions that are enclosed in parentheses. For example, the expression 2*(3+5) contains another expression, 3+5.

An operator can be arithmetic, relational, or logical. Parentheses are treated as operators; however, they do not imply multiplication.

An operand can be any one of the following.

Integer constant	A character string of 1 to 10 characters. An integer constant can be either numeric or a literal
Symbolic name	An alphanumeric character string of 1 to 10 characters that is recognized by CCL (A symbolic name has a numeric value, which is either an installation-defined constant or a user- or CCL-defined variable.)

CCL function	A special operand recognized by CCL; determines attributes of a file or a parameter (An expression may consist entirely of a CCL function.)
Expression	A CCL expression enclosed in parentheses; expression is evaluated, and the result becomes an operand

For all practical purposes, an expression may be as long as the user wishes, provided there is either a) or a . within the first 50 operands.

CCL assumes any character string beginning with a numeric character is numeric. It cannot contain any nonnumeric characters, except for an option post radix of B or D. Any alphanumeric string must begin with an alphabetic character.

Expressions may be used with CCL control statements IFE, WHILE, DISPLAY, and SET and the FILE function. The separator preceding an expression is not part of the expression; using a comma for this separator, rather than a left parenthesis, improves readability. The separator following the expression must be a comma. Computations are accurate to 10 decimals digits (20 octal digits), and overflow is ignored.

OPERATORS

The three types of CCL operators are arithmetic, relational, and logical. Arithmetic operators perform the various arithmetic operations. The relational operators produce a value of one if the relationship is true and zero if it is false. Logical operators evaluate the full 60 bits of each operand, producing a 60-bit result. If a CCL statement or relational operator evaluates this result as true or false, CCL considers a result with any bits set to be nonzero (true) and a result with no bits set to be zero (false). Any operand may be preceded by a leading plus or minus. A leading plus is ignored, and a leading minus indicates a negation.

ARITHMETIC

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

RELATIONAL

.EQ. or =	Equal
.LT. or <	Less than
.GT. or >	Greater than
.NE.	Not equal
.LE.	Less than or equal
.GE.	Greater than or equal

LOGICAL

.EQV.	Equivalence (If A.EQV.B is true, all 60 bits are set.)
.OR.	Inclusive OR (Any bit set in either A or B is set in the result.)
.AND.	AND (If A and B are true, A.AND.B is true.)
.XOR.	Exclusive OR (A bit is set in the result if the corresponding bit is set in either A or B but not both.)
.NOT.	Complement (If A is 0 or no bits are set, NOT A is all 60 bits set.)

ORDER OF EVALUATION

Exponentiation

Multiplication, division

Addition, subtraction, negation

Relations

Complement

AND

Inclusive OR

Exclusive OR, equivalence

INTEGER CONSTANTS

An integer constant is generally numeric, although it may also be a literal. It must be 10 characters or less, including the post radix if it exists. Integer constants may be specified with or without a post radix. If the post radix is omitted, the constant is assumed to be decimal. A post radix of B indicates octal, and a post radix of D indicates decimal.

A literal (a $-delimited character string) must be 10 characters or less, excluding the $ delimiters. If CCL encounters a literal, it is evaluated, and its display code value is right-justified and processed as an integer constant.

SYMBOLIC NAMES

A symbolic name is an alphanumeric character string that points to a location where a numeric value is stored. These numeric values may be defined at installation or may be variables set by the user or by CCL. All variables, except EM, OT, and SYS, have an initial value of zero.

The symbolic names in table 5-1 and the symbolic names with true or false values are valid in any CCL expression but meaningless within FILE or DT functions. FILE and DT functions have their own symbolic names.

Each symbolic name in table 5-1 has one or more of the following attributes.

Local

An X in this column indicates that the value is saved by a BEGIN statement before initiating a procedure and restored by a REVERT statement upon termination of a procedure (procedures are explained later in this section).

Set

An entry in this column indicates that the symbol has a variable value. The entry specifies how the symbol obtains its value. One or more of the following characters may be listed in this column for each symbolic name.

- B Set by BEGIN
- O Set by the operating system
- R Set by REVERT
- U Set by a user on the SET control statement or the SETJCI macro

Compare

An entry in this column is a symbolic name. The symbolic name with a compare entry is a subset of that entry name and may be compared to it. For example, CPE has the entry EF in the compare column. To find out if an error was a CPU abort, the value of EF could be compared to the value of CPE using a CCL statement, such as IFE (covered later in this section). For example, if IFE, EF=CPE, LS. is true, the error was a CPU abort.

TABLE 5-1. SYMBOLIC NAMES WITH ARITHMETIC VALUES

Name	Local	Set	Compare	User's Range of Values	Description
R1	X	U		0 to 131071D or 377777B	Value in control register 1.
R2	X	U		0 to 131071D or 377777B	Value in control register 2.
R3	X	U		0 to 131071D or 377777B	Value in control register 3.
R1G		U		0 to 131071D or 377777B	Value in global control register 1.
DSC	X	U,O		0 and 1	Dayfile skipped control statement flag.
EF	X	U,O		0 to 62D or 76B	Error flag.
EFG		U,O		0 to 62D or 76B	Global error flag.
TLE			EF		Time limit error; time limit was exceeded.
ARE			EF		Arithmetic error; user had a floating-point arithmetic error or a read or write outside of SCM or LCM range.
PPE			EF		PPU abort; a peripheral processor program aborted.
CPE			EF		CPU abort; job supervisor detected an error, or the system processed an ABORT macro and found no REPRIEVE macro.
MNE			EF		Monitor call error; a user's RA+1 request contained an error.
ODE			EF		Operator drop; a drop aborted the job.
PSE			EF		Program stop error; a program stop (zero instruction word) occurred in the user's program.
ESE			EF		EXIT,S. processing; an ABORT macro with S option initiated a search for an EXIT,S. statement.
MSE			EF		Mass storage limit; the limit was exceeded.
OT		O			Job origin type.

TABLE 5-1. SYMBOLIC NAMES WITH ARITHMETIC VALUES (Contd)

Name	Local	Set	Compare	User's Range of Values	Description
SYO			OT		System job; the job originated from the system console.
BCO			OT		Batch job; the job was submitted at the central site or from a terminal. Job output is sent to the central site.
EIO			OT		Remote batch job; the job was submitted from a terminal and output is sent to a terminal.
TXO			OT		Time-sharing job; a statement was submitted from a terminal through INTERCOM.
SYS		O			The host operating system.
NOSB			SYS		NOS/BE operating system.
SC2			SYS		SCOPE 2 operating system.
PNL		B,R			Procedure nesting level: 0 Job control statements. 1 1st level procedure. . . . 50 50th level procedure.
EM		U		0 to 7B	Current exit mode; user sets on the MODE statement.
MFL		O			Maximum field length.
MFLL		O			Maximum ECS field length.
TIME		O			Current time of day (hhmm).
VER		O			Version of the operating system.

The following symbolic names have true or false values.

TRUE	1
T	TRUE=1
FALSE	0
F	FALSE=0
SWn	SENSE switch, n=1-6

The symbolic names R1, R2, R3, and R1G are variables the user can set equal to an expression. The expression must consist of symbolic names or integer constants. CCL evaluates the expression as a numeric value (for example, R1 + 1 is a valid expression).

DSC is also a variable, but both the user and CCL control its value. Unless the user specifies the value of DSC to be one, DSC remains zero and skipped statements are not printed in the dayfile. When DSC is set to one, any skipped statements that follow are printed in the dayfile. Some CCL error processing changes DSC to one, forcing skipped statements to be printed. Skipped statements have a .CCL.. prefix.

The system and the user may set the values of EF and EFG. As the system encounters errors, CCL changes the value of EF to the numeric value of the error type. After processing a procedure (covered later in this section) CCL sets the value of EFG to the procedure's value of EF, unless the value of EFG is already nonzero. CCL's range of values for EF and EFG are the numeric values of symbolic names TLE, ARE, PPE, CPE, MNE, ODE, PSE, ESE, and MSE; their values are system defined.

CCL controls the values of OT and SYS. The range of values for OT are the installation-defined, numeric values of SYO, BCO, EIO, and TXO. The range of values for SYS consists of the numeric values of NOSB and SC2. SC2 and NOSB are defined at installation time.

If the user-defined numeric value of a symbolic name exceeds the given range of values, CCL truncates the value retaining the sign, if signed, without issuing an error message.

In the CYBER 170 series, EM is a 4-digit octal value, rather than a single-digit octal value. To reduce the value of EM to the single-digit set by the MODE statement, use the expression EM.AND.7. To ensure correct evaluation, enclose this expression in parentheses.

CONDITIONAL STATEMENTS

The following conditional control statements bracket groups of other control statements to be conditionally processed or skipped.

IFE

SKIP

ELSE

ENDIF

IFE, SKIP, and ELSE statements precede a group of control statements. IFE and ELSE conditionally skip or process the statements. (SKIP always causes skipping.) An ELSE or ENDIF statement follows the group of statements, terminating skipping initiated by an IFE,SKIP, or ELSE statement.

All conditional statements require a label string parameter. The label string consists of 1 to 10 alphanumeric characters, beginning with an alphabetic character. When an IFE, SKIP, or ELSE statement (with a label string) initiates skipping, skipping continues until CCL encounters a terminating statement (ELSE, ENDIF) with a matching label string.

If no such terminating statement is found while skipping within the job control statement record, CCL skips all remaining statements and the job ends. If no such terminating statement is found while skipping within a called procedure (covered later in this section), CCL skips all remaining statements in the called procedure, issues an abort, and continues processing with the job or calling procedure.

NOTE

If the job's time limit is exceeded while CCL is skipping, the job aborts and the position of the job control statement file is undefined. CCL stops skipping, and the system begins searching for an EXIT statement. Results may be altered; the user should increase the time limit and resubmit the job.

By default, skipped control statements are not written on the dayfile. The SET statement can change this default, allowing skipped statements to appear in the dayfile.

IFE

The IFE statement conditionally causes the skipping of a group of succeeding control statements. If the expression is true, the statements which follow are processed; if not, the statements are skipped according to the label string. ELSE or ENDIF terminates the skip. The separator following the expression must be a comma.

The format of IFE is:

IFE,exp, ls.

Both parameters are positional and required.

exp	CCL expression. Character strings must be integer constants, symbolic names, or CCL functions.
ls	Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

Example:

```
IFE,R1.GT.7,JUMP.
COMMENT.1
COMMENT.2
ENDIF,JUMP.
REWIND,FILE1.
```

If the value in control register 1 (R1) is greater than 7, the comments are processed. If not, CCL skips to the ENDIF,JUMP. statement. Whether the value is greater than 7 or not, the system rewinds FILE1.

SKIP

The SKIP control statement causes unconditional skipping of the control statements that follow. Skipping is terminated by an ENDIF statement.

The format of SKIP is:

SKIP,ls.

The parameter is required.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

Example:

```
SKIP,HALT.          SKIP initiates skipping and all the following statements are ignored until
    .               ENDIF.
    .
    .
ENDIF,HALT.
```

ELSE

The ELSE statement can either terminate or initiate skipping. It terminates skipping when used in conjunction with IFE, provided that the label strings match. If the IFE statement does not initiate skipping (the expression within the statement is true), ELSE initiates skipping and is terminated by an ENDIF statement with a matching label string. ELSE does not terminate skipping initiated by a SKIP statement or another ELSE statement.

The format of ELSE is:

ELSE,ls.

The parameter is required.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

Example:

```
IFE,DSC=0,GO.
COMMENT.THIS IS NOT SKIPPED.
ELSE,GO.
COMMENT.THIS IS SKIPPED.
ENDIF,GO.
```

Since the value of the dayfile skipped statement flag (DSC) is 0 by default, the IFE expression is true; no statements are skipped. When the ELSE,GO. statement is processed, ELSE initiates skipping. The ENDIF,GO. statement terminates the skipping.

ENDIF

The ENDIF statement terminates skipping when used in conjunction with IFE, ELSE, or SKIP statements. If not terminating a skip, ENDIF is ignored in terms of processing. It is, however, printed in the dayfile. The label strings of the statement that initiates skipping (IFE,ELSE, or SKIP) and the ENDIF statement must match. If CCL encounters an ENDIF statements with an unmatched label string, CCL ignores it.

The format of ENDIF is:

ENDIF,ls.

The parameter is required.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

Example:

```
IFE,R2.LE.6,LS1.
SET,R2=R2+1.
ENDIF,LS1.
DISPLAY,R2.
```

If the value in control register 2 (R2) is less than or equal to 6, the SET statement increases R2 by 1. If the value of R2 is greater than 6, IFE skips the SET statement, and ENDIF terminates that skip. The DISPLAY statement prints the value of R2 in the user dayfile, whether the IFE expression is true or not. (The SET and DISPLAY statements are covered later in this section.)

ITERATIVE STATEMENTS

CCL provides an iterative capacity. The WHILE and ENDW control statements bracket a group of control statements and cause it to be processed and reprocessed as long as the WHILE expression is true (can be zero times if the expression is never true). When the WHILE expression is no longer true, CCL processes the WHILE statement, evaluates the expression as false, and skips all statements until it finds the ENDW statement.

The WHILE and ENDW statements require a label string parameter. The label string consists of 1 to 10 alphanumeric characters, beginning with an alphabetic character. It identifies the group of statements to be conditionally reprocessed.

The format of WHILE is:

WHILE,exp,ls.

Both parameters are positional and required. The separator following exp must be a comma.

exp A CCL expression . Character strings must be integer constants, symbolic names, or CCL functions.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

The format of ENDW is:

ENDW,ls.

The parameter is required.

ls Label string; must match the label string on a WHILE statement.

The following rules apply to bracketing.

1. An ENDW statement brackets only that WHILE statement which has a matching label string.
2. When an expression is false and an ENDW with a matching label string is not found, a WHILE statement skips all remaining control statements in the control statement record.
3. The label string of a WHILE statement must be unique among all WHILE statements within the same procedure (procedures are explained later in this section). Label strings of WHILE statements within the job control statement record must be unique among all WHILE statements in the job.

Example:

```
SET,R1=1.
WHILE,R1<5,FROG.
FTN,I=TOAD.
LGO.
REWIND,LGO.
SET,R1=R1+1.
ENDW,FROG.
```

The value of control register 1 (R1) is set to 1. The FTN compiler takes input from file TOAD and executes different FORTRAN jobs as long as the value of R1 is less than 5 (four times). Each pass through the loop increases the value of R1 by 1.

ADDITIONAL CCL STATEMENTS

The following CCL statements affect the values of symbolic names. The DISPLAY statement, however, only prints an evaluated expression in the user dayfile, and that expression need not contain a symbolic name.

DISPLAY

The DISPLAY control statement evaluates an expression and sends the result to the user dayfile (but not the system dayfile) in both decimal and octal format. The largest decimal value which may be displayed is 10 digits. If the value is larger than 10 digits, GT followed by 9999999999 is displayed. If the value is negative and larger than 10 digits, LT followed by a minus and 9999999999 is displayed. In octal code, numbers as large as 20 digits can be displayed.

The format of DISPLAY is:

DISPLAY,exp.

The parameter is positional and required.

exp A CCL expression. Character strings must be integer constants, symbolic names, or CCL functions.

Examples:

1. DISPLAY, 1111111B*10000B.

   produces

   1227132928 11111110000B

2. DISPLAY,SYS.

   produces

   4 4B

3. DISPLAY,2**37.

   produces

   GT 9999999999 2000000000000B

4. DISPLAY,-2**37.

   produces

   LT -9999999999 -2000000000000B

SET

The SET statement allows the user to set the value of symbolic names. Only a subset of the symbolic names known to CCL may be set. This subset consists of the control registers (R1, R2, R3, and R1G), the error flags (EF and EFG), and the dayfile skipped control statement flag (DSC). The control registers are 18-bit signed quantities, and the error flags are 6-bit unsigned quantities. If the value of the expression is too large, it is truncated (retaining the sign if signed), and no error message is issued. DSC is a single bit, which is set to one if the expression value is nonzero.

The format of SET is:

SET,sym=exp.

The parameters are positional and required.

sym Symbolic name to be set.

exp A CCL expression. Character strings must be integer constants, symbolic names, or CCL functions.

Examples:

1. **Job Dayfile**

```
SET,R3=4.
DISPLAY,R3.
      4 4B
SET,R3=R3+1.
DISPLAY,R3.
      5 5B
```

The value of control register 3 (R3) is set to 4, as the DISPLAY statement shows. On the second SET statement, the expression R3+1 increments the value of R3 by 1.

2.
```
IFE,EF=CPE,SKIP.
SET,R1=1.
ENDIF,SKIP.
DISPLAY,R1.
```

IFE compares the value of the error flag (EF) to the value indicating a CPU abort error (CPE). If they are equal, the value of control register 1 (R1) is set to 1. If the values of EF and CPE are not equal, the SET statement is skipped. Whether EF equals CPE or not, the value of R1 is displayed as 0 or 1.

3. **Input**

```
SKIP,HERE.
COMMENT.NO. 1
ENDIF,HERE.
SET,DSC=1.
IFE,DSC=0,THERE.
COMMENT.NO. 2
ENDIF,THERE.
```

Job Dayfile

```
SKIP,HERE.
ENDIF,HERE.
SET,DSC=1.
IFE,DSC=0,THERE
.CCL..COMMENT.NO. 2
ENDIF,THERE.
```

In this example, the value of DSC is initially 0. COMMENT.NO.1 is not printed in the dayfile. DSC is set to 1, making the IFE expression false. CCL skips to ENDIF,THERE. but prints COMMENT.NO.2 in the dayfile.

FUNCTIONS

CCL functions determine attributes of a file with one exception, the NUM function analyzes parameters. CCL functions can be used as an entire CCL expression or as a part of an expression. These functions are not statements in themselves and must be part of a CCL statement. The CCL functions are as follows:

FILE	Determines the status and attributes of a file
DT	Determines the device type of a file
NUM	Determines if a parameter has a numeric value

FILE

The FILE function determines the attributes of a file. Each symbolic name listed in the expression is evaluated as true (1) or false (0). Whenever more than one symbolic name is used in an expression, FILE analyzes each single name and applies the operators to the resultant values [that is, FILE (lfn, $sname_1$+$sname_2$) translates to FILE (lfn,$sname_1$)+FILE(lfn,$sname_2$)]. The result of a FILE function using arithmetic operators is not necessarily 1 or 0.

The format of the FILE function is:

FILE(lfn,exp)

The parameters are positional and required.

lfn Logical name of the file to be analyzed.

exp An expression consisting of operators and FILE symbolic names; may consist of one FILE symbolic name.

The FILE function is used as an expression or part of an expression in a CCL statement. The expression within a FILE function may not include the NUM function or another FILE function.

The FILE function must use exactly the same combination of separators and terminators shown in the format. If there is any deviation, CCL considers it an error and aborts the job.

Only the DT function or the following symbolic names can be used within the expression of a FILE function. Any other symbolic name or character string within the expression is unknown or an implicit DT function (see DT later in this section).

MS File is on mass storage.

OP File is opened.

AS File is attached to the user's job (that is, NOS/BE recognizes the file's lfn; the file exists).

IN File is INPUT.

PR File is a print file.

PH File is a punch file.

PF File is an attached permanent file.

LO File is local [that is, the file is a temporary (scratch) file; attached permanent files are not local].

TT File is connected to a terminal.

TP File is on a magnetic tape.

EOF The last operation moved forward and encountered an EOP; the file is now positioned after the EOP.

BOI File is positioned at the beginning-of-information (only valid for mass storage).

LB File is labeled.

WR File has write permission. (Write permission is modify permission, extend permission, or both.)

RD File has read permission.

MD File has modify permission.

EN File has extend permission.

Examples:

1.
```
IFE,FILE(WHOFILE,LO.OR.TP),LEAP.
CATALOG,WHOFILE,PERMFILE,ID=MINE.
ENDIF,LEAP.
UNLOAD,WHOFILE.
```
If WHOFILE is either a temporary file (LO) or on a magnetic tape (TP), WHOFILE is cataloged as permanent file (PERMFILE) on mass storage and unloaded. If WHOFILE is neither on a temporary file nor on a magnetic tape, WHOFILE is returned to the system by UNLOAD.

2.
```
IFE,FILE(MYFILE,BOI),MOVE.
COPY,INPUT,MYFILE.
ELSE,MOVE.
REWIND,MYFILE.
COPY,INPUT,MYFILE.
ENDIF,MOVE.
```
MYFILE is an attached permanent file with all permissions granted. If MYFILE is at the beginning-of-information, it is modified by COPY. The ELSE statement initiates skipping that is terminated by ENDIF. If MYFILE is not at the beginning-of-information, ELSE stops the IFE SKIPPING; MYFILE is then rewound and modified. The ENDIF statement is ignored.

DT

The DT function determines information about the device type on which a file resides. DT may be used only within the expression of a FILE function. The value of the DT function is true if the 2-character mnemonic (dt) matches the 2-character device type mnemonic of the file. The mnemonic must be enclosed in parentheses, and the left parenthesis must be preceded by DT.

The format of the DT function is:

FILE (lfn, . . . DT(dt) . . .)
expression

The parameters are positional and required.

lfn Logical name of file to be analyzed.

dt A 2-character mnemonic indicating a device type. An entire FILE expression may consist of dt.

See Device Types, section 6 for the valid device type mnemonics.

CCL assumes that any 2-character symbol within a FILE function that is not a FILE function symbolic name is an implicit DT function (for example, if MT is found, it is treated as DT(MT) and is false unless the device type happens to be MT).

Examples:

In the following examples, file FRANK resides on a 7-track magnetic tape.

1. DISPLAY,FILE(FRANK,DT(MT)).

   yields

   1 1B

   It is true that FRANK is on a 7-track tape; therefore, the value 1 is displayed.

2. IFE,FILE(FRANK,TP.AND.DT(MT)),LS1.
   COPY,FRANK,OUTPUT.
   ENDIF,LS1.
   UNLOAD,FRANK.

   FRANK is on a 7-track magnetic tape; therefore, it is copied to output and then unloaded. If the DT function were false or FRANK were not on magnetic tape, FRANK would be unloaded.

NUM

The NUM function determines if a character string is numeric or not. It evaluates the character string as true (1) if it is numeric or false (0) if it is nonnumeric. NUM may be used as an expression or as part of an expression in a CCL statement. The NUM function may be more useful within a procedure. (An example of the NUM function used within a procedure can be found in the discussion of procedures later in this section.) CCL considers any deviation from the given format an error and aborts the job.

The format of the NUM function is:

NUM(c)

The parameter is required.

c A character string; 1 to 40 characters. Special characters must be $-delimited.

Example:

```
IFE,NUM(FTNFILE)=0,GO.
FTN,I=FTNFILE.
LGO.
ENDIF,GO.
COMMENT.DO SOMETHING ELSE.
```

The character string, FTNFILE, is nonnumeric; therefore, the IFE expression is true. The FORTRAN job is compiled and executed, and the comment is processed.

PROCEDURES

A procedure is a group of control statements (including CCL statements) separate from the job control statement record, which may be called by a job. Calling a procedure provides a simplified method of processing that group of control statements. A procedure can be called by a job repeatedly, by another procedure, or by itself.

The discussion of procedures is broken into the following four parts.

1. Procedure Residence

   Describes the various ways a procedure may be stored.

2. Procedure Structure

   Explains the composition of a procedure; it contains the following subsections.

   a. Procedure Header Statement

      Explains the CCL statement required of all procedures.

   b. Procedure Body

      Discusses the special capabilities of all statements within a procedure other than the procedure header statement.

3. Procedure Call and Return

   Describes how a procedure is inserted into the job control statement stream; it contains the following subsections.

   a. Procedure Call

      Describes how a procedure may be called by its name or by a BEGIN statement. These two types of calling statements are referred to as the procedure call statement. This subsection explains ways the parameters on a procedure call statement may be processed and how these parameters may be substituted into the procedure body.

   b. Procedure Return

      Explains how processing returns from a procedure to the calling job or procedure. This is done automatically by CCL or by the user with a REVERT statement. This subsection also indicates how the values of various symbolic names are affected by a return from a procedure.

4. Procedure Commands

   Describes the following special CCL commands which may be used within a procedure.

   . DATA Permits data to be stored within a procedure.

   .EOR Writes an EOR onto the data file.

   .EOF Writes an EOP onto the data file.

   .* Enables the user to include in a procedure comments that will not be printed in the user dayfile.

PROCEDURE RESIDENCE

A procedure is stored as a record (system-logical-record) on a file or library. When CCL reads the procedure, it leaves the file positioned at the beginning of the record following the procedure. Hence, data can be stored after the procedure to be processed with the procedure. A procedure may have the following relationships to the file on which it is stored.

One procedure may occupy the entire file (one record).

Several procedures may reside on the same file in records stored in a sequential manner. When the procedure is specified by name, the file is searched for this procedure in a circular fashion. CCL automatically rewinds the file and continues searching if an EOI is reached before the procedure is found. If the entire file is searched and the procedure is not found, CCL aborts the job.

Processing a procedure does not alter the contents of the file from which it was obtained.

A file containing procedures may be a permanent file.

A procedure may be stored in rotating mass storage if the file containing it is sequential.

A procedure may be stored on a library.

A procedure may reside on the input file of the calling job; however, CCL does not rewind the input file if an EOI is encountered before the procedure is found.

To save a procedure through INTERCOM program text editor (EDITOR), use the NOSEQ parameter on the SAVE command. If the NOSEQ parameter is omitted, the EDITOR line numbers are not removed and interfer with CCL processing.

PROCEDURE STRUCTURE

A procedure consists of a procedure header statement and a procedure body. The procedure must be named and begin with the header statement. The body contains all statements between the header statement and an end of record. The body must contain at least one control statement. All control statements are legal within a procedure, including CCL statements. The body may also include special procedure commands and data.

PROCEDURE HEADER STATEMENT

A procedure starts with a procedure header statement that declares the name of the procedure and identifies any formal keywords and their default values. Unless it contains an error, the header statement is not printed in the dayfile. The characters . PROC must be followed by a comma. The header statement must be terminated by a period. The separator between parameters must be a comma. A header statement may continue over more than one card or line of input, provided that a separator is the last character on each card or line. Procedure names can be 1 to 7 alphanumeric characters. They may begin with or consist entirely of numeric characters.

The format of the procedure header statement is:

.PROC,pname,$p_1 p_2,\ldots,p_n$.

The pname parameter is positional and required; parameters p_i are optional.

pname The name of the procedure (may be 1 to 7 alphanumeric characters beginning with a numeric character). pname cannot be BEGIN.

p_i A user-created parameter in one of the following forms.

fk	Formal keyword
fk=default1	Formal keyword with default
fk=default1/default2	Formal keyword with two defaults

fk is its own default1 value if a default value is not specified. If two defaults are specified, the user can select only one on the procedure call statements. The form fk= may be used to specify a null default. The formal keywords must be 1 to 10 alphanumeric characters; formal keywords beginning with or consisting of numerics are legal. The defaults may be 1 to 40 alphanumeric characters. The values default1 and default2 may be $-delimited character strings. The number of parameters is limited to an installation-defined number. The released default value is 50.

Formal keywords appear in the procedure body and may be replaced by their default values during substitution. Substitution occurs when the procedure is called, replacing formal keywords with default values or values from the call statement.

A formal keyword may be $-delimited in any p_i form; however, it may contain only alphanumeric characters. If it contains special characters, substitution does not occur.

There are two special defaults which can be used in the procedure header statement. These defaults are ≡FILE and ≡DATA (# FILE and #DATA in ASCII). CCL supplies the value of these defaults. They can be overridden just as any other default of a formal parameter by specification on a procedure call statement.

If the default specification is ≡FILE, its value is the name of the file from which the procedure was read, and the record after the procedure is read. If the procedure resides on a library, using ≡FILE produces the characters ≡LIB during substitution. When the default specification is ≡DATA, the value is the name of the default temporary file to which a .DATA command writes data statements. (The .DATA command is explained later in this section.)

Examples:

1.
```
. PROC,ZZZ,P1=≡FILE.
FTN,I=P1.
```
After procedure ZZZ is called, the FTN compiler looks for input from the file containing procedure ZZZ. The record following ZZZ is read.

2.
```
. PROC,SNORE,P2=≡DATA.
FTN,I=P2.
```
After procedure SNORE is called, the FTN compiler searches for input from the file generated by the . DATA command.

PROCEDURE BODY

The procedure body consists of all statements which follow the procedure header statement. The procedure body can contain any control statement including CCL statements, as well as calls to other procedures.

The procedure body may use any of the formal keywords defined on the procedure header statement. When a procedure is called, CCL scans the statements of the procedure body before processing. The values specified by the parameters in the procedure call statement are substituted for the occurence of the formal keywords in each statement of the procedure body. If a formal keyword is not indicated on the call statement, default1 from the header statement replaces occurrences of the formal keyword. A parameter on the call statement may indicate default2.

The right arrow, → (underline character in ASCII), is reserved by CCL for use as a linking character. It may be used within the procedure body to separate two character strings when they do not have a standard separator between them. After processing and substitution, any right arrows are removed, and the character strings are joined.

An equivalence symbol, ≡ (# in ASCII), inhibits substitution of a character string if it immediately precedes the string. The character string is not compared to the list of formal keywords, substitution is inhibited, and the equivalence symbol is removed. If a separator immediately follows an equivalence symbol, CCL accepts the separator without examination. Consecutive equivalence symbols yield one equivalence symbol, and substitution occurs if a formal keyword follows. An equivalence symbol followed by a right arrow produces one right arrow, and the character string that follows is substituted if it is a formal keyword. When two character strings are separated by a right arrow followed by an equivalence symbol, CCL joins the character strings without substitution of the second string.

Examples:

1. .PROC,INHIBIT,I=BB.
   FTN,≡I=I.
   COMMENT.≡I QUIT,QU→I→T. QU→≡I→T.

   Substitution yields:

   FTN,I=BB.
   COMMENT.I QUIT, QUBBT, QUIT.

   Only the single character I is recognized as a formal keyword; therefore, character string QUIT is not affected by substitution. When I is separated by right arrows, QU, I, and T become separate character strings.

2. . PROC,SUB,I=BB.
   FTN,I=I.
   COMMENT.≡≡I QU≡→I→T, QU≡→IT.

   Substitution yields:

   FTN,BB=BB.
   COMMENT. ≡BB QU→BBT,QU→IT.

   Without the equivalence symbol before I, the FTN parameter is meaningless after substitution.

PROCEDURE CALL AND RETURN

A procedure is stored outside the job control statement record and CCL logically inserts it into the job control stream when the job calls it. A call statement in the job calls the procedure. After the final processed control statement of the procedure, a CCL† - or user-issued REVERT statement continues processing with the job control statement after the call statement (figure 5-1).

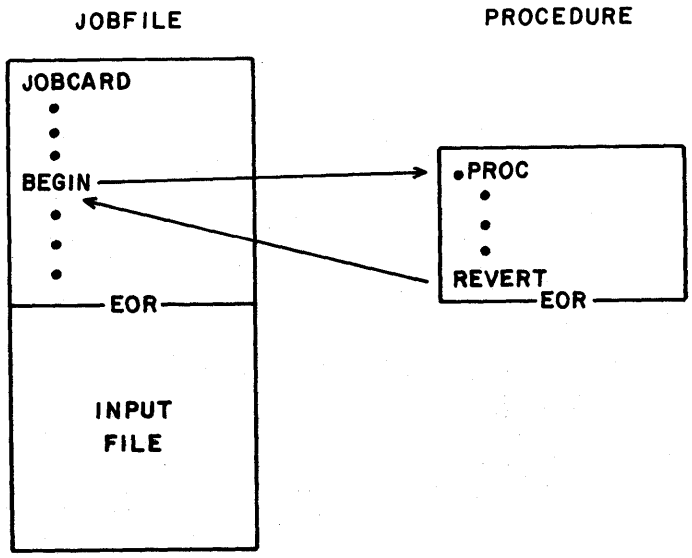


Figure 5-1. Calling a Procedure from a Job

If called by a call statement in another procedure, the procedure is logically inserted into that procedure's control statement stream. After the final processed control statement of the procedure, a CCL† - or user-issued REVERT statement continues processing in the calling procedure with the control statement after the call statement (figure 5-2).

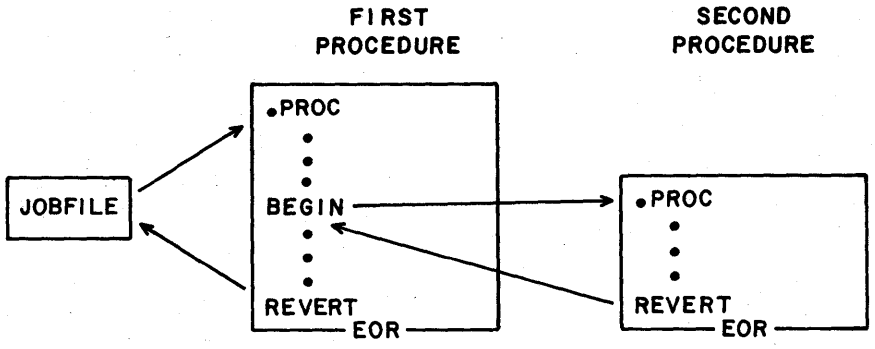


Figure 5-2. Calling a Procedure from Another Procedure

† CCL issues a REVERT statement if the user does not supply one within a procedure.

PROCEDURE CALL

Procedures may be called (caused to be processed) by a procedure call statement in a job or procedure. The call statement is either a BEGIN or call-by-name statement. The term call statement refers to both forms of the statement. The BEGIN statement can call a procedure on a permanent file, and the call-by-name statement can call a procedure on a library. Both statements can call a procedure on a local file.

The syntax of the call statement is similar to that of other control statements. The call statement must be terminated by a period or a right parenthesis. The separator following each parameter must be a comma. The call statement may continue over more than one card or line or input if a separator is the last character on each card or line.

BEGIN STATEMENT

The format of BEGIN is:

BEGIN,pname,pfile,p_1,p_2, . . . ,p_n.

The pname and pfile parameters are positional. When the user does not specify a parameter, CCL assumes a default value. Parameters p_i are optional.

pname — The procedure name as declared on the header statement. Default is the next procedure on file pfile. If the default is used and pfile is at end of information, CCL rewinds pfile and calls the first procedure.

pfile — The name of the file where procedure pname is located. Default is an installation-defined file name. If the default is used, CCL searches for a local file with the default file name, then attempts to attach a permanent file with the default file name and id PUBLIC.

p_i — A parameter having one of the following forms:

- fk — A formal keyword; fk is the same keyword used in the related parameter on the procedure header statement.
- v — A value; may be 1 to 40 characters. If any characters except a slash (/) or a leading minus are nonalphanumeric, v must be a literal (for example, A/B/C/ is a 5-character specification for a formal keyword).
- fk=v — Value v is substituted for formal keyword fk.

To call a procedure pname on a file pfile, CCL searches for a local file pfile. If no local pfile is found, CCL attempts to attach permanent file pfile with user id PUBLIC. Once pfile is located, CCL searches for procedure pname. If pfile or pname is not found, the job aborts. Parameters correspond to formal keywords declared on the header statement. If the user needs only default values, the BEGIN statement may have no parameters. The separator following the verb BEGIN must be a comma or a left parenthesis.

CALL-BY-NAME STATEMENT

The format of call-by-name is:

pname,p_1,p_2,. . .,p_n.

The pname parameter must be specified; parameters p_i are optional.

pname — The procecure name as declared on the header statement. There is no default.

p_i — A parameter having one of the following forms (see BEGIN Statement for further explanation):

- fk — A formal keyword.
- v — A value; 1 to 40 characters.
- fk=v — Value v is substituted for formal keyword fk.

To call a procedure by name, CCL searches for a local file pname. If file pname is found, CCL looks for procedure pname on it. If no local file pname is found, CCL searches the currently defined library set for procedure pname. If procedure pname is not found, the job aborts. Parameters correspond to formal keywords declared on the header statement. If the user needs only default values, the call-by-name statement may have no parameters. The separator following pname must be a comma or a left parenthesis.

SUBSTITUTION

When CCL calls a procedure, it searches the procedure body for any formal keywords declared on the header statement. Substitution occurs when CCL replaces formal keywords in the procedure body with default values from the header statement or values from the call statement. The parameters on the call statement determine which values replace the formal keywords.

The two modes in which CCL processes parameters on the call statement are positional and equivalence. In positional mode, parameters are a value v or null. The values in the call statement are substituted for formal keywords from the procedure header statement, replacing first keyword with first value, second keyword with second value, and so on (that is, wherever the first keyword from the header statement appears in the procedure body, it is replaced by the first value listed on the call statement, and so forth). A null parameter indicates the first default from the header statement, not a null string. A null string may be specified only by fk=. If the user places an fk parameter in positional mode, CCL treats it as a v parameter.

Example:

Procedure on File MYFILE	**Calls and Expansion**	**Explanation**
. PROC,EXAMPLE,I,J,K=XY. REWIND,I,J,K.	BEGIN,EXAMPLE,MYFILE. yields REWIND,I,J,XY.	All defaults from the header statement are used.
	BEGIN,EXAMPLE,MYFILE,B,,Z. yields REWIND,B,J,Z.	B replaces I, and Z replaces K in the procedure body. The empty parameter indicates default1 for J.

In equivalence mode, the parameter forms are fk=v and fk. If a formal keyword with two defaults was specified on the procedure header statement, the user can select the formal keyword's second default with an fk parameter. The position of fk on the call statement is irrelevant. fk=v replaces formal keyword fk with value v, regardless of its position. If a parameter is not specified on the call statement for a formal keyword with either one of two defaults, the first default is substituted. A null parameter is ignored in equivalence mode. If the user indicates more than one value for a formal keyword on the call statement, the last specification is used. A v parameter in equivalence mode is an error. CCL issues an error message and aborts the job.

fk=v always specifies fk to have the value v. If v is null (fk=), appearances of fk in the procedure body are replaced by a null string. fk=sym+, fk=sym+D, and fk=sym+B cause the numeric value of the symbolic name sym to be converted to a decimal (fk=sym+,fk=sym+D) or octal (fk=sym+B) display code value. This value replaces fk in the procedure body. sym may be numeric (for example, J=99+B). These forms are not valid on the header statement.

Example:

Procedure on File MYFILE	**Calls and Expansion**	**Explanation**
.PROC,EXAMPLE,I,J=A/B. COPY,I,J.	BEGIN,EXAMPLE,MYFILE. yields COPY,I,A.	J is replaced by its first default; I is replaced by its first and only default.
	BEGIN,EXAMPLE,MYFILE,I=F,J. yields COPY,F,B.	F replaces I; the J parameter indicates second default for J.
	BEGIN,EXAMPLE,MYFILE,I=,J=Z. yields COPY,,Z.	I=substitutes a null string for I; Z replaces J.

The processing of parameters on the call statement is initially in positional mode. The switch to equivalence mode occurs as follows:

Equivalence mode is entered at the first occurrence of an fk=v parameter within the call statement.

If the procedure header statement contains a formal keyword with a second default, equivalence mode is entered after n parameters have been processed. n is the number of formal keywords in the header statement up to, but not including, the first formal keyword with a second default.

Example:

Procedure on File AFILE	**Call and Expansion**	**Explanation**
. PROC,SHOW,I,J=A,K=X,L=Y/Z. RETURN,I,J,K,L.	BEGIN,SHOW,AFILE,R,S,T,L. yields RETURN,R,S,T,Z.	L=Y/Z is in fourth position, setting n=3. R, S, and T are in positional mode: L is in equivalence mode, selecting the second default Z.

The rules of substitution in a procedure body are as follows:

Each line of the procedure body is divided into character strings and separators. If a character string is identical to a formal keyword, substitution occurs unless inhibited by an equivalence symbol.

All nonalphanumeric characters are separators, including blanks and dollar signs that bracket literals.

A right arrow is deleted, linking preceding and following strings.

One equivalence symbol is deleted, inhibiting substitution of the character string it precedes.

CCL does not remove blanks.

Normally, a $-delimited (literal) value v in a call statement is evaluated before substitution. Outer dollar signs are stripped off, and inner double dollar signs are reduced to a single dollar sign. However, if the related formal keyword in the procedure header statement is also $-delimited, evaluation of the literal value does not occur. Instead, CCL substitutes the literal with dollar signs intact. That is, substituting a literal value for a literal formal keyword produces a value that is delimited by double dollar signs.

PROCEDURE CALLS AND SUBSTITUTION EXAMPLES:

Procedure	Calls and Expansion	Explanation
1. . PROC,EXAMPLE,I,J,K=XY. REWIND,I,A,J,K.		Procedure EXAMPLE is on file MYFILE. A is always rewound since it is not a formal keyword.
	BEGIN,EXAMPLE,MYFILE,R,T,S. yields REWIND,R,A,T,S.	Parameter processing remains in positional mode.
	BEGIN,EXAMPLE,MYFILE,K=YY. yields REWIND,I,A,J,YY.	Equivalence mode is entered at once with K=YY. I and J assume their defaults.
	BEGIN,EXAMPLE,MYFILE,R, K=F,J=B. yields REWIND,R,A,B,F.	Equivalence mode is entered with K=F.
2. .PROC,ITEM,LGO,N=1/ 10000,L=OUTPUT/LIST. ITEMIZE,LGO,≡L=L,≡N=N.		Procedure ITEM is on a local file with the CCL default file name.
	BEGIN,ITEM,,LFN. yields ITEMIZE,LFN,L=OUTPUT,N=1.	≡ before L and N in ITEMIZE inhibits substitution of these two characters. First defaults apply to L and N. The order of formal keywords on the header statement does not affect the order of substituted keywords on ITEMIZE.

Procedure	Calls and Expansion	Explanation
	BEGIN,ITEM,,L=B. yields ITEMIZE,LGO,L=B,N=1.	B is substituted for L in equivalence mode while LGO and N assume first defaults.
	BEGIN,ITEM,,L,L. yields ITEMIZE,L,L=LIST,N=1.	The first L on the BEGIN statement is in positional mode. Specifying the second L in equivalence mode indicates L's second default.
	BEGIN,ITEM,,L,5. Fatal error	5 is in equivalence mode and does not match any formal keyword.
3. . PROC,TEST,LFN,PAR=D. COMPASS,I=LFN,PAR.		Procedure TEST is on file NOW.
	BEGIN,TEST,NOW,FILE1. yields COMPASS,I=FILE1,D.	FILE1 is substituted for LFN in positional mode; PAR assumes its first default.
	BEGIN,TEST,NOW,FILE2,$L= LIST,D$ yields COMPASS,I=FILE2,L=LIST,D.	In positional mode, FILE2 is substituted for LFN, and L=LIST,D is subsitutted for PAR. The PAR substitution creates an additional COMPASS parameter.

CALL BY NAME EXAMPLE:

Procedure	Calls and Expansion	Explanation
.PROC,GERTIE,A,B,C,M=MODE, BO=R/S. COPYL,A,B,C. M IS BO		Procedure GERTIE is on the currently defined library set.
	GERTIE(,,,MO) yields COPYL,A,B,C. MO IS R	Processing remains in positional mode with the empty parameters indicating first defaults and MO replacing M. Since the call statement does not specify a value for BO, its first default (R) is used.
	GERTIE,R,C=NEW,BO. yields COPYL,R,B,NEW. MODE IS S	The first parameter is positional mode, but equivalence mode starts with C=NEW. Equivalence mode causes BO to apply to BO, not M.

NUM FUNCTION SAMPLE PROCEDURE:

In the following example, the procedure SHOW is on file SAMPLE. The call statements indicate the procedure name and the file the procedure is on.

Procedure	Calls and Expansion	Explanation
. PROC,SHOW,IN=FTNFILE. IFE,NUM(IN)=0,LS. FTN,I=IN. LGO. ENDIF,LS. COMMENT. DO SOMETHING ELSE		
	BEGIN,SHOW,SAMPLE. yields IFE,NUM(FTNFILE)=0,LS. FTN,I=FTNFILE. LGO. ENDIF,LS. COMMENT.DO SOMETHING ELSE	No parameters were specified on the call statement; therefore, IN assumes its first default. The NUM function evaluates the substituted formal keyword as nonnumeric, making the IFE expression true. The FORTRAN job is compiled and executed and the comment is processed.
	BEGIN,SHOW,SAMPLE,8. yields IFE,NUM(8)=0,LS. FTN,I=8. LGO. ENDIF,LS. COMMENT.DO SOMETHING ELSE	The value 8 on the call statement is in positional mode replacing IN. The NUM function evaluates 8 as a numeric, making the IFE expression false. The FTN and LGO statements are skipped, and the comment is processed.

PROCEDURE RETURN

A REVERT control statement causes processing to return to the calling job or procedure at the statement immediately following the procedure call statement.

The format of REVERT is:

REVERT.	Processing returns to the calling job or procedure.
or	
REVERT,ABORT.	Same as REVERT., except that after processing returns, CCL issues an abort instead of a normal exit.

A REVERT statement can appear anywhere within a procedure. REVERT is commonly used in conjunction with a conditional statement to cause premature return to the calling job or procedure. The user may place REVERT at the end of a procedure, but this is unnecessary because CCL provides an implicit REVERT sequence. The CCL following each statement identifies it as generated by CCL.

REVERT.CCL EXIT,S.CCL REVERT,ABORT.CCL	This is the REVERT sequence CCL provides after the last processed statement of a procedure. If the last statement did not produce a fatal error, CCL processes the REVERT. statement. If the last statement did produce a fatal error, the first statement in this sequence is skipped. CCL provides an EXIT,S. statement to terminate skipping, and processes the REVERT,ABORT. statement.

The user may wish to use a EXIT control statement to create a REVERT sequence. The EXIT statement produces the same results whether it is in a procedure or the job file; it does not cause a return to the calling job or procedure. EXIT should be used with caution because it can terminate the job.

Example:

LGO. REVERT. EXIT,S. DMP. REVERT,ABORT.	If a fatal error occurs during the processing of the LGO. statement, the system skips to the EXIT,S. statement. SCM is dumped and CCL processes the REVERT,ABORT. statement. If no fatal error occurs during the processing of the LGO. statement, CCL processes the REVERT. statement.

During a REVERT, CCL might change the value of symbolic names R1, R2, R3, EF, EFG, and DSC. Their values can be set by the user before BEGIN and REVERT.

The values of control registers R1, R2, and R3, the error flag (EF), and the dayfile skipped control statement flag (DSC) are saved at the time of a procedure call and restored by a REVERT. If the value of control register R1G is changed within a procedure, REVERT does not restore it to the value before the procedure call.

Example:

1. The user sets DSC to one before a procedure is called.
2. Within the procedure, the user sets DSC to zero.
3. When CCL processes a REVERT, DSC is again one.

When the global error flag (EFG) is zero, it is set to the value of EF during a REVERT. REVERT restores EF to its value at the time of the procedure call (zero). This means the value of EF in the procedure may be passed back to EFG in the control statement sequence that call the procedure. The value of EF is not transferred to EFG unless the values of EF and EFG are zero before the procedure is called.

The following example shows input from the job file and a procedure and the resulting dayfile output.

	Input	Dayfile	Comments
Job file	.		
	.		
	.		
	SET,EF=0.	SET,EF=0.	EF and EFG are set to 0.
	SET,EFG=0.	SET,EFG=0.	
	BEGIN,TEST.	BEGIN,TEST.	
	DISPLAY,EF.	SET,EF=1.	EF is changed to 1.
	DISPLAY,EFG.	DISPLAY,EF.	
	.	1 1B	This portion of the dayfile is from the procedure TEST.
	.	DISPLAY,EFG.	
	.	0 0B	
		REVERT.CCL	
Procedure	. PROC,TEST.	DISPLAY,EF.	The value of EF is restored to 0.
	SET,EF=1.	0 0B	
	DISPLAY,EF.	DISPLAY,EFG.	The former value of EF is passed back to the job in EFG.
	DISPLAY,EFG.	1 1B	

PROCEDURE COMMANDS

Procedure commands are similar to directives and may be included in the body of a procedure to control the processing of data within the procedure. As CCL processes each statement of a procedure body, it makes any necessary substitutions and determines if the resultant statement is a procedure command. The commands have a fixed format with a period in column 1 and the command name beginning in column 2; the commands do not have a terminator. All command requirements must be met exactly; if not met, a statement is assumed not to be a command.

.DATA

A . DATA command separates control statements of the procedure from subsequent data statements. It allows the same parameter substitution occurring in the control statements to occur within the data statements and . DATA command. All statements following the . DATA command are written to a file when the procedure is called. Comments cannot follow the command name or the parameter (if specified); the remainder of the command must be blank.

The format of the . DATA command is:

. DATA,lfn

The parameter is optional.

lfn Writes data to file lfn; if not specified, CCL writes data to a default file name.

The . DATA command causes CCL to generate a temporary file and supply a default name unless a file name is specified by the user. Statements within the procedure may reference the default file via the special default ≡DATA in the procedure header statement. If the file already exists, CCL returns it and creates a new file. Hence, the .DATA command cannot be used to add data to an existing file. After the data is written, the file is rewound. If the user specifies a file name on the .DATA command, the ≡DATA default will not reference that file name. A user-specified file name must either be declared on the header statement for substitution or used directly in the procedure body.

Data associated with a procedure may also be contained in the records following it. The user may reference this data with the ≡FILE default in the procedure header statement. The procedure file is always positioned at the beginning of the record following the called procedure. When procedure data is not contained in the procedure, substitution of parameters does not occur within the data.

Examples:

In the following examples, both procedures and their data are on file SLEEP.

1.

Procedure

```
.PROC,SNORE,P1=≡DATA,X=Y.
FTN,I=P1.
COMMENT.IF X IS TRUE,IT IS
JUNE.
   .
   .
   .
.DATA
   PROGRAM X(INPUT,OUTPUT)
   .
   .
   .
```

After Substitution

```
BEGIN,SNORE,SLEEP.
FTN,I=ZZCCLAA.
COMMENT. IF Y IS TRUE, IT IS
JUNE.
   .
   .
   .
.DATA
   PROGRAM Y(INPUT,OUTPUT)
      .
      .
      .
```

All input after the .DATA command has been written onto the default temporary file, ZZZCCLAA. The ≡ DATA default tells the FTN compiler to search for input from ZZCCLAA. Substitution occurs in the FORTRAN program, as well as in procedure SNORE.

2.

```
.PROC,ZZZ,P2=≡FILE,X=Y.
FTN,I=P2.
COMMENT.IF X IS TRUE, IT IS
   JUNE.
   .
   .
   .
7/8/9
   PROGRAM X(INPUT,OUTPUT)
      .
      .
      .
```

```
BEGIN,ZZZ,SLEEP.
FTN,I=SLEEP.
COMMENT.IF Y IS TRUE, IT IS
   JUNE.
      .
      .
      .
7/8/9
PROGRAM X(INPUT,OUTPUT)
   .
   .
   .
```

The ≡FILE default tells the FTN compiler to search for input from file SLEEP. Since CCL left file SLEEP positioned after ZZZ, the compiler starts reading immediately after procedure ZZZ. Substitution occurs in ZZZ but not in the FORTRAN program.

.EOR

The . EOR command causes an end-of-record to be recorded on the data file specified by a . DATA command. After a . EOR command, statements are written onto a new record. CCL recognizes . EOR only after it encounters a . DATA command. Comments cannot follow the command name; the remainder of the command must be blank.

The format of the . EOR COMMAND is:

```
.EOR
```

.EOF

The . EOF command causes an end-of-partition to be recorded onto the data file specified by a . DATA command. After a . EOF command, statements are written onto a new partition. EOF is used instead of EOP for compatibility with other systems. CCL recognizes . EOR only after it encounters a . DATA command. Comments cannot follow the command name; the remainder of the command must be blank.

The format of the .EOF command is:

```
. EOF
```

.*

A . * command provides comments within a procedure. These comments do not appear in the dayfile. If the user wants comments to appear on the dayfile, the comment statement should be used. The remainder of this command (after the . *) can contain any combination of characters. When CCL calls a procedure, it discards all . * commands before it begins processing.

The format of the . * command is:

```
.* comment
```

SAMPLE JOBS

The following jobs demonstrate the use of a procedure and some of the other capabilities of CCL. Control statements generated by the processing of a procedure are indicated by a bracket.

1. This job demonstrates the use of the . DATA command. The user calls a procedure FTNPROC containing a FORTRAN program in the procedure body. The statements preceding the . DATA command compile and execute the program.

Job

```
BEGIN,FTNPROC,INPUT.
7/8/9
. PROC,FTNPROC,K=ΞDATA.
FTN,I=K.
LGO.
.DATA
        PROGRAM FTNPROC(OUTPUT)
        A=8.8
        B=4.4
        C=A/B
        PRINT *,C
        STOP
        END
6/7/8/9
```

Dayfile

```
BEGIN,FTNPROC,INPUT.
FTN,I=ZZCCLAA.
COMPILING FTNPROC
LGO.
    STOP
      .004 CP SECONDS EXECUTION TIME
REVERT.CCL
```

2. The user has a permanent file HERE containing a FORTRAN program he wishes to execute. The user attaches the file and calls procedure THIS to execute the program.

Job

```
ATTACH,HERE,ID=MINE.
BEGIN,THIS,INPUT.
7/8/9
.PROC,THIS,J=HERE.
FTN,I=J.
LGO.
6/7/8/9
```

Dayfile

```
ATTACH,HERE,ID=MINE.
PFN IS
HERE
PF CYCLE NO. = 001
BEGIN,THIS,INPUT.
FTN,I=HERE.
COMPILING HERE
LGO.
    STOP
      .003 CP SECONDS EXECUTION TIME
REVERT.CCL
```

3. In this example, the user wishes to execute a FORTRAN program on permanent file HERE; however, the user suspects that file HERE has been purged. The IFE statement checks the attach with the FILE function. If the attach is successful, procedure THIS executes the program. The ELSE statement skips control statements up to the ENDIF statement.

   If the attach is unseccessful, CCL skips to the ELSE statement. Another permanent file (THERE) containing a FORTRAN program is attached. Procedure THIS is called to execute this program. The call statement indicates THERE replaces formal keyword J in the procedure body.

Job

```
ATTACH,HERE,ID=MINE.
EXIT,U.
IFE,FILE(HERE,PF),GO.
BEGIN,THIS,INPUT.
ELSE,GO.
ATTACH,THERE,ID=MINE.
BEGIN,THIS,INPUT,J=THERE.
ENDIF,GO.
7/8/9
.PROC,THIS,J=HERE.
FTN,I=J.
LGO.
6/7/8/9
```

Dayfile

```
ATTACH,HERE,ID=MINE.
PFN IS
HERE
FILE NOT CATALOGUED, SN=PFQSET
PF ABORT
EXIT,U.
IFE,FILE(HERE,PF),GO.
ELSE,GO.
ATTACH,THERE,ID=MINE.
PFN IS
THERE
PF CYCLE NO. = 001
BEGIN,THIS,INPUT,J=THERE.
FTN,I=THERE.
COMPILING THERE
LGO.
    STOP
      .002 CP SECONDS EXECUTION TIME
REVERT.CCL
ENDIF,GO.
```

4. The user has two permanent files. File OTHER contains procedure THAT, which compiles and attempts to execute a FORTRAN program. File WHERE contains two FORTRAN programs the user wishes to check for errors.

   Once the two files are attached, the WHILE and ENDW statements bracket the control statements that test the programs on file WHERE. After ensuring both error flags (EF and EFG) have zero values, procedure THAT is called. THAT attempts to execute the program read from file WHERE. If there are fatal FORTRAN errors, the job aborts when the program is loaded. Processing resumes when the EXIT(S) statement is encountered. If there are no fatal FORTRAN errors, the SKIP statement skips over the EXIT(S) statement to ENDIF,JUMP. and processing resumes. The DISPLAY statement indicates the value of EF.

   Loading a FORTRAN program containing a fatal error changes EF to a nonzero value. When processing returns to the job control statement record, the value of EFG is set to the value that EF last had in the procedure. If there were no fatal errors, the value of R2 is incremented by one. R2 counts the number of executable FORTRAN programs. Each pass through the bracketed statements increments the value of R1 by one. R1 counts the number of passes through the bracketed statements.

CCL processes the WHILE statement a third time. CCL evaluates the WHILE expression as false, and all statements are skipped until an ENDW statement with a matching label string is found. At the end of the job, the value of R2 is displayed so the user can tell at a glance the number of successfully executed FORTRAN programs.

Job

```
ATTACH,OTHER,ID=MINE.
ATTACH,WHERE,ID=MINE.
WHILE,R1.LE.1,CIRCLE.
SET,EF=0.
SET,EFG=0.
BEGIN,THAT,OTHER,WHERE.
DISPLAY,EFG.
IFE,EFG=0,HOP.
SET,R2=R2+1.
ENDIF,HOP.
SET,R1=R1+1.
DISPLAY,R1.
ENDW,CIRCLE.
DISPLAY,R2.
6/7/8/9
```

Procedure (Contents of File OTHER)

```
.PROC,THAT,KEY.
FTN,I=KEY.
LGO.
SKIP,JUMP.
EXIT,S.
ENDIF,JUMP.
REWIND,LGO.
DISPLAY,EF.
6/7/8/9
```

Dayfile

```
ATTACH,OTHER,ID=MINE.
PFN IS
OTHER
PF CYCLE NO. = 001
ATTACH,WHERE,ID=MINE.
PFN IS
WHERE
PF CYCLE NO. = 001
WHILE,R1.LE.1,CIRCLE.
SET,EF=0.
SET,EFG=0.
BEGIN,THAT,OTHER,WHERE.
FTN,I=WHERE.
COMPILING ONE
LGO.
          FATAL LOADER ERROR -  SEE MAP
EXIT,S.
ENDIF,JUMP.
REWIND,LGO.
DISPLAY,EF.
     4    4B
REVERT.CCL
DISPLAY,EFG.
     4    4B
IFE,EFG=0,HOP.
ENDIF,HOP.
SET,R1=R1+1.
DISPLAY,R1.
     1    1B
ENDW,CIRCLE.
WHILE,R1.LE.1,CIRCLE.
SET,EF=0.
SET,EFG=0.
BEGIN,THAT,OTHER,WHERE.
FTN,I=WHERE.
COMPILING TWO
LGO.
    STOP
      .003 CP SECONDS EXECUTION TIME
SKIP,JUMP.
ENDIF,JUMP.
REWIND,LGO.
DISPLAY,EF.
     0    0B
REVERT.CCL
```

Dayfile

```
DISPLAY,EFG.
     Ø    ØB
IFE,EFG=Ø,HOP.
SET,R2=R2+1.
ENDIF,HOP.
SET,R1=R1+1.
DISPLAY,R1.
     2    2B
ENDW,CIRCLE.
WHILE,R1.LE.1,CIRCLE.
ENDW,CIRCLE.
DISPLAY,R2.
     1    1B
```

COMMUNICATION AREAS 6

FILE ENVIRONMENT TABLE

The file environment table (FET) is a communication area supplied by the user within his field length. Any file to be written, read, or otherwise manipulated or positioned, must have its own FET. The FET is interrogated and updated by the system and user file processing.

COMPASS programmers can create an FET in two ways:

Use the FET creating macros FILEB, FILEC, RFILEB, or RFILEC.

Use other COMPASS instructions to build a table in the format expected by the system.

Compiler language programmers need not be concerned with FET construction or manipulation, because the compilers will perform these tasks in response to compiler language instructions. When CYBER Record Manager is used for input/output, the user need supply only the file information table (FIT) data. CYBER Record Manager will construct and manipulate the FET from information in its FIT. The FIT is fully described in the CYBER Record Manager reference manual.

A minimum size FET is five words, which allows for processing of sequential unlabeled files. Random or labeled files, or files in which the user will process file conditions or errors with OWNCODE routines, require a longer table. Extensions to the FET – areas identified by pointers within the FET – are required for extended error and label processing. Some compilers append an area past word 13 of the FET, as explained in the respective manuals. When S and L tapes are processed, the FET must be at least seven words in length.

The format of the FET is shown in figure 6-1. Some fields are pertinent only to CYBER Record Manager manipulation; a description exists in the reference manual for CYBER Record Manager. Other fields contain different data depending on the file mode or residence.

FET CREATION MACROS

System macros in the COMPASS language facilitate generation of the FET.

All parameters except lfn, fwa, and f are optional. The fwa and f parameters must be in the order shown; others can be in any order. The macro parameters WSA, OWN, XPR, and IND are not order dependent, but order is fixed within these parameters.

The user must specifically allocate the circular buffer location in the field length as well as the buffers for the WSA, XPR, and XLR parameters. The macro identifies but does not create the buffers.

Four macros are available, depending on whether the file is coded or binary, random or sequential.

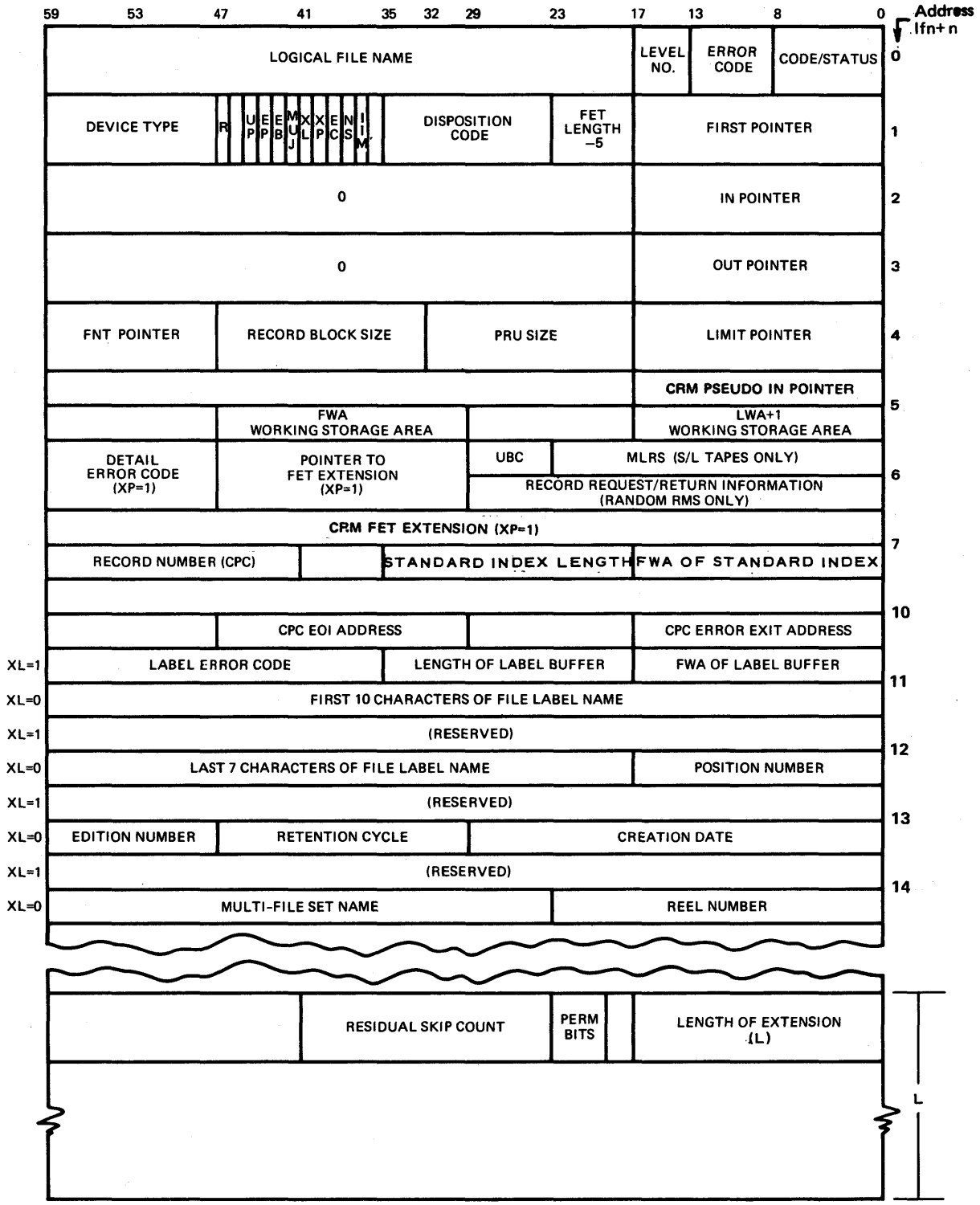


Figure 6-1. File Environment Table

CODED SEQUENTIAL FILE

```
lfn    FILEC    fwa,f,(WSA=addrw,lw),(OWN=eoi,err),LBL,UPR,EPR,XPR=xpadr,
                UBC=ubc,MLR=mlrs,(XLR=xladr,xll)
```

BINARY SEQUENTIAL FILE

```
lfn    FILEB    fwa,f,(WSA=addrw,lw),(OWN=eoi,err),LBL,UPR,EPR,XPR=xpadr,
                UBC=ubc,MLR=mlrs,(XLR=xladr,xll)
```

CODED RANDOM FILE

```
lfn    RFILEC   fwa,f,(WSA=addrw,lw),(IND=addri,li),(OWN=eoi,err),LBL,
                UPR,EPR,XPR=xpadr
```

BINARY RANDOM FILE

```
lfn    RFILEB   fwa,f,(WSA=addrw,lw),(IND=addri,li),(OWN=eoi,err),LBL,
                UPR,EPR,XPR=xpadr
```

Further explanation of parameter usage appears with descriptions of the FET fields below.

lfn	Logical file name
fwa	Circular buffer address; substituted in FIRST, IN, and OUT
f	Length of circular buffer; fwa + f is substituted in LIMIT to make buffer address lwa + 1; f should be at least one word larger than PRU size of the device on which the file resides
WSA	Working storage area keyword; parameters required for READIN and WRITOUT; relieves user of responsibility for buffer manipulation
addrw	First word address of working storage area
lw	Length of working storage; when coded files are being processed, the length must be at least as long as the longest record, or data will be lost
IND	Index buffer parameter keyword; required for name/number index random files only
addri	First word address of index buffer
li	Length of index buffer; for numbered indexed files, length should allow one word for each record plus a one word header; for named indexed files, two words are required for each record in addition to the index header
OWN	OWNCODE routine parameters keyword
eoi	Address of routine to be executed if end-of-volume, end-of-device, or end-of-information occurs; UPR must be used
error	Address of routine to be executed if file action errors occur; EPR must be used

UPR	User specifies processing at end-of-volume, end-of-pack for user device sets, or end-of-information; sets bit 45 of word 2
LBL	Label information will follow for magnetic tape file; LABEL macro providing label information must immediately follow the FET creating macro to which it pertains
EPR	User specifies handling of file action error conditions; sets bit 44 of word 2; does not set extended error processing flag
UBC	Unused bit count keyword; required only for S and L tapes
ubc	Specifies number of bits in last word of record that do not contain valid data
MLR	Maximum record size keyword; required only for S and L tapes
mlrs	Maximum number of 60-bit words in record
XPR	Extended error information to be returned by system
xpadr	First word address of FET extension for extended error processing
XLR	Extended label processing keyword
xladr	First word address of extended label processing buffer
xll	Length of extended label buffer

Examples:

To create a minimum FET for the standard INPUT file:

```
LBUFFER  EQU    65
INPUT    FILEC  BUFFER,LBUFFER
```

To create an FET for a binary random file:

```
LBUFFER  EQU    65
LINDEX   EQU    25
FILEABC  RFILEB BUFFER,LBUFFER,(IND=INDEX,LINDEX)
```

To create an FET for a labeled tape file with user processing at end-of-volume condition. OWNCODE routine is supplied:

```
LBUFA    EQU    65
TAPE1    FILEB  BUFA,LBUFA,LBL,UPR,(OWN=PROCEOR)
TAPE1    LABEL  SORTINPUTTAPE,32,90
```

To create an FET for a list file. OWNCODE routines are supplied and the working storage area is used:

```
LBUFB    EQU    65
PRINT    FILEC  BUFB,LBUFB,(WSA=LINE,14),(OWN=ENDING,ERRORS),UPR,EPR
```

FET FIELD DESCRIPTION

Words of the FET are numbered 1-13 in decimal, corresponding to the addresses lfn through lfn + 12 decimal. All parameter values are octal unless otherwise noted. Bits are numbered 0-59 right to left in decimal.

LOGICAL FILE NAME (lfn) (bits 18-59 at lfn)

The lfn field contains one to seven display-coded letters or digits starting with a letter, left justified; if less than seven are declared, unused characters are zero-filled. This field is used as common reference point by the central processor program and the peripheral processor input/output routines.

The lfn parameter declared in an FET creation macro is also used as the location symbol associated with the first word of the FET. A reference to lfn in the file action requests is a reference to the base address of the FET.

CODE AND STATUS (CS)(bits 0-17 at lfn)

The CS field is used for communication of requested functions and resulting status between the central processor program and the peripheral processor input/output routines. This field is set to the request code by CPC when a file action macro request is encountered. When the FET is generated, bits 2-17 should be zero.

The code and status bits have the following significance:

Bits 14-17 — Record level number. On skip and write record requests, this subfield is set by CPC as part of the function code. On read requests, it is set by CIO as part of the status when an end-of-record is read. Initially the level subfield is set to zero when the FET is generated.

Bits 9-13 — Status information upon request completion. Zero indicates normal completion. Nonzero indicates an abnormal condition, not necessarily an error; an OWNCODE routine, if present, will be executed. Status codes are described with the EOI OWNCODE and Error Exit Address discussions. Initially, this subfield is set to zero when the FET is generated.

Bits 0-8 — Used primarily to pass function codes to a peripheral processor. Function codes are even numbers (bit 0 has a zero value). They are listed as CIO codes below.

When the request has been processed, bit 0 is set to one. When the FET is generated, bit 0 must be set to one to indicate the file is not busy.

Bit 0 — Current status of request (0 = file being processed, 1 = request complete).

Bit 1 — Specifies the mode of the file (0 = coded, 1 = binary). Bit 1 is not altered by CPC when a request is issued.

Bits 2-8 — Pass function codes to a peripheral processor (file action requests).

Bits 3 and 4 — These bits will be set to binary 10 if end-of-record is read, or to binary 11 if end-of-partition is read.

CIO function codes listed below can be set in bits 0-8 of the CS field by the user before calling CIO to carry out the function. They are set by CPC when file action macros are used. All values are octal.

All codes indicated by – are illegal; all reserved codes are illegal. All codes are shown for coded mode operations; add 2 for binary mode. Example: 010 is coded READ, 012 is binary READ. Upon completion of operation, code/status in FET is changed to an odd number, usually by adding 1 to the code. In some cases, code is further modified to indicate manner in which operation concluded. Example: a READ function 010, at completion, becomes 011 (buffer full), 021 (end of system-logical-record), or 031 (end-of-partition).

General code meanings are:

200 series for special reads or writes (reverse, skip, non-stop, rewrite, etc.)

300 series for open and close

400 series reserved for CDC

500 series reserved for installations

600 series for skip

700 series reserved for CDC

Code	Function	Code	Function	Code	Function
000	RPHR†	104	OPEN/WRITE/NR	224	REWRITER
004	WPHR†	110	POSMF	234	REWRITEF
010	READ	114	EVICT	240	SKIPF
014	WRITE	120	OPEN/NR	250	READNS
020	READSKP	130	CLOSE/NR	260	READN†††
024	WRITER††	140	OPEN	264	WRITEN†††
034	WRITEF	144	OPEN/WRITE	300	OPEN/NR
040	BKSP	150	CLOSE	330	CLOSER
044	BKSPRU	160	OPEN	340	OPEN
050	REWIND	170	CLOSE/UNLOAD	350	CLOSER
060	UNLOAD	174	CLOSE/RETURN	370	CLOSER/UNLOAD
100	OPEN/NR	214	REWRITE	374	CLOSER/RETURN
				640	SKIPB

†Applies to SI tapes only.

††When a WRITER function is issued with level 17 specified, the function is changed to a WRITEF. Thus, a function issued as a 24 will return as a 34.

†††Applies to S and L tapes only.

DEVICE TYPE (dt)(bits 48-59 at lfn + 1)

The device type value will be returned to the FET device type field when a file action request is issued if FET length exceeds the minimum. The 6-bit device type will occupy bits 54-59; bits 48-53 will hold recording technique identification for magnetic tapes, if applicable. The mnemonic is used in the REQUEST control statement.

Mass storage devices have the following codes:

Request Mnemonic	Device Type	Device
–	01-05	Reserved for CDC
AM	06	841 Multiple Disk Drive
–	07-12	Reserved for CDC
AY	13	844-21 Disk Drive
AZ	14	844-41 Disk Drive
AH	15	819 Disk Drive
–	16-17	Reserved for CDC
AX	20	ECS resident files
–	21-25	Reserved for CDC
LM	26	Link medium file
–	27	Reserved for CDC
–	30-37	Reserved for installations, mass storage only

Magnetic tapes have the following codes:

Request Mnemonic	Device Type (Octal)		Recording Technique (Right 6 bits of FET dt Field in Binary)
MT	40	7-track magnetic tape	xxxx00 HI density 556 bpi xxxx01 LO density 200 bpi xxxx10 HY density 800 bpi xxxx11 Reserved for CDC xx00xx Unlabeled xx01xx SI standard U and Z labels xx10xx 3000 series label (Y) xx11xx Reserved for CDC 00xxxx SI data format 01xxxx Reserved for CDC 10xxxx S data format 11xxxx L data format
NT	41	9-track magnetic tape	xxxx00 Reserved for CDC xxxx01 GE density 6250 cpi xxxx10 HD density 800 cpi xxxx11 PE density 1600 cpi xx00xx Unlabeled xx01xx SI standard U label (ANSI) xx10xx 3000 series label (Y) xx11xx Reserved for CDC 00xxxx SI data format 01xxxx Reserved for CDC 10xxxx S data format 11xxxx L data format

Request Mnemonic	Device Type (Octal)		Recording Technique (Right 6 bits of FET dt Field in Binary)
–†	42	member multi-file set 7-track tape	Same as in MT
–†	43	member multi-file set 9-track tape	Same as in NT
–†	62	7-track multi-file set tape	Same as in MT
–†	63	9-track multi-file set tape	Same as in NT

Unit record devices have the following codes:

Request Mnemonic	Device Type (Octal)	Device
TR††	44	Paper tape reader
TP††	45	Paper tape punch
–	46-47	Reserved for installations
LP††	50	Any available line printer
–	51	Reserved for CDC
LQ††	52	512 line printer
LR††	53	580-12 line printer
LS††	54	580-16 line printer
LT††	55	580-20 line printer
–	56-57	Reserved for installations
CR††	60	405 card reader
KB	61	Remote terminal keyboard
–	64†††-65	Reserved for CDC
–	66-67	Reserved for installations
CP††	70	415 card punch
DS	71	6612 keyboard/display console
GC††	72	252-2 graphic console
HC††	73	253-2 hardcopy recorder
FM††	74	254-2 microfilm recorder
PL††	75	Plotter
–	76-77	Reserved for installations

†Code is generated when a tape is declared to have MF characteristics; the multi-file set code 62 or 63 is used only in system tables; it is not returned to the user's FET.

††Supporting software must be supplied by the installation.

†††Device code 64 cannot be assigned. REQUEST processing uses code 64 to indicate a tape file in the process of being assigned.

RANDOM ACCESS(R)(bit 47 at lfn + 1)

A one in the R field indicates a random access file. R may be set to 1 by using the RFILEB or RFILEC macro. When a file is opened or closed, the R setting determines action performed with regard to the index as shown below.

The index is that used by name/number index random files, not CYBER Record Manager.

OPEN	FET R=0	FET R=1
No index	No index action	FET R bit is set to zero.
Index	No index action	Index is read into index buffer; if index buffer is not specified, FET R bit is set to zero and a non-fatal diagnostic is sent to dayfile. The index buffer is zeroed out before the index is read.

If a non-existent file is opened, the value of the R bit is not altered. The index buffer specified in the FET is zeroed out.

CLOSE	FET R=0	FET R=1
File currently has index	File is flagged as not having index	If index buffer exists or previous operation was write, the index is written; and file is flagged as having index. If buffer is not specified, a nonfatal diagnostic occurs.
File currently has no index	No index action	If file is written while R=1 during this job, or if previous operation was write, the file is flagged as having an index and the index is written. If index buffer is not specified, a non-fatal diagnostic occurs.

The above actions are taken only if the contents have been altered since the file was last opened.

When any other file action request is issued, the r setting determines the access method to be used. If r = 0, the file is read or written beginning at the current location. If r = 1, the file is read or rewritten according to the logical disk address in FET word 7, or written at the end-of-information; and the logical disk address is returned to FET word 7.

RELEASE (N) (bit 46 at lfn + 1)

This bit is reserved for the operating system.

USER PROCESSING (UP) (bit 45 at lfn + 1)

The UP bit may be used to control tape end-of-volume and device set end-of-device processing. If the UP bit is zero, unit swapping is automatic without notification to the user; the function in process when end-of-volume or end-of-device is detected is completed on the next unit. If the UP bit is set to one, the user is notified when an end-of-volume or end-of-device condition arises. End-of-volume for tape files is defined as a tape mark followed by an EOV1 label for labeled tapes and SI format unlabeled tapes, or as the first tape mark after the EOT reflective spot for unlabeled S and L tapes. End-of-device for RMS files is defined by an overflow RBT word pair.

If the UP bit is set, end-of-volume and end-of-device status (02) is returned in bits 9-13 of the FET code and status field. Functions that do not transfer data from the circular buffer will have been completed; data transfer function may be re-issued as indicated by an examination of the buffer pointers. If CPC is in use, control is returned to the EOI OWNCODE routine if declared in bits 30-47 of lfn + 8. If a continuation volume or device is desired, a CLOSER function should be issued. If end of volume processing without a continuation volume is desired, a CLOSER/RETURN should be issued.

ERROR PROCESSING (EP) (bit 44 at lfn + 1)

The EP bit is set when the calling program is to be notified of error conditions arising from file actions. Error codes returned to the code and status field are listed under the error address field. Control is given to the user OWNCODE routine at error address when EP is set. If EP has not been set, the operator is informed of the error and must authorize job termination or continuance regardless of the error. The following errors cause control to be returned to the user when the EP bit is set:

CIO code not legal on this device

READ or SKIP forward function immediately follows WRITE function

FET buffer pointers out of bounds

READ attempted on a file without read permission

WRITE attempted on permanent file not positioned at end of information

Open function on an existing random indexed file with too small index buffer

REWRITE on permanent file without MODIFY permission

WRITE on permanent file without EXTEND permission

EVICT on permanent file

Device is full and overflow is not allowed

Parity error on an ECS resident file

Index error on an ECS resident file

Unrecovered RMS error

NO RECOVERY (EB) (bit 43 at lfn + 1)

This bit can be set to control error recovery. If it is set, no attempt will be made to recover errors encountered while reading data on magnetic tape.

MULTI-USER JOB (MUJ) (bit 42 at lfn + 1)

Set only when the file is being processed by a multi-user job. Currently, the EDITOR routine in INTERCOM is the only system-supplied multi-user job. When bit 42 is set, user id, user table address, and a special code (for routine 3TT) appear in lfn + 5.

EXTENDED LABEL PROCESSING (XL) (bit 41 at lfn + 1)

This bit affects processing of labels on magnetic tape. Format to be used in the label fields in lfn + 10 through lfn + 12 depends on this setting. Standard label processing of required labels occurs when XL=0. If XL >1, the user can process optional labels, as described under Tape Label Processing later in this section.

EXTENDED ERROR PROCESSING (XP) (bit 40 at lfn + 1)

The upper 12 bits of FET word 7 detail errors indicated by bits 9-13 of FET word 1 if the XP bit equals 1, as explained under FET Extension Pointer field. An error message is displayed on the B display and is written to the dayfile. If this bit is not set, the operator is informed of unrecovered errors and has the option of dropping or continuing the job.

The EP bit must be set before control can return to the user OWNCODE to process these errors. Also, the UP bit must be set to gain control at end-of-volume.

When XP is set, the FET extension pointer in word 7 must be set.

EC (bit 39 at lfn + 1) Reserved for operating system.

NON-STANDARD LABEL (NS) (bit 38 at lfn + 1)

Setting this bit to 1 indicates non-standard labels exist. All processing must be done by the user program. Non-standard labels are not supported on SI format tapes.

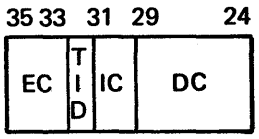
DISPOSITION CODE (bits 35-24 at lfn + 1)

The values shown below are returned to the FET disposition code field when a file action request is issued with the FET length greater than the minimum. A file with a special name automatically is assigned the corresponding disposition code value when the file is created.

Codes on LABEL or REQUEST control statements for tape files set these values in the FET:

Code	FET Value (Octal)	Disposition
CK	xx01	Checkpoint
IU	xx02	Inhibit automatic unload of tape
CI	xx03	Checkpoint and inhibit unload tape
SV	xx04	Inform operator to save tape
CS	xx05	Checkpoint and save tape

For rotating mass storage files, bits 35-24 of lfn + 1 are divided into four fields. The user cannot alter file disposition by changing this field. Rather, the DISPOSE or ROUTE control statement or macro must be used.



EC (bits 35-33) External characteristics:

Code	FET Value (Binary)	Description	Special File Name
Default/default	000	Default print train/default punch character set	OUTPUT/PUNCH
– – /EC=SB	001	Reserved/punch standard binary	– – /PUNCHB
EC=A4/EC=80COL	010	ASCII 48-character print train/punch free-form binary	– – /P80C
EC=B4/ – –	011	BCD 48-character print train/reserved	– – / – –
EC=B6/EC=026	100	BCD 64-character print train/punch 026	– – / – –
EC=A6/EC=029	101	ASCII 64-character print train/punch 029	– – / – –
EC=A9/EC=ASCII	110	ASCII 96-character print train/punch ASCII	– – / – –
– – / – –	111	Reserved for installations	

TID (bit 32) Terminal identifier which applies only to local files, not queue files:

Code	FET Value (Binary)	Description
TID=C	1	Ignore remote ID in file routine
TID=id	0	Route file to remote user with terminal identification id

IC (bits 31-30) Internal:

Code	FET Value (Binary)	Description	Special File Name
IC=DIS	00	File format is display code	OUTPUT/PUNCH
IC=ASCII	01	File format is ASCII	----/----
IC=BIN	10	File format is binary	----/PUNCHB,P80C
--	11	Reserved	----/----

DC (bits 29-24) Disposition code:

Code	FET Value (Octal)	Description	Special File Name
--	01	Reserved	
--	02	Reserved	
--	03	Reserved	
--	04	Job ready for scheduling	
--	05	Job has tape requirements	
--	06	Job has tape requirements with VSN information	
--	07	Reserved	
PU	10	Punch	PUNCH,PUNCHB,P80C
FR†	20	Film print	FILMPR†
FL†	22	Film plot	FILMPL†
HR†	24	Hardcopy print	HARDPR†
HL†	30	Plot	PLOT†
PR	40	Print on any available printer	OUTPUT
P2	42	Print on 512 line printer	
LR	43	Print on 580-12 line printer	
LS	44	Print on 580-16 line printer	
LT	45	Print on 580-20 line printer	

LENGTH OF FET (bits 18-23 at lfn + 1)

The system FET length is determined as follows: FET first word address + 5 + lgth = last word address + 1. The minimum FET length is five words (lgth=0). If the minimum FET is used, only the logical file name, code and status field, FIRST, IN, OUT, and LIMIT are relevant; other fields are not checked by the operating system. An FET of six words (lgth+1) is used if a working storage area is needed for blocking/deblocking. An FET of eight words (lgth+3) is used if the r bit is set, indicating an indexed file. Length is nine words (lgth=4), if OWNCODE routines are declared.

†Supporting software must be supplied by the installation.

FNT POINTER (bits 48-59 at lfn + 4)

The FNT pointer is set by the operating system, upon return from a file action request, to the location of the file entry in the file name table. The pointer is placed in the FET to minimize table search time and does not affect the program. In the case of a minimum FET, CPCIO updates the pointer; PPCIO does not.

RECORD BLOCK SIZE (bits 34-47 at lfn + 4)

If the file resides on an allocatable device, the size of the device record block is returned in this field when the file is opened. It is given as the number of physical record units in a record block. If the number of PRUs is not defined or is variable, the field is set to zero. Record block size is not returned if a minimum FET is used.

PHYSICAL RECORD UNIT SIZE (PRU) (bits 18-33 at lfn + 4)

The physical record unit size of the device to which the file is assigned is returned in this field when a file is opened. It is given as the number of central memory words. The PRU size is used by CPC to determine when to issue a physical read or write. PRU size will not be returned if a minimum FET is used.

FIRST, IN, OUT, LIMIT (bits 0-17 at lfn + 1 through lfn + 4)

The fields contain the beginning address (FIRST) and last word address + 1 (LIMIT) which define the file circular buffer. The IN and OUT pointers indicate the address of data placed into or removed from the buffer. System and programmer use of these fields is discussed under the heading Circular Buffer Use.

WORKING STORAGE AREA (WSA) (lfn + 5)

The two fields in this word of the FET specify the first word address (bits 30-47) and last word address + 1 (bits 0-17) of a secondary buffer within the program field length. The area is needed to use the system macros READIN and WRITOUT, which blocks or deblocks records from the area into the circular buffer. READIN and WRITOUT relieve the user of responsibility for circular buffer pointer manipulation.

DETAIL ERROR CODE (bits 48-59 at lfn + 6)

When the XP bit is set to 1, this field contains extended tape error processing codes which give additional detail of abnormal conditions resulting from the last input/output operation. The user is responsible for clearing this field after reading it.

Codes 1-77 (octal) are considered software warnings to the user; they are not results of hardware failures. The tape related codes and subsequent software warnings are as follows:

Error Codes (Octal)	Software Warning
24	Read error in opposite mode
25	Function not complete
27	Record fragment possible

Error Codes (Octal)	Software Warning
30	Data read exceeds MLRS/PRU size
31	Multi-file set ill-formed
32	Write attempt on protected volume
33	Write at 200 bpi not allowed on 66X tape drive
35	Multi-file name not found on multi-file device
36	Next volume unknown
37	File not allowed on assigned device

Codes 100-177 (octal) are considered cases where the tape unit has lost position. These codes are as follows:

Error Codes (Octal)	Position
100	Position uncertain – data intact
101	Position uncertain – data destroyed
102	Physical/logical positions disagree
103	Position uncertain – ready dropped during last operation

Codes 200-277 (octal) are considered unit oriented errors. Switching physical tape devices allows the program to continue after repositioning. These codes and subsequent errors are as follows:

Error Codes (Octal)	Unit
200	System error – tape table
201	Hardware – unit hung busy
202	Hardware – no end of operation
203	Hardware density change during I/O
204	Unit reserved by another buffer controller
205	Loop fault
206	Unable to read tape just written
207	Marginal transport indication
210	Lost data
211	Multiple load points on tape
212	No read after write
213	Coldstart
214	Irrecoverable write reposition error
215	Attempt to use downed unit

Codes 400-477 (octal) are errors resulting from hardware failure between the PPU and the physical tape unit. These codes and subsequent errors are as follows:

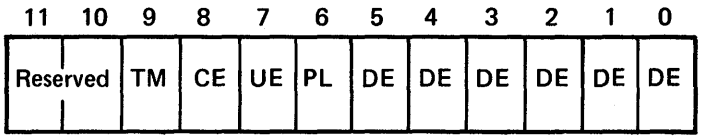
Error Code (Octal)	Data Path Error
400	Hardware – 6681 or 6683 malfunction
401	Hardware – MMTC memory parity error
402	Hardware – 6681 failed, no data on IAN
403	Hardware – transmission parity error
404	System error

Codes 1000-1005 (octal) are errors resulting from a bad tape. These codes and subsequent errors are as follows:

Error Codes (Octal)	Tape (Medium)
1000	Tape parity error
1001	25 feet erased tape
1002	Blank tape read
1003	Incomplete erasure of tape bad spot
1004	Noise in IRG
1005	Erase limit reached

Codes 6000-7777 (octal) are reserved for installations.

Codes are combined meanings of the following bits:



TM Tape medium

CE Controller error (controller, 6681, etc.)

UE Unit caused error

PL Position lost

DE Detailed error

The references to system noise record and last good record refer to procedures the system follows in recovery attempts.

Detailed error codes allow a central processor program to take appropriate action when a non-user caused error occurs. For example, the message UBC IN FET TOO LARGE does not have a detailed error code because it is a user caused error. On the other hand, the message TAPE PARITY ERROR is assigned to a detailed error code because the condition is an external caused error.

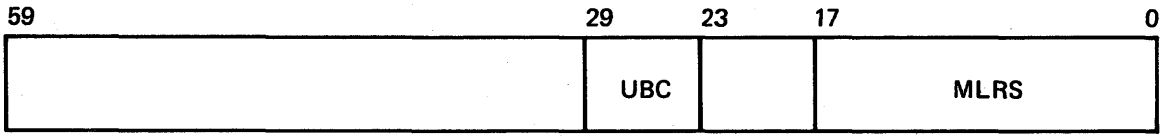
FET EXTENSION POINTER (bits 30-47 at lfn + 6)

When the XP bit is set, pointer is the required address of an FET extension. Currently, the extension is limited to a single word, but the length (L) parameter anticipates future expansion.

UNUSED BIT COUNT (UBC) (bits 24-29 of lfn + 6)

The unused bit count field is used only for files in S or L tape format. (If the device type is not magnetic tape, this word will contain indexing information). It is used for communication between the peripheral processor input/output routines and the user program.

For magnetic tapes with S or L data format, the structure of the word at lfn + 6 is:



For a READ or READSKP function, the operating system will store into this field the number of low-order unused bits in the last data word of the record. The UBC field is not used during a READN request. For a WRITE, WRITER or WRITEF function, the operating system will read the contents of UBC and adjust the length of the record accordingly. The operating system does not use the UBC field during a WRITEN request.

For example, to write a single record of 164 decimal characters, the data length is 17, to the next highest CM word. The number of low-order unused bits in the last word would be 36. The user would set UBC = 36, set IN and OUT pointers to reflect 17 words of data, and then issue a WRITE or a WRITER.

For 7-track tape, the UBC may range from 0 to 59, but will always be a multiple of 12 when set as a result of a read operation. If it is not a multiple of 12 for a write request, the operating system will truncate the value to the nearest multiple of 12; if UBC is 18, the operating system will execute as though it were 12, and if UBC is 6, the operating system will execute as though it were 0. The field in the FET remains unchanged.

For 9-track conversion mode tape, each 6-bit character in memory is converted to an 8-bit character on tape. The UBC is set to allow an integral number of characters to be written or read. The UBC is set to a multiple of 6 bits as a result of a read operation. For a write request, the operating system will truncate the value to the nearest multiple of 6. If the UBC is 19, the operating system will execute as if it were 18. The field in the FET remains unchanged.

For 9-track packed mode tape, four 6-bit characters in central memory are written as three 8-bit characters on tape; two central memory words are 15 tape characters. On a read, the UBC is set after an integral number of characters have been read from tape. If 3839 tape characters are read, 512 words are put in the buffer and the UBC is set to 8. If 511 words are written to tape, the operating system executes as if the buffer contains 512 words and the UBC is 56. The fields in the FET remain unchanged.

MAXIMUM LOGICAL RECORD SIZE (MLRS) (bits 0-23 of lfn + 6)

The MLRS field is applicable only for S or L format magnetic tape files. It defines the size of the largest physical record to be encountered when the S or L tape format is used. The size is given in number of central memory words.

For S tape format, if MLRS = 0, the value of the maximum PRU is assumed to be 512 words. For L tape format, if MLRS = 0, the assumed maximum PRU is LIMIT − FIRST − 1 for standard reads, and LIMIT − FIRST − 2 for READN.

Since S and L tapes record size is defined in characters, instead of central memory words, the last word may contain invalid data. Consequently, UBC is required to attest to the validity of all characters in this word.

RECORD REQUEST/RETURN INFORMATION (bits 0-29 of lfn + 6)

If the file resides on a mass storage device and has the r bit set in word 2, indexing information appears in words 7 and 8 for communication between the peripheral processor input/output routines and the user program.

The record request/return information field is set to zero when the FET is generated. Both the indexing functions and the peripheral processor input/output routines set the field during random file processing.

For other than the operating system indexing method, the following information is pertinent. At the start of writing a new system-logical-record, if the random access bit and the record request/return information field are non-zero, the latter field is assumed to contain the address of a location within an index. The PP routine inserts into that location (in bits 0-23) the PRU ordinal (starting from 1) of the system-logical-record. To read the record again, the random access bit should be set to non-zero and the PRU ordinal should be entered in the FET in the record request/return information field.

RECORD NUMBER (bits 36-59 at lfn + 7)

When an indexed file is processed, this field contains the ordinal of a record identified in the index. Records are numbered beginning with 1.

INDEX LENGTH (bits 18-35 at lfn + 7)

When an indexed file is processed, this field contains the number of words in the index. One word for each numbered record, or two words for each named record, plus a one-word header is required.

INDEX ADDRESS (bits 0-17 at lfn + 7)

This field contains the address of the index for a name or number index file.

EOI OWNCODE ADDRESS (bits 30-47 of lfn + 8)

This field contains the address of a user supplied OWNCODE routine to be entered when end-of-information, end-of-device, or end-of-volume status is encountered during magnetic tape or device set processing. The UP bit must be set if user end-of-volume or end-of-device processing is desired. If an OWNCODE address is specified, CPC enters this routine when end-of-information is encountered regardless of the setting of the UP bit.

CPC enters this routine when bits 9-13 of the code and status field is:

01	End-of-information encountered after forward operation
02	End-of-volume reached during magnetic tape forward operation
02	End-of-device reached during device set processing

Just before entering an end-of-information OWNCODE routine. CPC zeros bits 9 and 10 of the first word of the FET. However, as the routine is entered, register X1 still contains the first word of the FET as it appeared before those two bits were zeroed.

ERROR EXIT ADDRESS (bits 0-17 of lfn + 8)

This field specifies an address to receive control if an error condition occurs after a file action request. The EP bit must be set to cause control to pass to this OWNCODE address. The FET code and status field will reflect the error condition. If processing can continue, the error routine should exit through its entry point; otherwise, an abort request may be issued. If the error address field is zero, the run continues normally. The FET code and status bits reflect the error condition upon normal return to the program.

CS Bits 9-13 (Octal)	Meaning
04	Irrecoverable parity error on last operation; or lost data on write. Unrecovered error other than device capacity exceeded on last magnetic tape operation.
10	During a magnetic tape read, the physical record size exceeded circular buffer or maximum allowable PRU size (MLRS for S and L tapes). Such magnetic tape error is termed device capacity exceeded. During a mass storage write, all mass storage space meeting the file requirements was in use or otherwise unavailable.
20	Additional error status retumed.
21	End of multi-file set. File position number is greater than that of the last member in the set. Any subsequent attempt to reference the logical file name assigned to the nonexistent member will result in a fatal error.
22	Fatal error.
23	Index full.
24	Interlock broken for shared rotating mass storage devices.
25	Attempt made to read or write record number n of a random file, but n exceeds index size.
26	Attempt made to read named record from random file, but name does not appear in index.
27	Attempt made to write named record on random file, but name does not appear in index, and index is full.
30	Function legal but not defined on device.
31	Permanent file permission not granted.
32	Function legal except for permanent files
33	No public set has the required attributes.
34-37	Reserved for future use.

If both EOI and error routine execution are needed, the error routine is executed. Just before entering an error OWNCODE routine, CPC zeros bits 11-13 of the first word of the FET. However, as the routine is entered, register X1 contains the first word of the FET as it appeared before those bits were zeroed.

LABEL PARAMETERS (lfn + 9 through lfn + 12)

Words 10-13 of the FET may contain information pertaining to magnetic tape labels. The format and use of these fields depends on the setting of the extended label processing bit in word 2. The LABEL macro generates fields for normal label processing. Further details appear under the Tape Label Processing heading.

Parameters in these fields must be display code. If other than the LABEL macro is used to create them, display code zero may be used to add leading zeros to numeric fields. Character fields, which are left justified, may be display code blank filled.

RESIDUAL SKIP COUNT (RSC) (bits 24-41 at P + 0)

When XP is set and P is the address of the FET extension word, RSC is the residual skip count. If SKIPF, SKIPB, or READSKP functions do not complete the specified number of skips, the count of records yet to be skipped is returned here. RSC will have a value when SKIPB encounters beginning-of-information even when the UP bit is not set. If SKIPF terminates at end-of-volume because UP is set, RSC will be set.

PERM BITS (bits 20-23 of P + 0)

The setting of these bits will duplicate that of the permanent file permission bits in the file name table. Permission is granted when the bit indicated is set.

Bit	Meaning
20	Read permission
21	Extend permission
22	Modify permission
23	Control permission

These bits are set when the user issues an OPEN function.

EXTENSION LENGTH (bits 0-17 at P + 0)

The length of the extension, including word P, is required. This value must be 1.

CIRCULAR BUFFER USE

For each file, the user must provide one buffer, of any length greater than a PRU size. The buffer is called circular because it is filled and emptied as if it were a cylindrical surface in which the highest addressed location is immediately followed by the lowest. The FET fields FIRST, IN, OUT and LIMIT control movement of data to and from the circular buffer.

Data is transmitted in physical record units; their size is determined by the hardware device. For example, rotating mass storage has an inherent PRU size of 64 CM words; binary mode magnetic tape files in SI format are assigned a PRU size of 512 words.

FIRST and LIMIT never vary during an I/O operation; they permanently indicate buffer limits to the user and the operating system.

The program that puts data into the buffer varies IN, and the program that takes it out varies OUT. During reading, the operating system varies IN as it fills the buffer; and the user varies OUT as he removes data from the buffer. During writing, the user varies IN as he fills the buffer with data; and the system varies OUT as it removes data from the buffer and writes it out.

The user cannot vary IN or OUT automatically except when using READIN and WRITOUT functions. To change these pointers within the program a new value is inserted into lfn + 2 (IN) or lfn + 3 (OUT). For convenience, the words containing IN and OUT contain no other items, eliminating the need for a masking operation. The system dynamically checks the values of IN and OUT during data transfers, making continuous read or write possible.

If IN = OUT, the buffer is empty; this is the initial condition. If IN>OUT, the area from OUT to IN − 1 contains available data. If OUT>IN, the area from OUT to LIMIT − 1 contains the first part of the available data, and the area from FIRST to IN − 1 contains the balance.

To begin buffering, a READ function may be issued. One or more PRUs of data are put into the buffer beginning at IN, resetting IN to one more than the address of the last word filled after each PRU is read. Data may be processed from the buffer beginning with the word at OUT, and going as far as necessary, but not beyond IN − 1. The user must then set OUT to one more than the address of the last word taken from the buffer. He sets OUT = IN to indicate that the buffer is empty.

When a READ macro request is issued, if the buffer is inactive and a read is not in process, CPC determines how much free space is in the buffer. If OUT>IN, OUT − IN words are free. If IN>OUT, (LIMIT − IN) + (OUT − FIRST) words are free. The system subtracts 1 from the number of free words, because it never must fill the last word since it would result in IN=OUT and give a false empty buffer condition. If the number of free words minus 1 is less than the PRU size, CPC does not issue a physical read request; control is returned normally.

The example below illustrates the use of IN and OUT pointers. Speed of operation is not considered; simultaneous processing and physical I/O are not attempted. The initial buffer pointer position is:

```
FIRST = BCBUF
IN = BCBUF
OUT = BCBUF
LIMIT = BCBUF + 500
```

The user issues a READ with recall request. Ignoring the possibilities of an end-of-partition, the system reads as many PRUs as possible (if PRU size is 64 words, 7 x 64 = 448 words) and leaves the pointers:

```
FIRST = BCBUF
IN = BCBUF + 448
OUT = BCBUF
LIMIT = BCBUF + 500
```

The user is processing items of 110 words. He takes four items from the buffer, leaving the pointers:

```
FIRST = BCBUF
IN = BCBUF + 448
OUT = BCBUF + 440
LIMIT = BCBUF + 500
```

The user issues another READ request since the buffer does not contain a complete item. The system is aware that IN>OUT, so that vacant space is LIMIT – IN + OUT – FIRST = 492 words; since it must not fill the last word, it must read fewer than 492 words.

The nearest lower multiple of 64 is 7 x 64 = 448, so the system reads 52 words into IN through LIMIT – 1, and then 396 more words into FIRST through FIRST + 395. It then resets IN so that the pointers look like:

```
FIRST = BCBUF
IN = BCBUF + 396
OUT = BCBUF + 440
LIMIT = BCBUF + 500
```

The system has just used the circular feature of the buffer; now the user must do so. The next time he wants an item, he takes the first 60 words from OUT through LIMIT – 1, and the remaining 50 from FIRST through FIRST + 49. Then he resets OUT, making the pointers:

```
FIRST = BCBUF
IN = BCBUF + 396
OUT = BCBUF + 50
LIMIT = BCBUF + 500
```

On input, this can continue indefinitely, with OUT following IN, around the buffer. The system stops on encountering an end-of-record or end-of-partition, and sets the code and status bits accordingly. The system may, or may not, have read data before the end-of-record; so it is up to the user to examine the pointers and/or process the data before taking end-of-record or end-of-file action.

In writing, the process is similar, but the roles are reversed. The user puts information into the buffer and resets IN; and when he calls the system, it removes information from the buffer and resets OUT. For writing, the system removes data in physical record units and empties the buffer if possible. The user must be careful not to overfill the buffer; IN must not become equal to OUT. During the process of emptying the buffer, the operating system resets OUT after each PRU has been written and checked for errors.

ESTABLISHING OWNCODE ROUTINES

The EOI address and error address fields in word 9 of the FET define user supplied routines. CPC calls these routines when the UP or EP bits are set.

An OWNCODE routine should be set up like a closed subroutine with execution beginning in the second word of the routine. CPC calls an OWNCODE routine by copying the exit word of CPC into the first word of the OWNCODE routine, putting the contents of the first word of the FET into register X1, and branching to the second word of the OWNCODE routine. If an unrecovered RMS error occurred, the FET pointers are left positioned after the last good write operation and the file positioned after the bad PRU.

Termination of an OWNCODE routine by a branch to its first word causes a branch to the point in the program to which CPC would have returned if the OWNCODE routine had not been called.

Although CPC clears status bits in the first word of the FET before the OWNCODE routine is called, the contents of this word can be examined in register X1. All registers used in the main program except A1, X1, A6, and X6 are saved and restored by CPC.

TAPE LABEL PROCESSING

The label processing that occurs for magnetic tapes is indicated by the XL bit setting, bit 41 of the second word of the FET. Extended label processing is possible only when this bit is set. An explicit open is required.

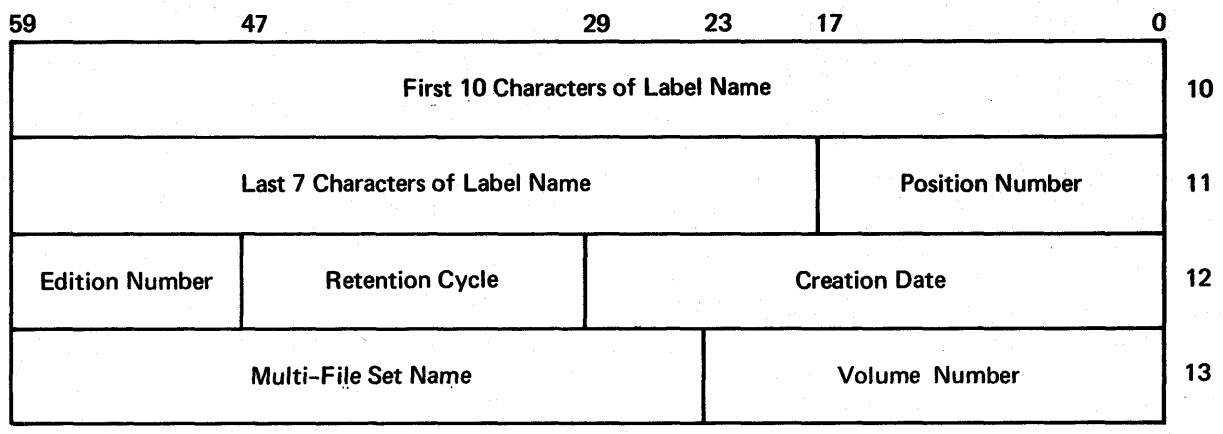
When the bit is off, the system generates output data and checks input data only for required ANSI, Z format, and Y (3000 series) format labels. Labels that are processed by standard processing (excluding Y labels) are label types VOL1, HDR1, EOF1, and EOV1. Default values are written if the user does not specify otherwise.

Checking of the VOL1 label of ANSI or Z formats ensures that the VSN requested for the job is the one assigned.

STANDARD LABEL PROCESSING

Only standard labels are processed when the XL bit is off. Any existing optional labels will be ignored.

If the FET for the file is at least 13 words long, words 10-13 hold file header label data in the following format:



When input tapes are read, any user information in these fields is compared with that written in the HDR1 label on the tape before the file is opened. A discrepancy in a label field stops job processing until the operator takes action to continue it. If a field is not specified in the FET, any value on the tape HDR1 label is accepted. This checking cannot be done with an FET less than 13 words, but any labeled tape will be accepted for processing.

When output tapes are opened, any information in words 10-13 is used to create the HDR1 label for the file. Otherwise, default values are written. If two OPEN functions with rewind are performed, the system retains the information written the first time. Thus, a label area supplies the label information regardless of which programs run afterwards.

LABEL MACRO FOR FET FIELDS

Fields in words 10-13 of the FET can be set for standard label processing by means of the LABEL macro. This macro must follow immediately the macro creating the FET to which it pertains. The LABEL macro generates data for VOL1 and HDR1 labels but does not directly cause any action on the file.

```
lfn    LABEL    labname,ed,ret,create,vol,mfn,pos
```

lfn	Logical file name used in FET creating macro.
labname	Label name or file identification of 1-17 characters; default is 17 blank characters.
ed	Edition number specifying file version of 1-2 decimal digits; default is 01.
ret	Retention indicator indicating the 1-3 digit decimal number of days the file is to be protected against accidental destruction; default is installation parameter.
create	Creation date in format of 2 digits for year and 3 digits for day(yyddd); default is current date.
vol	1-4 decimal digits indicating volume within a multi-volume set; default is 0001.
mfn	Multi-file name of 1-6 characters indicating the set to which lfn belongs; default is binary zero.
pos	Position number of 1-3 decimal digits indicating position of file lfn in multi-file set mfn; default is 000.

The macro expansion results in display code values with binary zero fill for all parameters given. If a parameter is absent from the macro, it is binary zero filled. Character fields are left justified; numeric fields are right justified.

When a file header label is written subsequently using the FET fields, default values are assigned for any field containing binary zero. On the tape, character fields are display code blank filled and numeric fields are display code zero filled. The fields, as written on the tape, are returned to the FET.

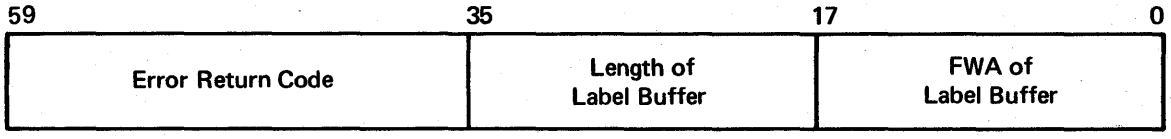
When the information in the FET is used to check existing labels, binary zero fill characters will be converted to the display code blank appropriate for character fields or display code zero for numeric fields before comparison is made. Fields in the FET containing all binary zeros are not compared. Checking procedures compare fields in the FET with those on the tape; not all fields in the FET need be specified; neither must the FET contain a value for all fields written on the tape.

If the header label on the tape mounted does not match the FET fields, the operator can attempt to locate the correct tape and assign it to the job, or accept the mounted tape with non-matching label fields. If the mounted tape is accepted, the values returned to the FET will reflect the header label on that tape.

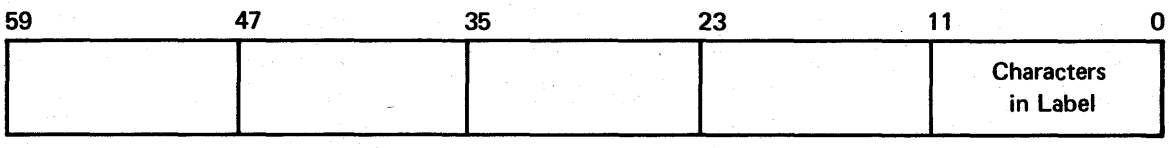
EXTENDED LABEL PROCESSING

When the XL bit is set, a user label buffer, rather than the FET, is used to hold labels for processing. The system processes the required labels, and the user may process optional labels in the buffer.

Buffer location must be defined in word 10 of the file FET as follows:



Within the buffer, each label must be preceded by a status word.



Only bits 0-11 should be set by the user to show the number of characters in the label. Remaining fields are set and used by the label processor. The last label should be followed by the status word containing zeros in bits 0-11.

Each label in the buffer appears, in display code, with the same format it has on the tape. Specific label field characteristics are discussed with Tape Labels in section 3.

When input tapes are read, the label processor searches the buffer for a HDR1 label. Any label information in the buffer is compared with that on the tape; any differences will require operator action. The system validates only the HDR1 label; other labels are the user's responsibility. If a HDR1 label in the buffer contains binary zero in any field, no label checking is done on that field. After an OPEN function is issued, all labels read by the system are delivered to the buffer, beginning with VOL1.

When output tapes are generated, any user labels to be written must be present in the label buffer when an OPEN or CLOSE function is issued. The buffer may, but need not, include the system required labels. The operating system will generate the required labels if they are not present in the label buffer. VOL1 labels in the label buffer will be ignored; HDR1 labels in the label buffer will be used if they are appropriate at that point in file processing. EOF1 or EOV1 labels in the label buffer will be used if they are present when the CLOSE function is issued.

For multi-file set processing with the XL bit set and calls to the COMPASS macro POSMF, word 10 of the FET must point to a label buffer. One of the first entries in the buffer must be a formatted HDR1 label with the multi-file name in the set identifier field. The position number field in the label has 4 digits; a position number of 9999 is required to write a label. Labels are always written at the end of all existing files in the multi-file set.

COMPASS INTERFACE WITH OPERATING SYSTEM 7

USER/SYSTEM COMMUNICATION

A user program can request action by another part of the operating system in several ways:

A CYBER Record Manager macro can be called to create or manipulate a file. This results in a call to other operating system functions.

A file action macro can be called. This results in a call to CPC (central program control) which posts a request in RA+1 to communicate with Monitor.

The system communication routine SYS= can be called through various macros.

Central processor subroutine CPC can be called through a return jump instruction. CPC then communicates with Monitor.

A request for PP program execution or system action can be placed in location RA+1 of the user field length to communicate directly with Monitor.

These requests are necessary to perform all file action such as opening, closing, reading, or writing a file, in addition to receiving information such as current time or date from the system.

BASIC COMMUNICATION: RA+1 REQUESTS

All requests from the user program to the system are made through RA+1 of the user program, which is initialized to zero. The system Monitor frequently examines RA+1 during program execution. If RA+1 is not zero, Monitor assumes that the contents are a request for a PP program or a system action, and initiates request processing. Executing an XJ instruction immediately after setting RA+1 non-zero speeds up processing. Bit 59 of RA+66 is set if the XJ hardware is available. When Monitor processing is complete, RA+1 is reset to zero. The requests to Monitor must be in the general format:

Bit 42-59	3 display code characters of a PP program name.
Bit 40	1 if automatic recall is requested. With automatic recall, control is not returned to the calling program until the request is executed. If automatic recall is not requested, the user program must determine whether or not the request is complete by checking a status word.
Bit 36-39	Zero.
Bit 0-35	Parameters that are required by the particular function being requested.

The user has the option of setting RA+1 directly, or calling a system or file action macro that sets it. If the user sets it directly, the format must conform to that shown above.

When Monitor accepts the request, it fills location RA+1 with zeros. For all requests except RCL, TIM, ABT, or END, the zero means only that Monitor has accepted the request and has no relation to whether the requested

task is complete. A user program posts an RA+1 request, then loops until that location is zero, before proceeding with other code. The user should make sure that RA+1 is clear before issuing a request.

Task completion normally is noted by the change of bit 0 in a status word from 0 to 1. The address of the status word must be greater than RA+1 and less than RA+FL. For requests made with automatic recall, the complete status bit is always set to 1 before control returns to the program, as explained below. Bits 0-17 of the RA+1 request points to the status word. For file action requests, this status word is the first word of the FET for that file.

RECALL CONCEPT

A recall request issued in a program causes the central processor assigned to that job to be relinquished temporarily. The length of time that the job leaves the processor depends on whether periodic or automatic recall was requested. During the amount of elapsed time before the job is reassigned, the central processor is also dependent on the relative priority of the job in the system, but jobs in recall are among the first considered by the scheduling routines.

Periodic recall puts a job in recall status for a short period of time that depends on other Monitor activity. Monitor reschedules the job when any PP program has completed processing for that job.

Automatic recall (auto recall) causes the job to relinquish control of the central processor for the time required to execute a request peripheral processor or Monitor function. The job remains in recall until after Monitor detects a status bit change to a word in the user field length which is set when the peripheral processor completes its task. For file action requests, the complete bit is bit 0 of the first word of the FET for the file. For other request, the address of the status word is specified by the user, and must be greater than RA+1 and less than RA+FL.

With programs using recall whenever appropriate, central processor time for a job is minimized and overall system central processor use is improved. If a program cannot proceed until a requested task is complete, it can allow Monitor to assign the central processor to another job until such time as the task is complete. Recall is particularly useful when input/output tasks are considered. A programmer can request recall in four ways:

RCL request to Monitor through program location RA+1

PP program call in RA+1 with recall bit set

RECALL macro request

File action macro with recall parameter. Any non-blank character establishes the recall parameter. R or RECALL can, but need not, be used.

Central processor programs can post an RA+1 request with the display code characters RCL in bits 42-59 and obtain periodic or auto recall depending on the remainder of the request. Periodic recall results from RCL in bits 42-59 with bits 0-41 containing all zeros. Automatic recall is obtained with bit 40 set to 1 and bits 0-17 containing an address of a word in the user field length which has 0 in bit 0. A PP program is expected to set bit 0 of the parameter word to 1 when its task is complete. If bit 0 is set to 1 when the RA+1 request is posted, Monitor makes RCL a non-operation. For other (PP) RA+1 calls with recall, PP Monitor checks the completion bit after the job is in auto-recall.

The RECALL macro results in periodic recall when no parameter list is given with the macro. If a file name is specified, automatic recall is produced. No separate status word is involved with periodic recall. The user program must check the code and status field of the FET for complete status to determine whether program execution can continue. The RECALL macro will not exit to OWNCODE routines.

When file action macros are used, automatic recall is requested by a recall parameter. Any non-blank character or string of characters can appear as this parameter. The characters RECALL are often used, but a single arbitrary character is sufficient.

The recall parameter can be specified for all the read and write macros except READIN and WRITOUT. However, the internal execution of these two macros ensures that automatic recall is always in effect.

USING CPC

Before CPC can honor a file action request, the file environment table (FET) must have been established for the file to be processed. Calling sequences to CPC can be generated either directly or through the use of system macro statements.

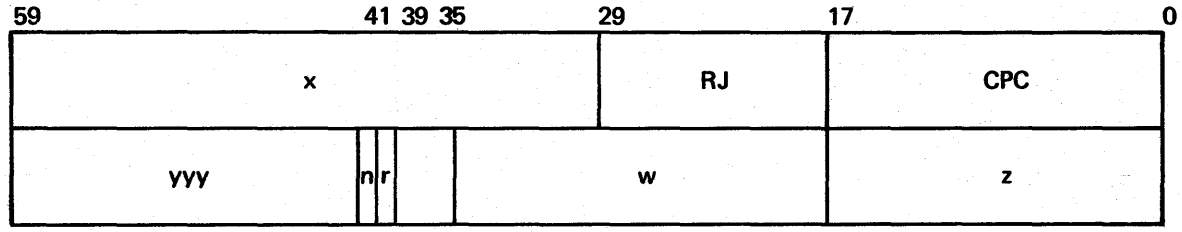
The user communicates with CPC through macro requests and the FET. Communication with the operating system is handled by CPC through setting and checking RA+1. CPC may also cause the execution of one or more user OWNCODE subroutines for which addresses are specified in word 9 of the FET.

A normal exit is made from CPC if the request is honored and no error condition occurs. Register X1 contains zero upon exit. If the status is other than request completed, register X1 contains the code and status bits set in the FET before the OWNCODE routine was entered.

Automatic recall should be used when the program makes an I/O or system action request but cannot proceed until that request is satisfied. Control is not returned to the program until that request is satisfied. Periodic recall can be used when the program is waiting for any one of several requests to be satisfied. In this case, the program is activated periodically so that the user can determine whether or not the program can proceed.

CALLING SEQUENCE TO CPC

Format of the calling sequence to the CPC subroutine:



RJ	Return jump instruction
CPC	Entry point to the CPC subroutine
r	Set if auto recall requested

If n=0 indicating a file action request:

yyy	Display code characters CIO, or
000001	Only file recall is desired. Display code characters RCL are generated in RA+1. OWNCODE routines are executed if appropriate.
000002	For most read or write functions. A function in progress is not reissued by CPC. When the file becomes inactive. CPC issues the next request, Display code characters CIO are generated.

000003 For all other functions. When the file becomes inactive, CPC issues the request. Display-code characters CIO are generated.

000004 or 000007 Equivalent to 000003; included only for compatibility with previous systems.

x SA1 base address of FET

z Request code (one of the CIO codes listed in section 5).

w Skip count for SKIPF, SKIPB, and BKSPRU; otherwise ignored.

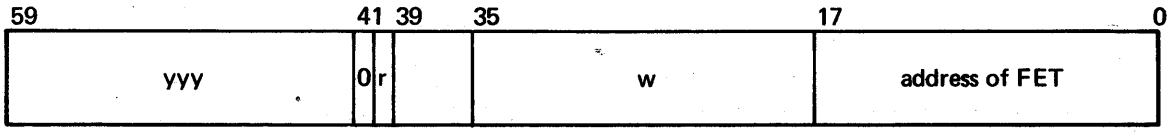
If n=1 indicating other than a file action request:

yyy Display-coded name of the called PP program

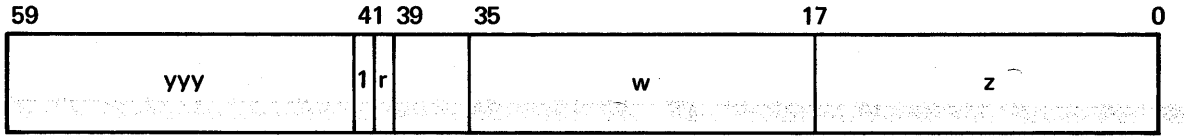
x Not relevant

z, w Parameters as required

For file action requests, CPC places the CIO function request code in the code and status field of the FET before writing the request in RA+1. Bits not specified in the calling sequence are reserved for future system use. A file action request to Monitor is formatted by CPC in RA+1 as follows:



A system action request to Monitor is formatted in RA+1 as follows:



CPC EXECUTION

Bit 41 of word 2 is set to 1 in the calling sequence of all requests except file action requests. This bit is actually a flag for CPC and has no relevance to either Monitor or the processing PP program. The setting of bit 41 causes CPC to recognize that the address given in A1 is not relevant, and that the word following the return jump to CPC contains a properly formatted request. No additional processing is done on these requests, except for MESSAGE. The request is simply placed in RA+1.

A request which utilizes an FET is signalled by a value of zero in bit 41 of word 2 of the calling sequence. CPC will in this case, do considerable processing for the user. The processing basically consists of three steps: wait until the FET is inactive; process any abnormal conditions; and initiate the new request. The high order 18 bits of word 2 in the calling sequence may contain a numerical value rather than a PP program name. These values are of the form 2X + Y, where X represents the ordinal in a table of PP program names, and Y is 1 or 0 to indicate whether or not the FET must be inactive before processing can continue. If a PP program name appears in these 18 bits, CPC waits for inactive FET status before initiating the new request.

1. Upon receipt of a file action request, CPC waits for previous activity on the specified FET to be completed unless the Y bit is zero; CPC requests automatic recall until FET word 1 contains an odd value. The Y bit is zero for READ, WRITE, and OPEN requests. If the request is OPEN, the assumption is made that no previous activity has occurred. READ and WRITE are handled specially.

2. If the Y bit is one, the results of the previous operation are tested. A zero in bits 9-13 of the FET code and status field indicates there are no abnormal conditions and processing goes to step 3. However, if there are abnormal conditions but no OWNCODE addresses are given, the contents of FET word 1 are saved for subsequent use as an exit parameter before processing goes to step 3. The error OWNCODE routine is entered if bits 9-13 have a value of 4 or higher (end-of-information or end-of-volume may also be present); the EOI OWNCODE is entered if the value is less than four. An OWNCODE routine is entered as though a return jump instruction was issued. Execution begins at the start address plus 1. An exit from the routine will, however, return control to the main program, not to CPC. The request which triggered this activity may or may not have been issued; and the program must decide whether to re-issue it. An OWNCODE routine is entered with X1 containing word 1 of the FET complete with bits indicating abnormal conditions; FET word 1 itself has been cleared of the abnormal bits.

3. If the new request is for READ, WRITE or REWRITE, and the FET is already active with the same request, CPC exits, it would be pointless to stop the I/O merely to reactivate it. If, however, the FET is inactive or active with a different request, steps 1 and 2 above are executed as a subroutine. If the new request is a READ, an additional check is made for end-of-logical record or end-of-partition status on the previous request; the new READ is ignored and an exit taken from CPC if either status is present. If a program is reading without recall, the user is forced to clear the logical record bit at the end of each record to ensure that he is aware of the end-of-logical record.

CPC now makes preparations to communicate the new request to the system. The new request code from word 2 of the calling sequence is inserted into bits 0-17 of FET word 1; the old mode bit (bit 1) is not disturbed. The RA+1 request is formatted from the following items:

PP program name obtained from the CPC calling sequence.

Setting of the auto-recall bit in the calling sequence.

First word address of the FET.

RA+1 is set and CPC waits for a zero quantity to re-appear. If the auto-recall bit was set, CPC executes step 2 above as a subroutine. CPC then exits with X1 containing zero if no abnormal conditions were encountered; otherwise X1 contains the value from FET word 1.

CPC saves and restores all registers except A1, A6, X1 and X6.

LOCATIONS RA THROUGH RA+100

The first 100 octal locations within a user field length are used for communication between the operating system and a user job. An additional word, RA+100, is reserved for loading purposes. Many of the words are applicable only to internal operating system routines, and can be ignored by the programmer. Several of the fields in this area are useful in COMPASS programming when macros are called.

Figure 7-1 shows the communication area. Fields within it are:

R Dependent job string recheck bit

A Job swapout to operator action queue (1 = job will be placed under operation queue upon swapout regardless of job origin)

O CFO flag (1 = accept comment from operator)

T Storage move flag (1 = move being attempted)

P Pause flag; when set, program will halt until the operator takes action and clears the flag with GO command; if MESSAGE is called when P is set, the message will flash on the B display

SS Sense switches 1-6 set by SWITCH cards or by operator command ONSWn

M If set, system has CMU hardware available for use

L Library/file flag (1 = name is library name)

X If set, system has the XJ instruction available for use

JO JOB ORIGIN (0=system, 1=central site batch, 2=remote batch, and 3=terminal)

D RSS flag for DIS (see Operator's Guide)

C LOAD complete flag set when load requested by LOADREQ is finished

Locations RA+70 through RA+77 contain the control statement currently being processed. When a job step begins, the control statement verb is placed in bits 18-59 of RA+64, left-justified and binary-zero filled. The parameters are placed in bits 18-59 of RA+2 through RA+52, one parameter per word, left-justified and binary-zero filled. A parameter longer than seven characters is continued in the next word. A zero word marks the end of the parameter list. Bits 0-3 of each parameter word contain one of the following codes which indicates the separator or terminator that followed the parameter.

00	Continuation	04	(	10	;
01	,	05	+	16	other
02	=	06	-	17	. or)
03	/				

The number of words containing parameters (0-51) is placed in bits 0-17 of RA+64.

Example:

This example shows a user field length containing a fictitious control statement verb and parameters created to illustrate all possible separator and terminator codes. The statement ABC(P1=LGO/FILE34*B,P3 +09.2$-$;2(,P5%LAST) appears in RA+2 through RA+77 as follows.

Location	Contents (Octal)	Display Code Equivalent	Control Character
RA+ 2	2034 0000 0000 0000 0002	P1:::::::B	=
RA+ 3	1407 1700 0000 0000 0003	LGO::::::C	/
RA+ 4	0611 1405 3637 4700 0000	FILE34*:::	continued
RA+ 5	0200 0000 0000 0000 0001	B::::::::A	,
RA+ 6	2036 0000 0000 0000 0005	P3:::::::E	+
RA+ 7	3344 5735 5300 5500 0006	09.2$: ::F	-
RA+10	5300 0000 0000 0000 0010	$::::::::H	;
RA+11	3500 0000 0000 0000 0004	2::::::::D	(
RA+12	0000 0000 0000 0000 0001	:::::::::A	,
RA+13	2040 0000 0000 0000 0016	P5:::::::N	other
RA+14	1401 2324 0000 0000 0017	LAST:::::O	terminator
RA+15	0000 0000 0000 0000 0000	::::::::::	
RA+64	0102 0300 0000 0000 0013	ABC::::::K	
RA+70	5555 5501 0203 5120 5534	ABC(P 1	
RA+71	5554 1407 1755 5006 1114	=LGO /FIL	
RA+72	0536 3747 0256 2036 4533	E34*B,P3+0	
RA+73	4453 5735 5353 0055 5346	9$.2$$: $-	
RA+74	5353 5353 7735 5155 5620	$$$$;2(,P	
RA+75	4055 6355 1455 0155 2355	5 % L A S	
RA+76	2455 5255 0000 0000 0000	T) ::::::	
RA+77	0000 0000 0000 0000 0000	::::::::::	

When a control statement is read in response to a CONTRLC macro with the crack parameter, the same interpretation takes place except the verb is taken as the first parameter and placed in RA+2. Bits 18-59 of RA+64 are not altered but bits 0-17 show the parameter word count of the new statement.

Location RA+1 is set by the user, or macros called by the user, when a function is requested from the operating system.

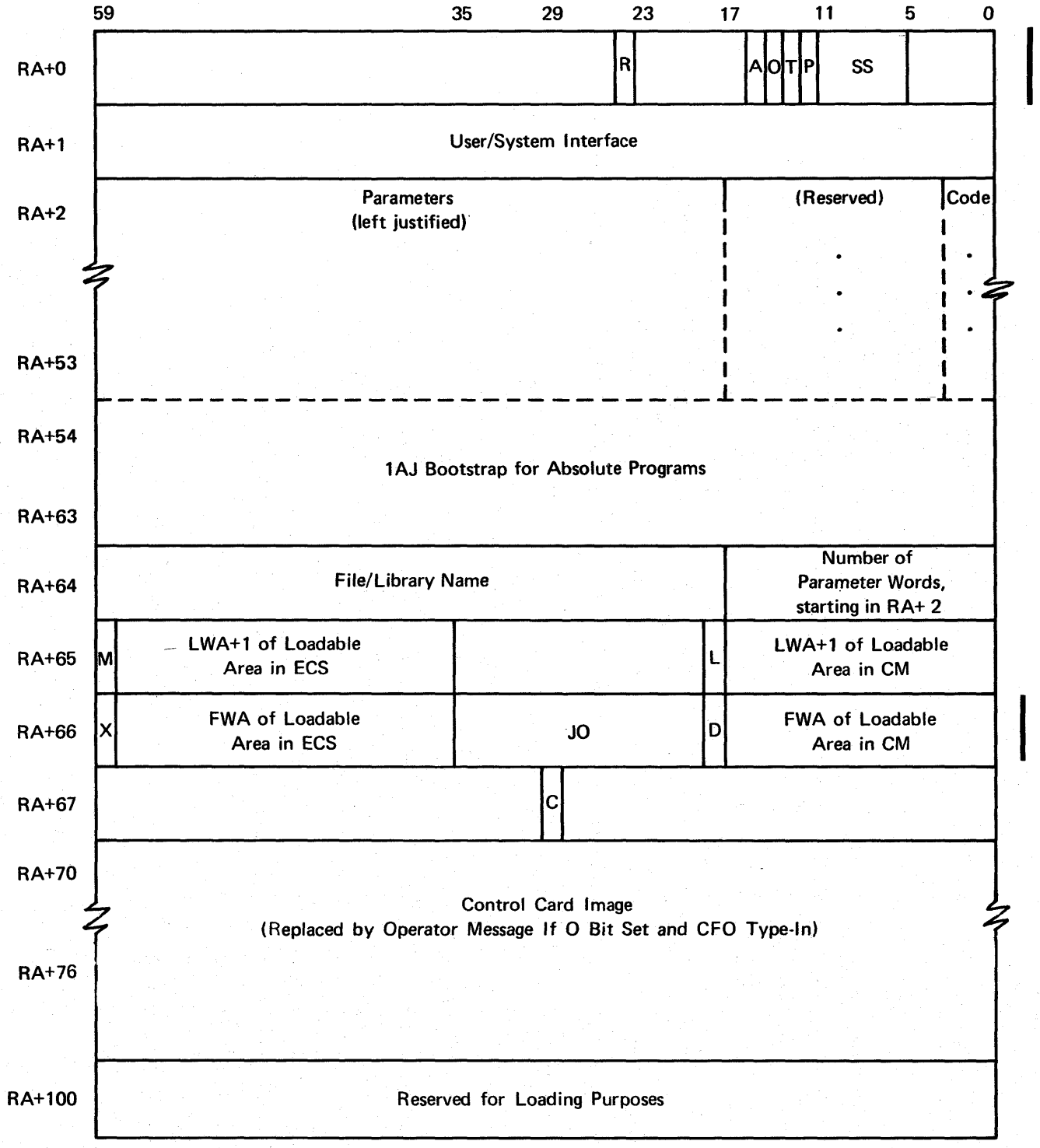


Figure 7-1. Communication Area RA through RA+100

CYBER RECORD MANAGER MACROS

CYBER Record Manager consists of a group of routines providing input/output facilities common to several products. User programs written in COBOL or FORTRAN can communicate with the Record Manager through compiler language calls; COMPASS programmers communicate through the macros listed below.

CYBER Record Manager supports the following file organizations:

Sequential files in physical order

Word Addressable files on mass storage with continuous non-blocked data

Indexed sequential files in which records are physically and logically in order by symbolic keys.

Direct Access files containing records in fixed length blocks; record location is determined by hashing a key to identify a block

Actual Key files in which each record is stored in a location specified by the key associated with that record

The operating system considers all the above types of organization as sequential files. None have name/number indexes similar to those discussed elsewhere in this manual.

The record and block formats supported by CYBER Record Manager are listed below.

Record Type	Description
F	Fixed length records
D	Record length is given as a character count, in decimal, by a length field contained within the record
R	Record terminated by a record mark character specified by the user
T	Record consists of a fixed length header followed by a variable number of fixed length trailers, header contains a trailer count field in decimal
U	Record length is defined by the user for each read or write
W	Record length is contained in a control word prefixed to the record by CYBER Record Manager
Z	Record is terminated by a 12-bit zero byte in the low order byte position of a 60-bit word. Binary zero fill can precede the record terminator; thus, the record an end in 12 to 66 bits of zero.
S	Record consists of zero or more blocks of a fixed size followed by a terminating block of less than the fixed size. These S records are equivalent to the system-logical-records discussed elsewhere in this manual.

Block Type	Description
K	All blocks contain a fixed number of records; the last block can be shorter
C	All blocks contain a fixed number of characters; last block can be shorter
E	All blocks contain an integral number of records; block sizes may vary up to a fixed maximum number of characters
I	A control word is prefixed to each block

COMPASS macros used by CYBER Record Manager reside in the system text overlay IOTEXT; if system defaults are installed, macros also reside in overlay SYSTEXT. General macro names and functions are given below; specific variants of these macros are detailed in the Record Manager reference manual along with other product capabilities.

Macro	Function
File Creation and Maintenance Macros	
FILE	Creates file information table (FIT)
FETCH	Retrieves value of any field in FIT
STORE	Sets value in field of FIT
File Initialization and Termination Macros	
OPENM	Prepares a file for processing; initiates label processing
CLOSEM	Terminates file processing; initiates label processing
Data Transfer Macros	
GET	Transfers data from file to working storage area
GETP	Retrieves a portion of a record from a file
PUT	Transfers data from working storage area to a file
PUTP	Transfers a portion of a record to a file
CHECK	Determines completion status of I/O operations
File Positioning Macros	
SKIP	Repositions file backward or forward
REWINDM	Rewinds volume to beginning-of-information
SEEK	Provides overlap between I/O and processing by positioning while processing

File Updating Macros

DELETE Deletes record from file

REPLACE Replaces record in file

Boundary Condition Macros

WTMK Records a tape mark on a tape file

WEOR Records end of a section

ENDFILE Records end of a partition

A FILE control statement equivalent to the FILE macro also is available.

Files created by CPC can be read or written by CYBER Record Manager once they are properly described to Record Manager. Similarly, a file created by Record Manager can be read by CPC if the file structure conforms to that required by READ and WRITE macros. A file should not be manipulated by both Record Manager and CPC within a given run.

The reference manual for Record Manager contains details of its use. CYBER Record Manager macros are not further discussed in this manual.

SYSTEM COMMUNICATION MACROS

Communication between the operating system and a program written in COMPASS is provided by the following macros. These macros exist within all of the COMPASS system text overlays CPCTEXT, IOTEXT, SYSTEXT, SCPTEXT, and TXT6RM.

SYSCOM MACRO

This macro defines standard symbols and macros.

SYSCOM B1

If B1 is present, the COMPASS pseudo instruction B1=1 is generated. This informs COMPASS that register B1 contains 1 throughout the program, and can affect the code produced by the R= pseudo instruction. The micro MODEL is defined as the two characters 74. The symbols listed below are made available for use by the user program.

RA.SSW	=	0	Sense switches in bits 11-6.
RA.MTR	=	1	System monitor request register.
RA.ARG	=	2	Start of control statement argument list.
RA.PGN	=	64B	Bits 59-18 = program name.
RA.ACT	=	64B	Bits 17-00 = argument count.
RA.LWP	=	65B	Last word pointers for overlay load.
RA.CMU	=	65B	Compare move unit flag (bit 59)

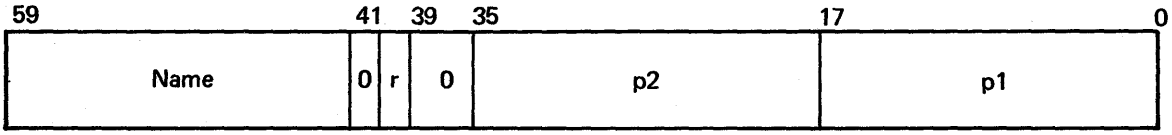
RA.FWP	= 66B	First word pointers for overlay load.
RA.CEJ	= 66B	Bit 59 = central exchange jump enable flag.
RA.LDR	= 67B	Loader communication word.
RA.CCD	= 70B	First word of control card image.
RA.ORG	= 100B	Origin of overlay header word for absolute programs.

SYSTEM MACRO

This macro is used for issuing system requests for which no specific system macro is provided. It is also used by many of the system action macros. Registers X1, X6, A1, and A6 are destroyed during macro execution.

SYSTEM name,recall,p1,p2

The SYSTEM macro generates the following in X6 and issues a return jump to SYS=.



name Display-coded name of PP program

r Optional recall parameter

p1 First parameter to PP program

p2 Second parameter to PP program

The value of p1 or p2 cannot exceed 377777 (octal).

COMMON USES OF SYSTEM MACRO

ABS is a system program used by a central processor program to dump absolute core. This request is done by issuing a call to PP routine ABS. The call to ABS can be issued with or without auto-recall either by using the SYSTEM macro or by placing the call in RA+1 directly. If auto-recall is not used, the program uses:

SYSTEM ABS,,from,thru.

If auto-recall is used, the programmer establishes a parameter word that contains the thru and from values. The format of the parameter word is:

Bit 11-0 Zero-filled, bit zero used as complete bit

Bit 29-12 Thru value

Bit 47-30 From value

The central processor program then uses:

SYSTEM ABS,R,pointer to parameter word.

DMP is a system program used by a central processor program to dump specified portions of field length. This request is done by issuing a call to PP routine DMP. The call to DMP can be issued with or without auto-recall either by using the SYSTEM macro or by placing the call in RA+1 directly. If auto-recall is not used, the program uses:

SYSTEM DMP,,thru,from.

If auto-recall is used, the programmer establishes a parameter word that contains the thru and from values. The format of the parameter word is:

Bit 11-0	Zero filled, bit zero used as complete bit
Bit 29-12	Thru value
Bit 47-30	From value

The central processor program then uses:

SYSTEM DMP,R,pointer to the parameter word

Only the first 131K words of memory can be dumped with DMP. See the ABS control statement if 198K or 262K memory is to be dumped.

REGISTER SAVE/RESTORE FUNCTION

To save or restore registers, a program can issue a call for an XJR function through RA+1. This special call is processed entirely by central monitor (CPMTR).

To issue the XJR call the program can use the SYSTEM macro as follows:

SYSTEM XJR,R,addr,1

XJR	Name of system process.
R	Recall parameter. This call must be made with recall.
addr	Address of a 16-word parameter area to contain the exchange package. The format of this area is described with the DMP control statement.
1	Save function requested; if omitted restore requested.

For the 1 save function, CPMTR saves the job's current exchange package in the parameter area. Registers X1, X2, X6, A1 and A6 are destroyed by the SYSTEM macro and by the subroutine SYS=.

For the restore function, CPMTR sets up an exchange package containing X0-X7, B1-B7, A0-A7 and P from the parameter area. RA, FL, EM, RE, FE and MA registers come from the job's current exchange package. The result, then, replaces the job's current exchange package. Execution resumes at the address pointed to by P in the parameter area. This is the only safe way to set registers A1 through A7 to values outside the current field length.

INTEGER DIVIDE Opdefs

These opdefs provide for division of 48-bit integers.

```
IXi     Xj/Xk
IXi     Xj/Xk,Bn
```

The integer quotient (fraction truncated) result in register Xi has sign extension in bits 59-48. The first form destroys register B7, and the second form destroys register Bn. The contents of Xj and Xk are the floating point normalized operands.

SYSTEM ACTION MACROS

The macros described in this section allow the user to receive status information from the operating system and to change some job parameters. Calling these macros from a COMPASS central processor program results in RA+1 requests for Monitor functions or PP programs.

The macros reside in the following COMPASS system text overlays: CPCTEXT, IOTEXT, SYSTEXT, and SCPTEXT. All of the system action request macros call the system communication subroutine SYS=, except as noted in the individual macro descriptions; these macros do not call CPC. The subroutine SYS= resides in the library NUCLEUS.

Generally, the system action macros use registers alike; only registers X1, X6, A1, and A6 are destroyed. All registers except X1 and X6 can be used as parameters. Register X1 can be used as the first, but not as the second, parameter. Register X6 cannot be used as a parameter. Exceptions are noted in the macro descriptions.

ENDING PROGRAMS

Programs can be ended by one of two macros:

ABORT	Abnormal termination
ENDRUN	Normal termination

These functions result in a Monitor request for ABT and END, respectively; they are executed immediately by Monitor.

ABORT MACRO

The ABORT function causes abrupt termination of the present program, and, if an EXIT, EXIT(U) or EXIT(S) does not appear among the remaining control statements, causes job termination.

ABORT lfn,p2,p3

lfn — Position allows for SCOPE 2 compatibility. Any non-blank value in this field causes an assembly error under NOS/BE.

p2 — Optional parameter. Characters ND in this field suppress the DMPX user dump. Characters NODUMP suppress the DMPX user dump and cause control statement processing to be resumed only after an EXIT(S) control statement has been encountered. EXIT, EXIT(C), and EXIT(U) control statements are skipped. Any other non-blank value in this field is ignored.

p3 — Optional parameter. Character S in this field causes control statement processing to be resumed only after an EXIT(S) control statement is encountered. EXIT, EXIT(C), and EXIT(U) control statements are skipped. Any other non-blank value in this field is ignored.

The DMPX user dump produced when p2 is blank (or any non-blank value except ND or NODUMP) shows the contents of the exchange package, contents of the operating registers, and memory locations near the location of the ABORT call.

The effect of the various EXIT control statements on job processing and DMPX production after an ABORT call is shown in the following chart. Resume indicates that the statements following EXIT are executed; skip indicates that the following statements are not executed.

ABORT	Call	DMPX	EXIT.	EXIT(C)	EXIT(S)	EXIT(U)
ABORT		Yes	Resume	End job	Resume	Resume
ABORT	,ND	No	Resume	End job	Resume	Resume
ABORT	,ND,S	No	Skip	Skip	Resume	Skip
ABORT	,,S	Yes	Skip	Skip	Resume	Skip
ABORT	,NODUMP	No	Skip	Skip	Resume	Skip
ABORT	,NODUMP,S	No	Skip	Skip	Resume	Skip

ENDRUN MACRO

The ENDRUN function is usually the last instruction to be executed in a user program. No parameters can be used with this request.

ENDRUN

Monitor causes the operating system to examine the control statements and begin processing of the next control statement. If the next control statement contains a 7/8/9 multiple punch or is EXIT. or EXIT(S), the job is terminated.

GETMC MACRO

The GETMC macro obtains the characteristics of the mainframe on which the user's routine is executing.

The format of the macro is:

GETMC addr

addr — Address of a word where the following information is returned.

Bits 59-49	Reserved for software characteristics
Bit 48	System assembled for 63-character set
Bits 47-36	ECS size/1000B
Bits 35-24	Number of PPUs
Bits 23-20	Reserved for hardware characteristics
Bits 19-18	CYBER 176 mainframe flag 0 Not a 176, 1 Type A 2 Type B 3 Type C
Bit 17	PPUs running at 2x speed (CYBER 170 series only)
Bit 16	CYBER 17x mainframe
Bit 15	CMU is present
Bit 14	CEJ/MEJ option is present
Bit 13	CPU 0 has instruction stack
Bit 12	CPU 1 is present
Bits 11-1	Memory size/200B
Bit 0	Completion bit

FIELD LENGTH REQUEST

The amount of extended core storage or central memory assigned to a job can be changed by the MEMORY macro. The MEMORY macro can also be used to obtain the current ECS or central memory field length assigned to the job, obtain the maximum ECS or central memory field length available to the job, or release all ECS assigned to the job.

Format of the MEMORY macro call:

MEMORY type,address,recall,length,nabort

type	CM, SCM† or blank, central memory request; ECS or LCM,† extended core storage memory request.
address	Address of request/reply word; if omitted an assembly error results.
recall	Optional recall parameter. If recall is specified, control is not returned to the user's program until the request is honored. Any non-blank parameter is acceptable. Recall is required on all requests for memory increases.
length	Optional parameter giving number of words of field length requested.
nabort	Optional parameter which averts a job abort if non-blank prevents job termination when requested field length exceeds field length defined on the job statement, or when other problems involving field length discrepancies occur in loading the user's job. If a non-blank nabort parameter is used, and an ABORT cannot be prevented, the current field length is returned in bits 30 through 59 of the status word. (Memory is allocated in portion of 100 (octal) for central memory and 1000 (octal) for ECS.)

Format of MEMORY macro request/reply word is two 30-bit fields:

Bits 0-29 should always contain zero when the request is issued. Bit 0 is set to 1 upon completion of the request.

If the length parameter in the MEMORY macro call is blank, the upper 30 bits of the request/reply word determine the action taken.

If bits 30-59 contain zero, the current field length of the type specified in the macro call is returned right justified in bits 30-59.

If bits 30-59 contain negative zero (7777777777B) and the type parameter in the MEMORY macro call is ECS or LCM, all extended core storage assigned to the job is released. If a negative zero is given when the type parameter is not ECS or LCM, an error condition results. Also, the message MEM ARG ERROR is issued to the dayfile and the job is aborted.

If bits 30-59 contain negative one (7777777776B) the maximum type field length available to the job is returned right justified in bits 30-59 of the request/reply word.

---

†SCM and LCM are allowed for compatibility with SCOPE 2.

Any value, other than those described above, in bits 30-59 of the request/reply word is assumed to be the field length desired; and this value is requested. If the request is satisfied, the field length is returned right justified in bits 30-59 of the request/reply word; and bit 0 of the request/reply word is set to 1. The system rounds the user's field length to the nearest 100 (octal) central memory words or 1000 (octal) ECS words above the requested length.

If the request cannot be satisfied and the nabort parameter in the MEMORY macro call was not blank, the current field length is returned in bits 30-59 of the request/reply word, and the job continues at that field length. If the nabort parameter was blank, the job is aborted.

Because system routines may read ahead, field length should not be reduced to within four words of last used location.

DAYFILE MESSAGES

A message is always placed in the control point message area and optionally entered into the job or system dayfile with the MESSAGE macro. The control point message area is displayed on the operator console B display, and the dayfiles are displayed on the operator console A display.

The message flashes for operator attention if its first character is $ or if the pause bit is set when MESSAGE is called (bit 12 of word RA+0). The MESSAGE macro calls the system communication subroutine MSG=.

MESSAGE addr,display,recall

addr — First word address of the message.

display — Ordinal specifying message disposition. If omitted, default is 0.

0	Enter in system and job dayfiles and control point message area.
1	In control point message area only.
2	Same as option 1 (for compatability with other systems).
3	Enter in job dayfile and control point message area.
4	Enter in CERFILE (system programs only).
5	Dayfile accounting message (system programs only).
6	Same as option 0 but do not send to user's terminal.
7	Same as option 3 but do not send to user's terminal.
8 or more	Same as option 1.
LOCAL	Display on B display and record in job dayfile.
other non-blank	Display on B display but do not record elsewhere.

recall — Optional recall parameter; if non-blank, MSG=constructs a status word.

Within the program the message must be stored in display code and should not contain any characters with display code values greater than 57 since these cannot be displayed on the console. Any display code value greater than 57B or 0 is replaced with a blank (display code value of 55B). Maximum message length of 80 characters is established by the dayfile processing routine: 40 characters appear on each line. Messages exceeding 80 characters are truncated. Those shorter than 80 characters must be terminated by a word with zeros in bits 0-11. The CERFILE option is an exception since the message length is always six CM words. It is assumed to contain binary data so no character checks are made. The data is entered in the CERFILE and nowhere else.

RECALL MACRO

RECALL causes the program to relinquish control of the central processor. The conditions that determine when the job regains control of the processor depends on the form of the macro used.

Periodic recall results from:

```
RECALL
```

Control returns to the user program after a short period of time or when any PP program terminates processing for the job. Once control is regained, the user must determine whether the condition that required recall is still present. This form of the RECALL macro calls the system communication subroutine RCL=.

Automatic recall results from:

```
RECALL  addr
```

addr — Address of a word (usually the first word of a FET) which has bit 0 set to 1 before control returns to the user program.

If CPCTEXT is used when addr is the first word of the FET and error or end-of-partition bits are set in the code and status field of the FET, control returns to a user OWNCODE routine if it exists. Such routines are established by setting the EP or UP bits and specifying OWNCODE addresses. If other texts are used for the assembly, the RECALL macro calls the system communication subroutine WNB= (wait not busy).

Since recall may be entered when an input/output operation is initiated, the RECALL macro is needed only if some useful processing can be done in the time the input/output operation is being completed.

STATUS INFORMATION

TIME AND DATE MACROS

The user can determine the date and time in several formats by accessing clocks kept internally by the system. Each of these functions calls the system communication routine SYS=.

CLOCK — Current system clock in hours, minutes, and seconds

DATE — Current date established at deadstart time when the system was loaded

JDATE — Current date in format yyddd for year and date

RTIME — Real time clock maintained by Monitor, in fractional seconds

TIME — Central processor time allowed and used by job

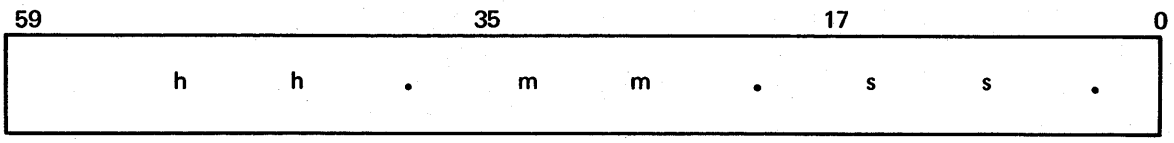
IOTIME — Input/output time allowed and used by job.

Each of these functions requires the user to identify a status word. The system returns the requested information before clearing location RA+1 to mark the function complete.

The macros, and the format of the status returned, are given below.

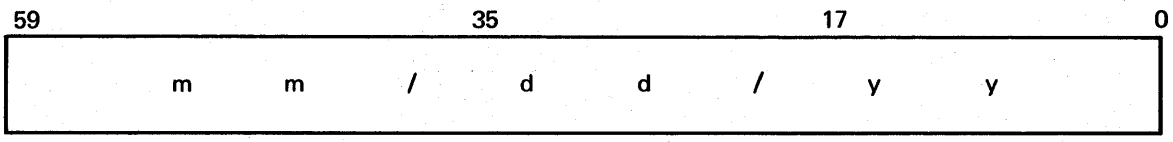
The system clock is that established when the operator loads the system. Display code hours, minutes, and seconds appear with periods and a leading blank as follows:

CLOCK status



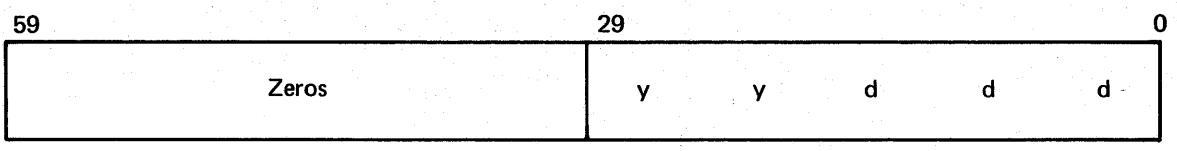
The date returned is that typed by the operator when the system was loaded. Its format is display code, and generally is mm/dd/yy for month, day, and year; this order may be changed at installation option. A leading and trailing blank appear.

DATE status



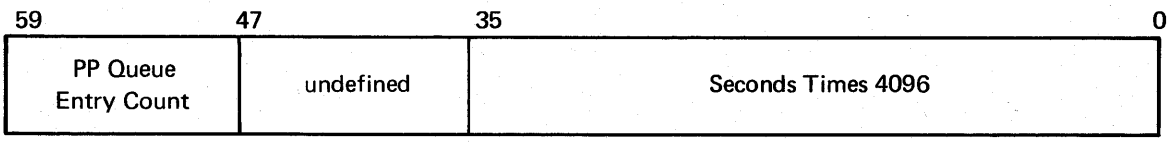
Date in a format suitable for calculating elapsed days is returned with JDATE. Five display code characters appear in the low order position; the first two digits are the year, the last three the number of the day in the year.

JDATE status



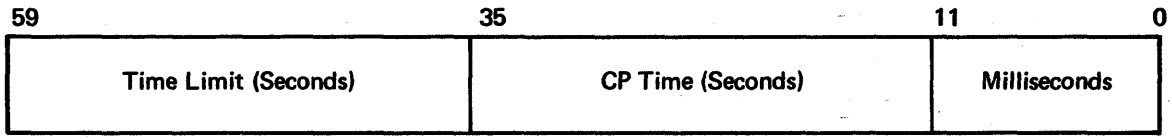
The real time clock is that maintained by Monitor for purposes such as determining peripheral processor time used. The status word will show seconds in bits 12-35 and units of 4096ths of a second (244 9/64 microseconds) in bits 0-11.

RTIME status



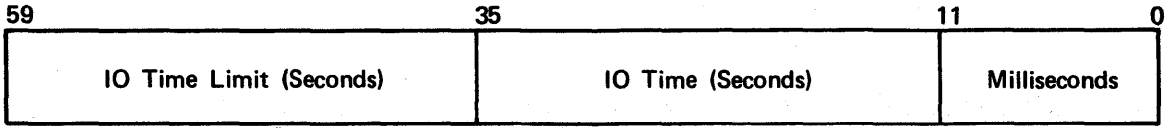
The job time limit is that requested on the job statement or assigned by installation default. Central processor time used is shown in seconds and milliseconds.

TIME status



The IO time limit is requested on the job statement or assigned by installation default. Used IO time is shown in seconds and milliseconds.

IOTIME status



STATUS MACRO

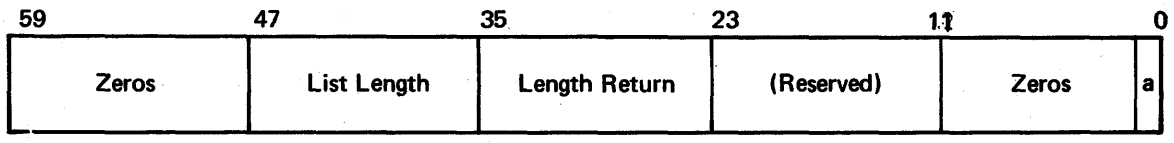
The STATUS function provides a user program with information about system resources. Two types of information are available depending on the value of the x parameter as described below.

The call to this macro is:

STATUS list,x,recall

list	Address of a header word containing the length of the area in which status information is to be returned. The status area begins at list+1.
x	x = 1 maps available space on all public rotating mass storage devices. x = 2 returns system information concerning files assigned to the user program. x = 3 PRU count for a file (or files) x = 4-777 reserved; 1000 to 7777 reserved for installation use.
recall	Optional recall parameter; any non-blank character.

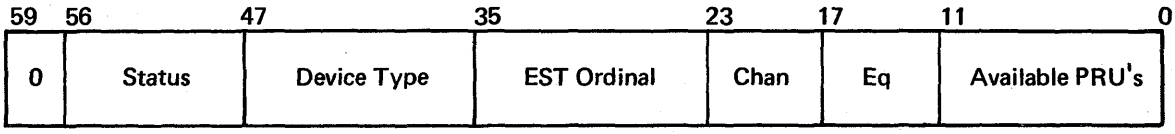
Format of the header word must be:



list length	Number of words, excluding this header word, to be used for return information; must be set by user to other than 0.
length return	Number of status words returned; set by operating system when list is complete.
a	Must be set to 0 before issuing a STATUS call.

The header word is also the auto recall reply word: when bit a becomes 1, the request is complete.

When x=1, the system returns one word of information for each rotating mass storage device available with the default allocation flag set in the RBR. Format is:



status — 9-bit binary field:

000	Unavailable device
020	Mounted device
040	Dismounted device
060	Idled device

device type — Hardware mnemonic in display code:

AH	819 Disk Drive
AM	841 Multiple Disk Drive
AY	7054/844-21 Disk Drive
AX	ECS resident files
AZ	7054/844-41 Disk Drive

EST ordinal — Position of entry for device in equipment status table (12-bit binary field)

chan — Channel number by which device can be accessed.

eq — Equipment (controller) number to which device is connected.

PRUs — Number of PRUs, divided by 100 octal, of space remaining on the device; value of 7777 indicates at least 262,100 PRUs available.

When x=2, the status area contains one three-word entry for each file name, which should appear left-justified, zero-filled in the first word of each entry. If the file exists, the file name is replaced by the first three words of the file name table (FNT). If the file does not exist, the file name is zeroed out. Information in the FNT is used by some compilers.

When x=3, the list length field in the STATUS macro list header word must specify two words for each file requiring size determination. The first word contains the file name, left-justified with zero fill. Upon return, the second word, bits 0 through 23, contains the PRU count for the file.

FILESTAT MACRO

The FILESTAT macro is an alternate for the STATUS macro:

```
FILESTAT list,recall
```

This macro is equivalent to:

```
STATUS list,2,recall
```

FILINFO MACRO

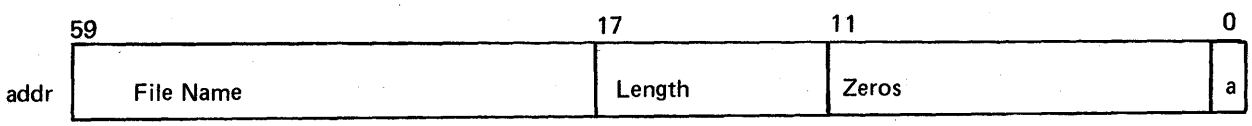
The FILINFO macro provides a user program with information about a file assigned to the user's control point.

The call to this macro is:

FILINFO addr

addr Address of a five-word table to receive file information.

Format of the header word must be:

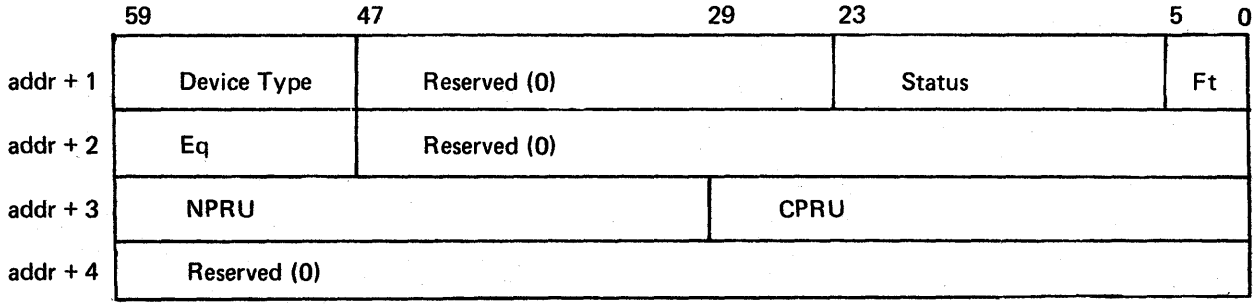


file name A valid display-coded file name. Must be local to the user's job.

length Table length including the first word (should be set to 5). Must be at least 4. If set to 4, four words are returned. If set to 5 or more, five words are returned.

a Must be set to 0 before issuing a FILINFO call (will be set to 1 when the operation is completed).

When the operation is completed, the table will have the following format.



	device type	Hardware mnemonic in display code:

AH	819 Disk Drive
AM	841 Multiple Disk Drive
AX	ECS resident files
AY	7054/844-21 Disk Drive
AZ	7054/844-41 Disk Drive
LM	Link medium file
MT	657 or 667 magnetic tape drive
NT	659, 669, or 679 magnetic tape drive
TR	3691 Paper Tape Reader
TP	3691 Paper Tape Punch
LP	Line printer (any)
LQ	Line printer (512)
LR	Line printer (580-12)
LS	Line printer (580-16)
LT	Line printer (580-20)
CR	Card reader (405)
KB	Remote terminal keyboard
CP	Card punch (415)
DS	Console display
GC	252-2 Graphics Console
HC	253-2 Hardcopy Recorder
FM	254-2 Microfilm Recorder
PL	Plotter

status	Bits 23-21	Sequential file position:

23	End-of-information
22	End-of-file
21	Beginning-of-information

Bits 20-18	Magnetic tape characteristics:

20	Labeled tape
19	9-track tape
18	7-track tape

Bit 17	File is open
Bit 16	File is connected to terminal
Bit 15	File is on mass storage
Bits 14-10	Reserved (0)

Bits 9-6 Permissions:†

9 Modify
8 Extend
7 Write
6 Read

ft File type (6-bit binary field):

00 Local scratch
01 Input (file name is INPUT)
02 Output (print disposition)
03 Punch (punch disposition)
04 Permanent file
77B Other disposition

eq Equipment number, the EST ordinal of the device (12-bit binary number)

NPRU File size in PRUs (if RMS file):

Bits 59-36 PRU count
Bits 35-30 0

CPRU Current file position (if RMS file) given as the number of PRUs from beginning-of-information (beginning-of-information is indicated by PRU count=1):

Bits 29-6 PRU count
Bits 5-0 0

GETJCI MACRO

The GETJCI macro allows a user program to transfer the job control information used by the CCL to a specified location in the job's central memory field length. Job control information fields can be changed by executing the GETJCI macro to obtain the current fields, then modifying the appropriate fields, and executing the SETJCI macro to save the new fields in the system area.

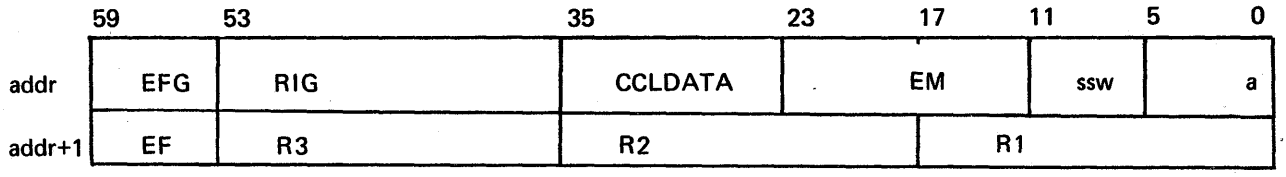
The call to this macro is:

```
GETJCI  addr
```

addr Address of a two-word table.

† For mass storage files, read, extend, and modify reflect permanent file permissions. Write permission is set if either modify or extend permission is set. For magnetic tape files, modify, extend, and write permissions are set if the write-enable ring is present; cleared if the ring is absent. Read permission is set unless the file is a multifile tape.

Format of this header word must be:



EFG	Contents of global error register †
RIG	Contents of global register †
CCLDATA	Contents of CCL register, for CCL use only (read by GETJCI)
EM	Value of error mode, set only by mode statement (read by GETJCI)†
ssw	Value of sense switches, set by SWITCH statement or by by SETJCI macro: Bit 6 Switch 1 Bit 11 Switch 6
a	Complete flag, must be reset to 0 before execution and set to 1 when function is complete
EF	Value of error flag. (If not set by the user, the system sets EF when the job aborts. If set to a non-zero number by the user, EF is saved by the system but does not cause job abort.)
R3-R1	Contents of local registers†

SETJCI MACRO

The SETJCI macro allows a user program to transfer the job control information used by CCL from a specified location in the job's central memory field length. Job control information fields can be changed by executing the GETJCI macro to obtain the current fields, then modifying the appropriate fields, and executing the SETJCI macro to save the new fields in the system area.

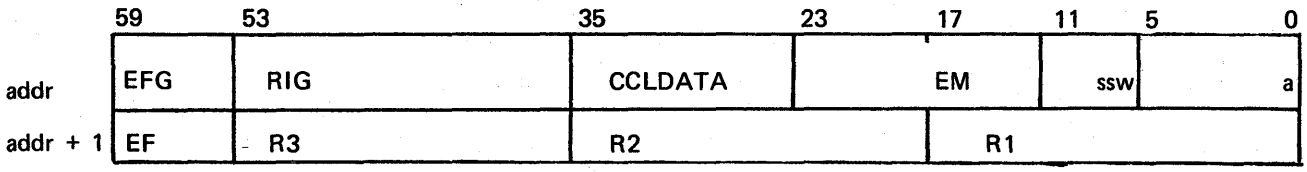
The call to this macro is:

SETJCI addr

addr Address of a two-word table.

† These registers are CCL symbol names.

Format of this header word must be:



EFG	Contents of global error register†
RIG	Contents of global register†
CCLDATA	Contents of CCL register, for CCL use only (ignored by SETJCI)
EM	Value of error mode, set only by MODE statement (ignored by SETJCI)†
ssw	Value of sense switches, set by SWITCH statement or by SETJCI: Bit 6 Switch 1 Bit 11 Switch 6
a	Complete flag, must be reset to 0 before execution and set to 1 when function is complete
EF	Value of error flag. (If not set by the user, the system sets EF when the job aborts. If set to a nonzero value by the user, EF is saved by the system but does not cause job abort.)
R3-R1	Contents of local registers†

† These registers are CCL symbol names.

DEPENDENT JOB COUNT

The dependency count of a job within a dependent string can be decremented from within a user program. This count also can be decremented by a control statement. Dependent jobs are explained in section 4 with the TRANSF description. Jobs in a dependent string do not execute until their dependency count is zero.

The TRANSF macro is used to decrement the count of a job dependent on the currently executing job.

TRANSF list

list — Beginning address of a list naming the jobs for which the dependency count is to be reduced.

Names in the list should be left-justified with zero fill; the last word must be all zeros.

READING CONTROL CARDS

With the CONTRLC function a central processor program can read or backspace within the control statements for the job. When the function is executed, the pointer to the next control statement is moved. The user is responsible for the resulting position of the control pointer.

CONTRLC status,function,dfile,crack

status — Address of a reply word.

function — Control statement pointer repositioning:

- READ — Move the statement image to RA+70 (octal) through RA+77 (octal) and change the pointer to point at the start of the succeeding control statement. The optional actions, described later, are done on the statement image in RA+70.
- BKSP — Change the pointer to point at the start of the control statement preceding the current statement.

dfile — Optional dayfile indicator. If non-blank the statement image is to be sent to the dayfile when the function is READ.

crack — Optional parameter; any non-blank character. When the function is READ, non-blank parameters from the statement are to be placed in locations RA+2 through RA+53, aligned as shown below.

The reply word also is used to pass the function code in bits 0-17. If the function type is specified as above in the macro call, the macro puts the code into the word. If the function field is blank, the user must put the proper value into the word. The following codes are used:

000010 (octal) READ 000040 (octal) BKSP

Bit 0 of the reply word is set to 1 when the function is complete. Bit 4 of the reply word is set to 1 if READ attempts to go past the last statement in the control statement record or in a CCL procedure. Bit 4 is also set to 1 if BKSP attempts to backspace past the job statement in the control statement record or past the procedure header statement in a CCL procedure.

If parameter cracking is requested, the parameters are stored left-justified with zero fill in bits 18-59; and a code indicating how the parameter ended is stored in bits 0-3. If the parameter is longer than seven characters, the first seven characters are stored with the 00 code and the parameter is continued in the next word. The word count is stored in bits 0-17 of RA+64 (octal).

Processing stops when a terminator is found. The parameter ending codes are as follows:

00	Continuation	05	Plus
01	Comma	06	Minus
02	Equals	10	Semicolon
03	Slash	16	Other
04	Left parentheses	17	Terminator

In the cracking process, a statement is always considered to be a continuation statement. Blanks are always squeezed out and cannot be used to delimit the first parameter (keyword). Also the first parameter is always put in RA+2, the second in RA+3 (assuming the first parameter is less than eight characters), and so on.

PROGRAM RECOVERY

Two means are available to recover the results of a program that aborts during execution:

The RECOVR macro can establish conditions under which control returns to the program after an error so that outstanding results can be saved or diagnostic information produced. The same results can be achieved by a direct RA+1 call to RPV.

The CHECKPT macro can call for a checkpoint during execution, such that the program can be restarted from the last checkpoint in the event of a subsequent abort.

RECOVR MACRO

With the RECOVR macro, a user program can gain control at the time when normal or abnormal job termination procedures would otherwise occur. Initialization of RECOVR at the beginning of a program establishes the conditions under which control is to be regained and specifies the address of user recovery code. If the stated condition occurs during program execution, control returns to the user code.

RECOVR macro expansion calls the SETUP. subroutine. If necessary, the system increases the CP time limit, IO time limit, or mass storage limit to provide an installation defined minimum of time and mass storage for RECOVR processing. No limit is increased more than once in a job.

RECOVR is concerned with conditions that affect job execution. The conditions under which control returns to the user, and the octal values that select them in the call to RECOVR, are:

Arithmetic mode error	001	System abort	020
PP call or auto-recall error	002	CP abort	040
Time or storage limit exceeded	004	Normal termination	100
Operator drop, kill, or rerun	010		

Conditions can be combined as desired, with octal values up to 177 allowed in the flag field of the call to RECOVR.

At least five seconds of central processor time always are available for user code execution. RECOVR makes the exchange jump package and RA+1 contents available to the program if user recovery code is executed, and gives the user the option of having normal or abnormal job termination output.

Initialization of RECOVR within code at the beginning of a program results in an entry in a stack of requests for PP program RPV. Although RPV can be called directly by a Monitor request in RA+1, use of the RECOVR utility is preferable for all except stand alone system utilities because operating system routines themselves use this capability. Only one set of recovery conditions can exist within RPV, but RECOVR allows up to five user and system set of flags and code for each program. The last RECOVR initialization will receive control first.

The second specification of a subroutine overrides its previous parameters. This override can be used to remove a subroutine from the RECOVR list by passing a mask of zero.

A checksum of the user recovery code can be requested during initialization. If flagged conditions subsequently occur, RECOVR again checksums the code before returning control to it. This gives some assurance of user code integrity before it is executed.

RECOVR is initialized from a COMPASS program with:

RECOVR name,flags,checksum

name	Address of code to be executed if flagged conditions occur; a return jump is made to this location
flags	Octal value for conditions under which recovery code is to be executed, as outlined above; default is 77
checksum	Last word address of recovery code to be checksummed; 0 if no checksum

If one of the flagged conditions occurs, three arguments are passed to the reprieve-time subroutine. A1 contains the address of the argument list; X1 contains the address of the first argument. The arguments are the following:

1. A 17-word (decimal) array showing the program situation when RPV was called. The first 16 words are an image of the exchange package. The seventeenth word is the contents of RA+1.

2. A flag that determines the type of program termination. If the user sets the flag non-zero, ENDRUN termination occurs upon completion of the last post-processing subroutine. If the flag remains zero, the original error code and the exchange package are restored and the job continues as if RECOVR had not been called. Altering the exchange package passed as argument 1 prevents the correct completion of the restore, but does not impair system operation.

3. An array, starting at RA+1, that allows a FORTRAN Extended subroutine to access all of the user's field length.

The subroutines called by RECOVR should return; if they do not, additional subroutine calls, if any, and the register/error flag restore is not performed.

If a program calling RECOVR contains overlays, both the call to RECOVR and the user recovery code should be a part of the level 0,0 code.

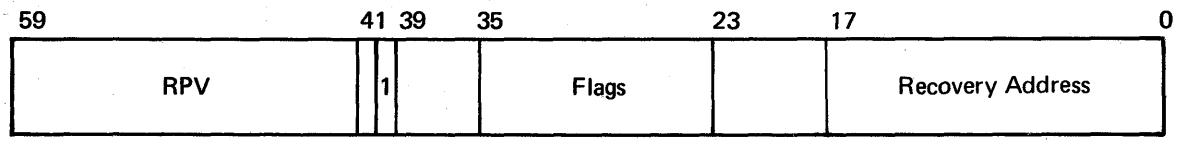
The exchange jump package returned by RECOVR is in the format shown with the DMP control statement, with the system error code that caused recovery code execution in bits 0-17 of the first word. If the P register shows zero in the package because a mode error occurred, bits 31-47 of RA+0 contains the P register value. System error codes that may be returned are:

Condition	Error Code (octal)	RECOVR Mask (octal)
Normal termination	0	100
Requested time limit exceeded	1	004
Arithmetic mode error	2	001
PP program requested abort	3	020
CP program requested abort	4	040
PP program cannot be called from RA+1	5	002
Operator DROP	6	010
Operator KILL	7	010
Operator initiated job rerun	10	010
CP abort (ABT + bit 36)	11	040
ECS parity error	12	020
Required auto-recall status missing	15	002
Job hung in auto-recall	16	002
Requested mass storage limit exceeded	17	004
xxx not in program library	20	002
IO time limit exceeded	21	004

The FORTRAN Extended language contains RECOVR subroutines as detailed in that reference manual.

CALLING RPV DIRECTLY

The PP program RPV can be called by setting RA+1 as follows:



The code at the recovery address should allow for a 21 octal word array to be returned. Control is returned to word 22.

An optional checksum of the recovery area can be requested in the user call. If the word at the recovery address contains all zeros, no checksum is taken. If the upper 30 bits contain the last word address of the recovery area, a checksum of the code address+21 through last word address is made and stored at the recovery address+1.

CHECKPT MACRO

A checkpoint of the program and files in use is obtained with the CHECKPT macro. The RESTART control statement is used to restart a job on the basis of information obtained from the checkpoint dump. See the CKP control statement for information about the checkpoint dump tape and other general information.

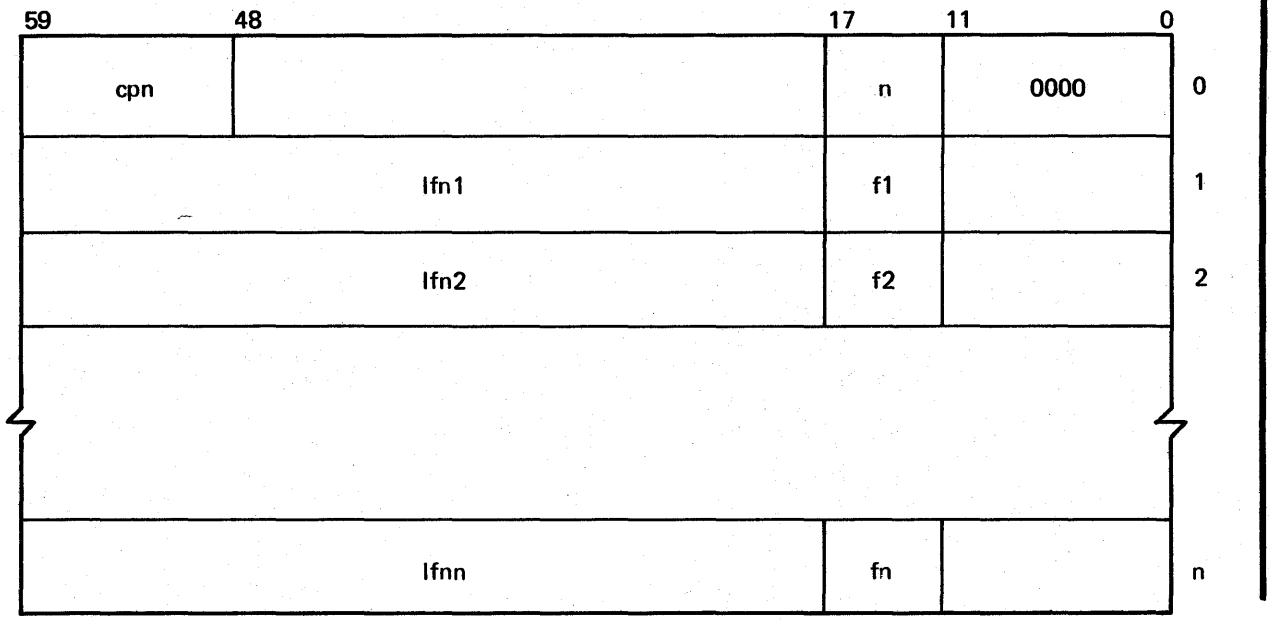
An executing program would request checkpoint at various logical points, such as end-of-partition, x logical records processed, x seconds of elapsed time, etc. Checkpoint requests may be issued more than once. The request takes the following form:

CHECKPT param,sp

sp — Mass storage files to be processed.

- 0 — All files
- Non-zero — Certain standard files plus files in a parameter list. Assumed 0 if sp is not given.

param — Address of a parameter list formatted as follows:



cpn — Contains the checkpoint number unconditionally returned by CHECKPT. A zero value indicates no checkpoint was taken.

n — Defines number of lfn entries in following list, to a maximum of 42 (decimal).

lfn — Name of user mass storage files to be processed; left-justified display code.

f Octal number indicating specific manner in which lfn is to be processed.

0 Mass storage file is copied from beginning-of-information to its position at checkpoint time, and only that portion is available at restart. The file is positioned at the latter point.

1 Mass storage file is copied from its position at checkpoint time to end-of-information, and only that portion is available at restart. The file is positioned at the former point.

2 Mass storage file is copied from beginning-of-information to end-of-information; the entire file is available at restart time. The file is positioned at the point at which the checkpoint was taken.

3 The last operation on the file determines how the mass storage file is copied.

When the manner of copying a mass storage file is to be determined from the last operation on the file, checkpoint derives f values from the last code status as follows:

f = 0 if code/status ends in 4, 5, 6, or 7 or if code/status ends in 0, 1, 2, or 3 and end-of-information is set.

f = 2 if code/status ends in 0, 1, 2, or 3 and end-of-information bit is not set.

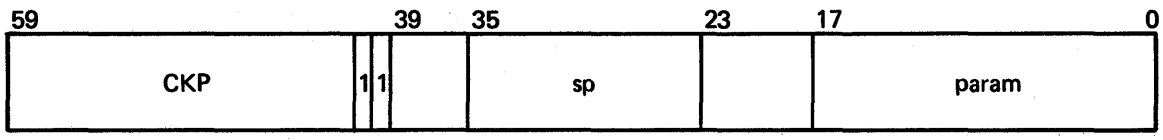
The following standard files, if they exist, are always copied to the checkpoint dump tape.

File	Default Copy Type
INPUT	2
OUTPUT	0
PUNCH	0
PUNCHB	3
LGO	3
CYBER Control Language Internal Files	2

The default copy type may be overriden by including the file name in the parameter list. For any file to be copied which is in neither the standard file list nor the parameter list, the copy type is f=3.

Generally, these values cause the entire mass storage file to be copied for: write operations, read operations resulting in end-of-information status, and rewind operations (excluding some OPEN functions).

The checkpoint macro generates the following code in X6 followed by a return jump to SYS=.



FILE ACTION MACROS

Each of the following functions addresses a file by its logical file name. A file environment table must exist for the file before its residence and use can be specified. The FET creating macros may be used, or the programmer can construct his own FET conforming to the format expected by the system.

When any file action request is issued, values are returned to the device type, disposition code, and FNT pointer fields in the FET.

All these functions, with the exception of READIN and WRITOUT, expand to a sequence of code that includes a return jump to routine CPC. READIN and WRITOUT bypass CPC by calling the random indexed record processors directly. For the other functions, CPC will call the appropriate PP routines to carry out the function specified.

Files manipulated by the following functions should not be manipulated by the functions described in the reference manual for the Record Manager within the same run.

The macros which call CPC are contained in the CPCTEXT system text overlay.

REQUEST MACRO

File residence can be specified by a REQUEST control statement or macro, with the same results.

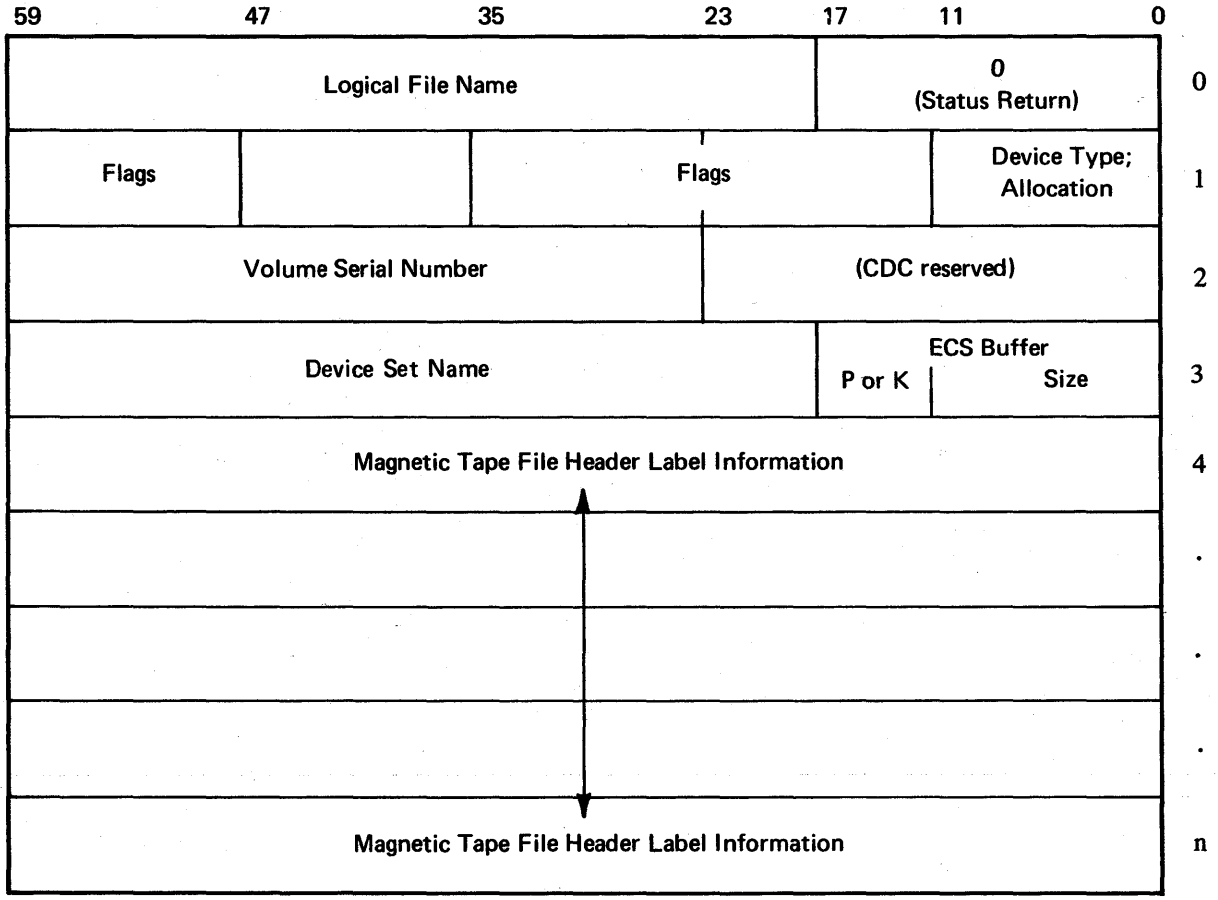
File action requests must reference the logical file name (lfn) of the file. If the file is a member of a multi-file set, all functions must reference the lfn of the set member. No function except REQUEST may be issued using the set name.

The REQUEST function informs the system of file characteristics.

```
REQUEST addr
```

addr is the first word of a variable length parameter list constructed by the user. The list must be at least two words long; maximum length required is that which supplies the parameters indicated by bits set in the flag field.

The parameter list must have the form shown below. The parameter list used with the REQUEST macro must be reinitialized after each call. Word 2, in particular, can be changed by the system during processing.



Parameters have the following meaning:

logical file name	Logical file name, left-justified, zero-filled.
status return	Initially, user should set to zero. The system returns the codes given below.
flag fields	Each bit is a flag for a particular condition listed below.
device type and allocation	Bits 6-11 are the octal device type code listed in the FET description of section 5; allocation styles of that device (except tape units) are installation defined.
volume serial number	Volume serial number identifying a particular device of a device set or a magnetic tape for automatic assignment. (Binary zeros in this field indicate a scratch tape.) When given, the VSN must be right-justified with display code zero fill.
device set name	1-7 letters or digits of device set name left-justified, blank-filled, with the first character alphabetic. Bit 17 of word 2 (addr+1) must be set if this parameter is given.
ECS buffer	If the file is to be buffered through ECS, bit 33 of word 2 (addr + 1) must be set. The size of the buffer must be in bits 0-11, with bits 12-17 showing a display code P if the size is in pages, or a K if the size is in thousands of words. The ECS parameter can also be used on a CYBER 170 Model 176 to request a specific number of LCM buffers for buffering data to the 819 disk. A request for an 819 device without the ECS parameter results in the default number of LCM buffers.

tape label fields — Label information for normal or extended label processing, formatted as shown below. Normal label processing is assumed unless bit 49 of word 2 (addr+1) is set.

The flags are individual bits that should be set to 1 to indicate the following conditions; otherwise the bits should be 0.

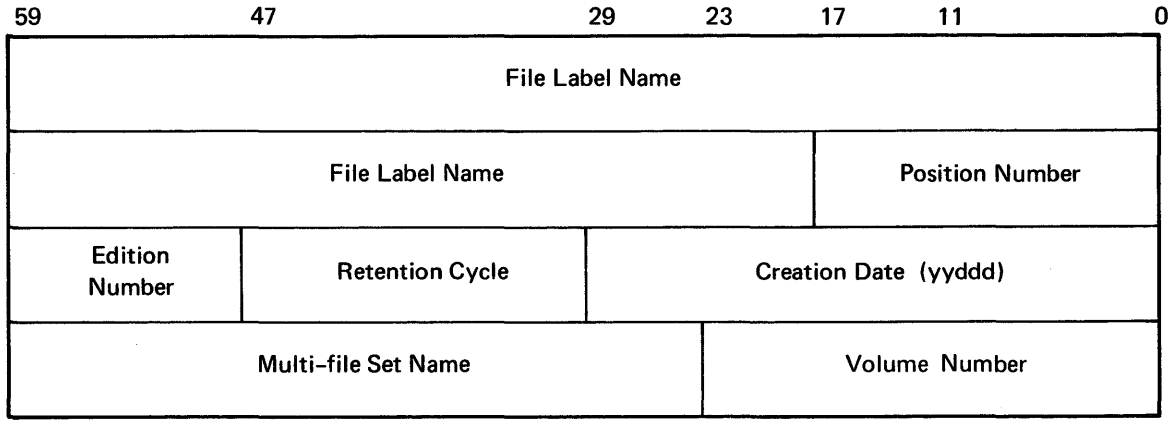
Bit	REQUEST Control Statement Equivalent	Meaning
55	*Q	1 = Assign file to queue device. Implies RMS device and causes automatic assignment. Not allowed for private device set.
54	IB	Inhibit system noise records.
53	NORING	Write enable ring prohibited in tape.
52	RING	Write enable ring required in tape.
51	MN	7-track or 9-track tape can be assigned.
50	A*	Assign any RMS device.
49	none	Parameter list words 5-9 have extended label processing format.
48	none	Parameter list words 5-9 have operating system label format.
33	EC	ECS buffering with parameters set in word 4. (Private device set files cannot be ECS buffered.)
32	OV	Overflow allowed to different device if that specified in word 2 is not available; if EP bit is set, a device capacity exceeded status is returned if no mass storage is available; permanent files overflow only to another permanent file device. OV implies RMS and forces automatic assignment.
31	PF	File must reside on a permanent file device. PF implies RMS and causes automatic assignment.
30	US	9-track tape conversion to ASCII codes.
29	EB	9-track tape conversion to EBCDIC codes.
28	*prefix to device type	Device to be assigned by system rather than operator (OV, PF, A*, *Q causes bit 28 to be set.)
27	none	Format of operator flashing message; if set, contents of RA+70 through RA+77 are displayed; if 0, REQ constructs the message from the REQUEST parameter list.

When this bit is set, the flashing B display message is not put in either job or system dayfile. Also, since the request parameters are extracted from the parameter list, the operator may see a flashing message which bears no relationship to the actual request; and as a result, may assign an incorrect device.

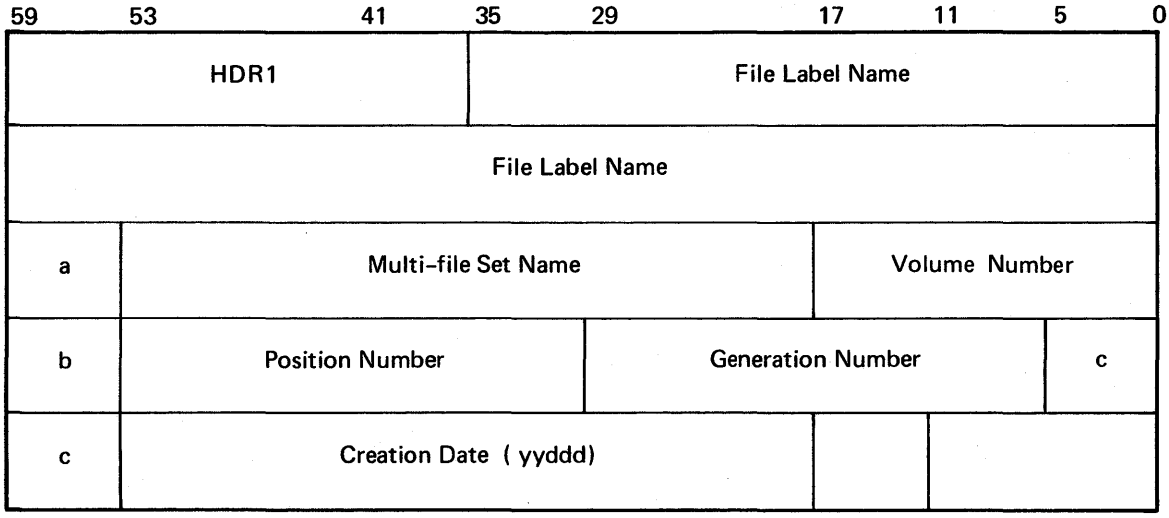
Bit	REQUEST Control Statement Equivalent	Meaning
26	2 prefix to device	Two magnetic tapes requested. (For two tape assignments, bit 28 and bit 25 are cleared.)
25	VSN	Word 3 contains a volume serial number for a magnetic tape or device set member. Bit 25 is cleared if bit 26 is set.
24	E	Magnetic tape is labeled currently.
23	NS	Non-standard labels on tape are considered data, not labels, by operating system. Not supported on SI tapes.
22	NR	Normal system tape read parity error processing is to be inhibited.
21	Z	Magnetic tape has Z format label of SCOPE 3.3, with character 12 of VOL1 label establishing data density.
20	none	Special return of error code to user; do not issue dayfile message or consult operator.
19	– –	Reserved.
18	MF	Request is for a multi-file set.
17	SN	Set name for a device set.
16	– –	Reserved.
15	absence of explicit density	Magnetic tape is to be written at system default density.
14	SV	Output tape to be saved.
13	IU	Inhibit physical unload of tape.
12	CK	Checkpoint tape request.

Format of the tape label fields depends on whether normal label processing is requested. The label fields must be in display code format, with acceptable values for each field, as detailed in section 3.

Label information for normal processing:



Label information for extended label processing:



a File label name continued

b Volume number continued

c Edition number

Once the REQUEST function is completed, bit zero of the first word of the parameter list is set to 1. In addition, bits 9-13 of word 1 may show one of the octal codes below. If so, the REQUEST function has been ignored and control returned to the program.

22 Recall bit was not set in call to PP routine REQ
24 File name table is full
26 Device of the requested device type is unavailable
30 File is already assigned to a device; the device type code is returned to the device type field of the parameter list

Two dayfile messages result from a successful REQUEST function. The first, directed only to the operator, contains parameters corresponding to those used in the internal parameter list. After assignment, a second message is written to the job and system dayfiles reflecting the assignement. For example, if a REQUEST function is made with dt set to zero, the operator display shows no device type. If the operator assigns a 7-track tape, however, the mnemonic MT appears in the job dayfile message.

Conflicts between dt requested and dt assigned by the operator must be resolved by the operator using the n.YES or n.NO type-in.

OPEN AND CLOSE FUNCTIONS

Two functions are available for opening files:

OPEN is applicable to all files

POSMF is applicable only to labeled multi-file tapes

Files can be closed with the following functions:

CLOSE is applicable to all files

CLOSER is applicable to sequential files on tape or on a device set; it gives the user control over end-of-volume processing

OPEN MACRO

An OPEN function is a file initialization and status checking operation. The user must issue an OPEN if:

Random files are to be processed by the user or system

User label processing is to follow

Sequential files are to be rewound without a REWIND function being issued

Otherwise, OPEN is not necessary. If an OPEN function is to be issued, it should be the first function issued on a given file; otherwise the effect of the OPEN function is undefined.

```
OPEN  lfn,x,recall
```

The x parameter may be any of the ten values:

absent	NR
READ	READNR
REEL	REELNR
ALTER	ALTERNR
WRITE	WRITENR

The WRITE or WRITENR values of x may be used to ensure that the file circular buffer is emptied if the job terminates abnormally before buffer contents have been transferred to an output device. The first data function following these OPEN functions must not then be a read or a forward motion function.

If the value of x is READ, REEL, ALTER, WRITE, or absent, sequential files are rewound. Any other value of x is not repositioned the file.

When an OPEN is issued, the following events occur:

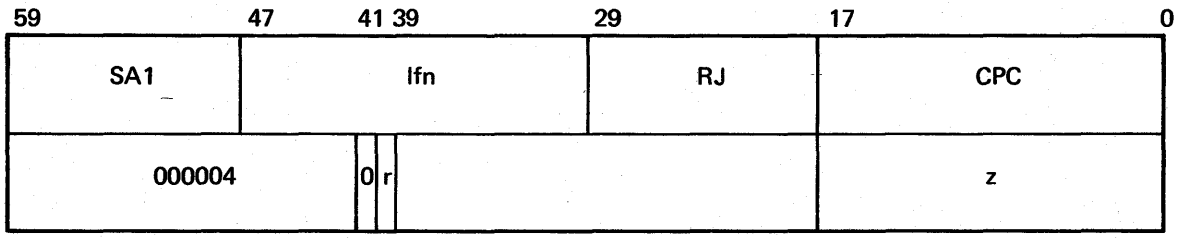
For sequential files, file position is changed to beginning-of-information unless a no rewind is specified by using an x parameter ending in NR. The r bit in the FET is set to zero.

For labeled magnetic tape files, processing depends on the presence or absence of the XL bit in the FET. If the XL bit is set, all labels are written from or delivered to the file label buffer. If the XL bit is off and labels are being written, the HDR1 label is formatted from data in the FET label fields. If labels are being read (XL off), the HDR1 label is returned to the FET label fields.

For random files, if the r bit is set, any existing index is read into the index buffer. If the index record is shorter than the buffer, unused buffer space is set to zeros. If the r bit is not set, an existing index is not read.

For all files, the physical record unit and record block sizes are returned to FET fields.

The macro OPEN generates the following code:



The z field depends on x.

160 if x is absent or ALTER
140 if x is READ
340 if x is REEL
144 if x is WRITE
120 if x is NR or ALTERNR
100 if x is READNR
300 if x is REELNR
104 if x is WRITENR

There is no difference in the action taken by the system for codes 160, 140 and 340; or for codes 120, 100 and 300.

POSMF MACRO

The POSMF function positions standard labeled multi-file sets. The multi-file set to be positioned is specified by the multi-file name in the label fields of the FET or, if the XL bit is on, in the HDR1 label field in the extended label buffer. The named multi-file set is positioned to a particular file and an OPEN with rewind function is performed. If a position number is specified in the label field or label buffer, that number specifies the file to be opened.

POSMF mfn,recall

mfn FET name of multi-file set

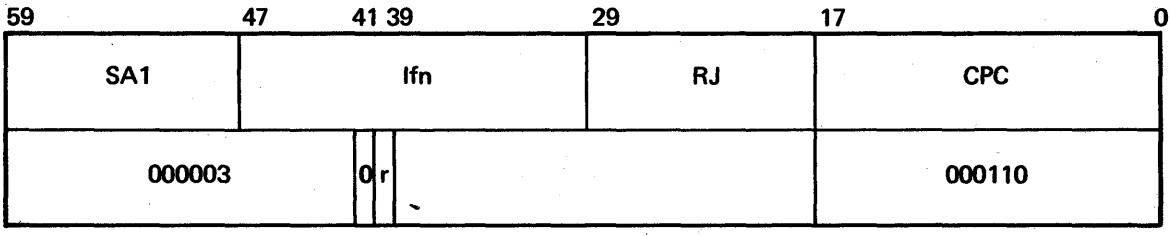
recall Non-blank value if for auto recall; otherwise, blank

The position number is specified in either word 11 of the FET or the extended label buffer, depending on the label processing to be performed. If normal label processing is to occur, bits 0-17 of word 11 of the multi-file name FET may contain the position number (position numbers begin with 1 for the first file). For extended label processing to occur, the XL bit must be set (bit 41 at mfn+1). The position number is expected to be in the ANSI standard position field of a record formatted as an HDR1 label within the label buffer. A fatal error exists if HDR1 is not found within the label buffer.

If the position number is 0, the set is positioned at the beginning of the next file. OPEN procedures for an existing file follow. If the position number is 999 in the FET or 9999 in the label buffer, the set is positioned after the last member file and OPEN procedures instituted for a new file.

End-of-set status (21 in bits 9-13 of mfn) is returned to the FET for the multi-file set if the explicit or implied position number is greater than the last member of the set. The position field in the FET will be one greater than that of the last member file.

The POSMF macro generates the following code:



CLOSE MACRO

A CLOSE function is a file terminating operation. The user must issue a CLOSE if:

Random files have been created or modified and a valid index is to be saved

End-of-job procedures listed below are to be initiated for a file before the actual end-of-job

Otherwise, a CLOSE is not necessary.

```
CLOSE  lfn,x,recall
```

The x parameter may be:

absent
NR
UNLOAD
RETURN

If the value of x is absent, UNLOAD, or RETURN, the file is rewound. NR specifies that the file is not to be rewound. Both of these positionings are possible only with sequential files; positioning is not defined on files for which an index is written.

When a CLOSE is issued, the following events occur:

For sequential files, position will be changed according to the rewind associated with the x parameter.

For labeled magnetic tape files, action depends on the x parameter. If no rewind is specified and the file is positioned after a newly written record, a file mark and an EOP trailer label is written, then the file will be positioned immediately before the file mark. If the file is to be rewound and it is positioned after a newly written record, an EOP trailer label is written before the rewind is initiated.

For unlabeled S and L tape files, four tape marks are written instead of an EOP trailer label. Otherwise, processing is the same as for labeled tape files.

For random files, the index is written as the last system-logical-record if the FET r bit is set, an index buffer is specified, and file contents have been altered since the last OPEN function was issued.

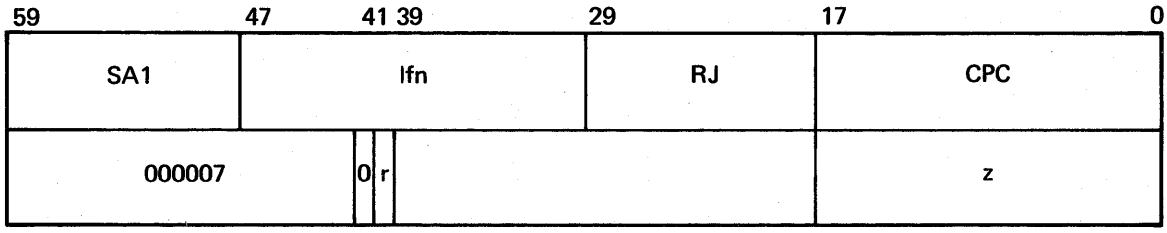
The user must empty the file circular buffer when files are being written; CLOSE does not empty the buffer.

When CLOSE/RETURN or CLOSE/UNLOAD is issued, end-of-job processing procedures occur for the named file.

Permanent files are detached from the job.

For magnetic tape files, a CLOSE/RETURN decreases the number of tape units required by the job as indicated with the MT or NT parameter on the job statements. A CLOSE/UNLOAD does not decrease this value. A CLOSE/UNLOAD or CLOSE/RETURN function issued on a member of a multi-file set acts as a CLOSE/REWIND on that member.

The CLOSE macro generates the following code:



The z field depends on x.

150 if x is absent
130 if x is NR
170 if x is UNLOAD
174 if x is RETURN

CLOSER MACRO

Processing of both magnetic tape and device set files continues across volume or device boundaries when data is skipped in a forward direction, read, or written. With the UP bit of the FET the user can request notification when a boundary is about to be crossed; and volumes or devices can be processed in other than ascending order.

The CLOSER function affords a degree of user control over processing at end-of-volume or end-of-device:

CLOSER lfn,x,recall

The x parameter may be:

absent	Rewind
NR	No rewind
UNLOAD	Rewind and unload
RETURN	Rewind and unload; do not swap reels

MAGNETIC TAPE PROCESSING

For magnetic tapes, the system initiates volume swapping if the UP bit is 0 when CLOSER is issued. The file is positioned on the next volume and file operations can continue normally. An OPEN function is not required for the second volume, but may be issued if the program is to receive the header label contents.

A volume swap is performed by the following steps:

1. If the tape is positioned after a newly written record, a volume trailer label is written.
2. The tape is unloaded and the operator is notified that processing on that volume is completed.
3. If two units were assigned to the file, unit numbers are interchanged so processing continues without changing tables referencing the unit.
4. The volume number of a labeled file is incremented by one in the system label table and, if declared, in the user's FET label fields.
5. The FET completion bit is set. End-of-volume status is not returned.

If the UP bit is set to 1 when the CLOSER is issued for a tape file, the user may specify the next volume to be processed. The following occurs:

1. If the tape is positioned after a newly written record, a volume trailer label is written.
2. The tape is rewound or rewound/unloaded according to the CLOSER parameter.
3. The operator is notified that processing on that volume is completed.
4. If two units were assigned to the file, unit numbers are interchanged.
5. The end-of-volume status and completion bits are set.

To establish the next volume to be processed, the user must enter the volume number in the FET label field (bits 0-24 in word 13) before another function is issued to the file. A following OPEN function is not required unless the program uses the header label of the new volume.

When CLOSER/RETURN is issued, normal end-of-volume processing is performed regardless of the UP bit. Instead of swapping to the next volume as in the CLOSER macro, the file is returned (disassociated from the job).

End-of-volume processing for CLOSER/RETURN is performed by the following steps:

1. If the tape is positioned after a newly written record, a volume trailer label is written.
2. The reel is unloaded according to the IU parameter on the REQUEST statement.
3. The FET completion bit is set; end-of-volume status is not returned.
4. The FNT entry is cleared and FET IN/OUT pointers are set to FIRST.

ROTATING MASS STORAGE DEVICE PROCESSING

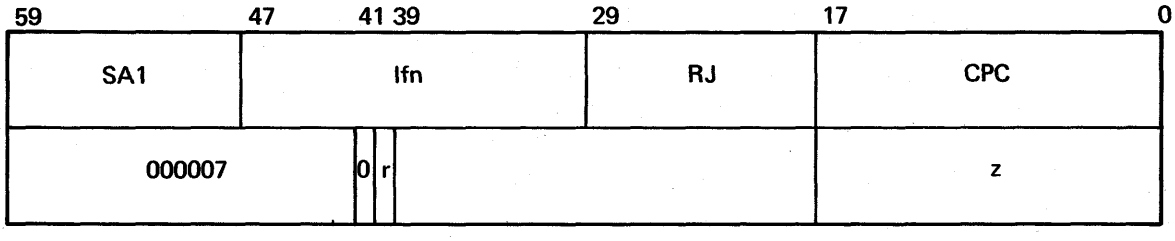
For an RMS device, the operating system performs the following:

1. If the file is at EOI and the last RBT word pair is not an overflow word pair, an EOI status (bit 9) is returned in the FET and FST; and an overflow word pair is added at the end of the RBT chain.
2. If the file is at EOI and the last RBT word pair is an overflow word pair, an EOI status is returned in the FET and FST.
3. If the file is not at EOI but positioned on an overflow word pair, the current position in the FST is updated and points to the RBT word pair following the overflow word pair.
4. If the file is not an EOI and is not positioned on an overflow word pair, the system skips to the next overflow word pair or to EOI (whichever it finds first) and then takes action as described above.

In all cases, the completion bit is set in the FET and FST and the end-of-device status (bit 10) is set in the FST. If the UP bit is set in the FET, then the end-of-device status is also returned in the CS field of the FET.

Processing continues on the next device as soon as the user issues another input/output function for the file.

The macro for CLOSER generates the following code:



The z field depends on x.

350 is x is absent
330 if x is NR, although result is the same as 350
370 if x is UNLOAD
374 if x is RETURN

READ FUNCTIONS

Six read functions are available for bringing information into central memory. The functions, and the main distinctions among them, are:

READ	Applicable to all mass storage and tape files. Reading stops when the end of a physical record or the end of a system-logical-record of level 0-16 is encountered.
READNS	Applicable to mass storage files only. Read does not necessarily stop at end-of-logical record.
READSKP	Similar to READ, but positions file to beginning of next logical record when the circular buffer is filled.
RPHR	Applicable to magnetic tapes in SI format only. Reads the next PRU delivering coded data in internal BCD codes (7-track). For 9-track SI tapes, the data is read in packed mode and delivered with no conversion.
READN	Applicable to magnetic tapes in S and L data format only.
READIN	Applicable to all mass storage and tape files.

All of these functions read information into the file circular buffer, with the amount of information read dependent on the specific function and the size of the buffer. As information is read into the buffer, operating system routines change the value of the IN pointer. This value, minus 1, is the address of the last word read. The user is responsible for using the IN pointer while removing information from the buffer, and for setting the OUT pointer to reflect the move, except when the READIN macro is called. READIN, like WRITOUT, relieves the user of responsibility for IN and OUT pointer manipulation. By means of a secondary buffer called a working storage area, READIN maintains circular buffer pointers.

As processing progresses, status information is returned to the code and status field of the FET. If the user has the EP bit set, control returns to his program for OWNCODE routine execution when file action errors occur. Otherwise, the operator is notified and given the option to drop the job.

The 18 bit code and status field will show the values listed below for the conditions that cause various read functions to terminate. Bits in the field have the purposes:

Bits 14-17	System-logical-record level number
Bits 9-13	File action error code
Bit 4	End-of-logical-record indicator
Bit 3	End-of-partition indicator if bit 4 is set
Bit 1	Mode indicator: 0 for coded, 1 for binary
Bit 0	Complete bit

For binary files, the low order octal digit of the code and status is 3 instead of 1.

Condition	Code/Status Setting for Coded Files
End-of-information encountered	741031
Zero-length PRU of level xx is read	xx0021
Level 17 system-logical-record or level 16 mass storage file read with READNS	740031
Next PRU will not fit into circular buffer	000011
Unrecoverable file action error code ee	0ee011

File action error codes are listed in the Error Exit Address field in the FET discussion of section 5.

When a read for a file is issued without recall, the IN pointer is updated as each PRU of data is moved to the buffer, allowing the user to remove data as fast as it is placed in the buffer. When the request is issued with recall, the pointer is not changed until the request is complete. For magnetic tape, the code status (bits 11, 12) is set for each record before the IN pointer is moved. Tapes can be read dynamically as follows:

EP must be on.

Check to determine if IN has moved; if not, repeat check.

When IN has moved, check CS field for errors. If none, process record. If errors occurred, wait for complete bit to set.

For S and L format files, the UBC field is set as a record is read.

All the following read functions, except READIN, expand to a two-word sequence of code which includes a return jump to routine CPC. The READIN function expands to call routine IO or IORANDM, which calls CPC.

Parameters appearing in the macros are:

lfn	Logical file name
recall	Optional recall parameter of any letter or digit

READ MACRO

The READ function is applicable to all types of files. READ causes information from the specified file to be placed in the circular buffer for the file in central memory.

```
READ  lfn,recall
```

Reading begins as long as the circular buffer has room for at least one physical record unit. It continues until:

The next PRU will not fit into the circular buffer.

End-of-logical-record or end-of-partition is encountered.

End-of-information is encountered.

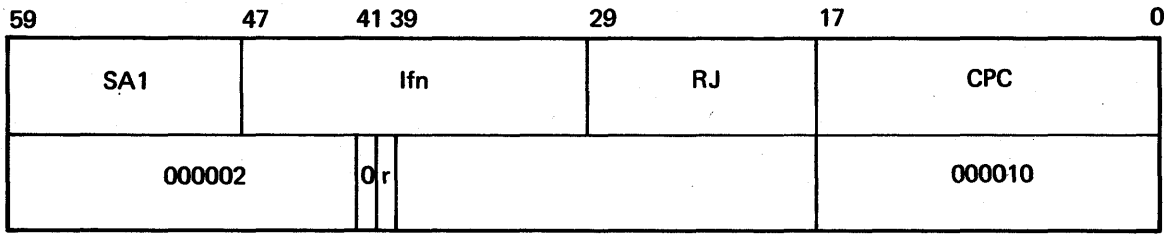
File action error occurs.

For S and L tapes, one physical record is read.

If the end-of-logical-record bit (bit 4 of word 1 of the FET) is set when READ is called, CPC ignores the request.

For S and L tapes, the unused bit count is returned to the UBC field in the FET word 7 when the read is complete.

The READ macro generates the following code:



READNS MACRO

The READNS function is applicable only to mass storage files. A single READNS often results in more information being transferred to the circular buffer than a READ issued to the same file since reading does not necessarily stop at the end of a logical record.

READNS lfn,recall

Reading begins if the circular buffer has room for at least one physical record unit. Reading continues until:

The next PRU will not fit into the circular buffer.

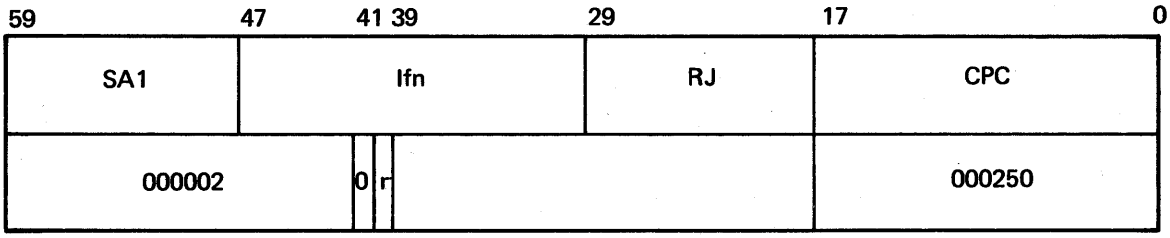
Zero-length system-logical-record of any level is read.

Level 16 or 17 system-logical-record is read.

End-of-information is encountered.

File action error occurs.

The READNS macro generates the following code:



READSKP MACRO

The READSKP function is applicable to all types of files. READSKP is used to identify and skip records. Reading continues until an end-of-logical-record is encountered, or the circular buffer is full. Once the buffer is full, the file is repositioned to the beginning of the next record. READSKP is halted by any conditions which halt a READ.

READSKP lfn,lev,recall

lfn Logical file name

lev Optional level number 0-17. Default value is 0.

recall Optional recall indicator.

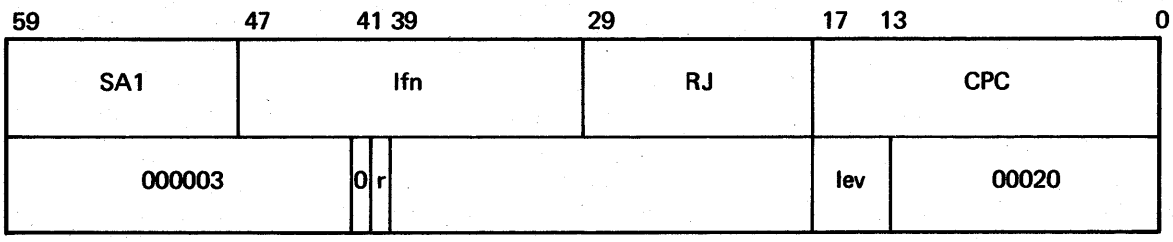
If a level parameter lev is specified for SI tapes or mass storage files, information is skipped until the occurrence of an end-of-logical-record with a level number greater than or equal to the one specified. For S and L tapes, only a request with level 17 is recognized; any other level in the request is ignored.

When the READSKP is executed, the end-of-logical-record bit (bit 4 in word 1 of FET) is set, since an end-of-logical-record is encountered in the skip to the beginning of the next record. This bit must be cleared by the user program before a subsequent READ, but not a READNS, is issued. When EP=1, a READ error prevents the skip; and control returns to the user.

For S and L tapes, the user should set the MLRS field before the READSKP is issued. If this field has a 0, the system sets it to 512 words for an S tape and to LIMIT–FIRST–1 for an L tape.

An end-of-volume condition on a magnetic tape file with the UP bit set terminates the skip of a READSKP even if the beginning of the next record has not been encountered. Otherwise, volume swapping takes place under system control.

The READSKP macro generates the following code:



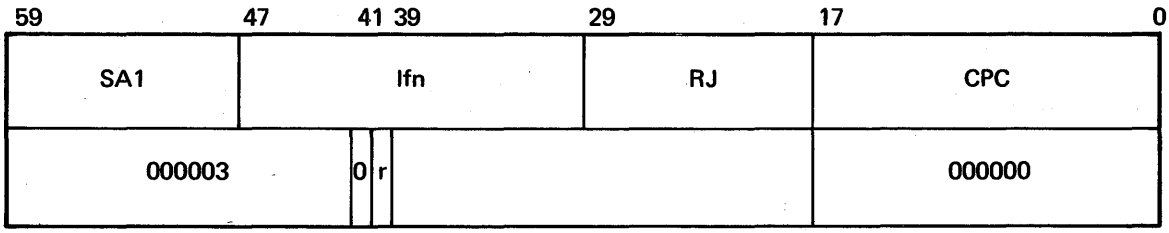
RPHR MACRO

The RPHR function is applicable only to magnetic tapes in SI format. RPHR causes all information existing in the circular buffer to be discarded and the next PRU to be read into the buffer.

```
RPHR  lfn,recall
```

For coded 7-track files, data is converted from external to internal BCD only. Conversion to display code is not made. No conversion takes place for 9-track tapes; the data appears as written. SI tapes are always written to contain exact multiples of central memory words by filling the last word with zeros.

The RPHR macro generates the following code:



READN MACRO

The READN function is applicable only to magnetic tape in S or L format. READN allows maximum tape throughput; as long as the user provides space in the circular buffer for two records and their header words, tape reading continues without releasing and reloading the read routine between physical records. This gives maximum utilization of interrecord gap time. The minimum buffer size for reading an S or L tape should be two words more than the maximum logical record size (MLRS field of the FET).

```
READN  lfn,recall
```

Before this function is issued, the MLRS field of the FET (bits 0-17 of word 7) must be set to the largest physical record that will be encountered. File mode must also be set.

Reading continues until:

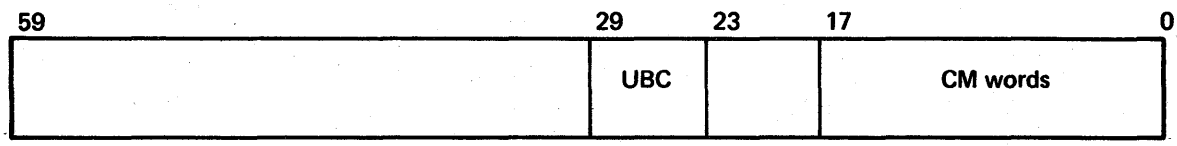
The next record will not fit into the circular buffer.

End-of-partition is encountered.

End-of-information is encountered.

File action error occurs.

The header word that precedes each physical record in the circular buffer is generated by the system; it does not exist on the tape. The format of the header word is:

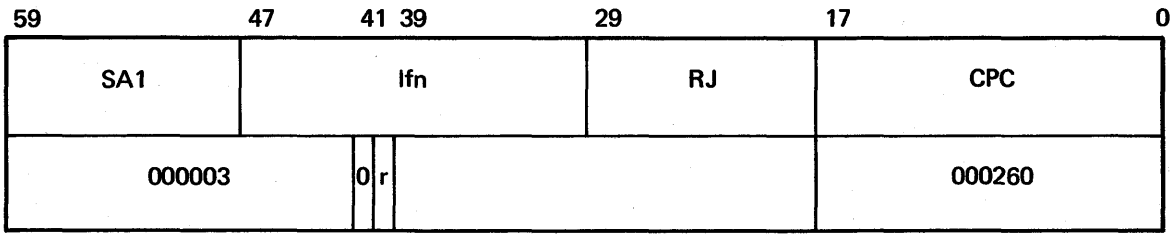


CM words Number of 60-bit words in the physical record

UBC Number of bits in the last word that are not valid data

After each complete physical record has been placed in the buffer, the system moves the IN pointer to reflect both the header and data.

The READN macro generates the following code:



READIN MACRO

The READIN function is applicable to all mass storage and tape files. READIN employs a user-provided working storage area as well as the file circular buffer. The user deals only with data in the working storage area; the system handles the circular buffer and the IN and OUT pointers of the FET.

Format of the READIN macro depends on the structure of the file being accessed. The second parameter is required only if the file is a random indexed file with a name or number index.

When READIN is executed, data from the circular buffer is placed in the working storage area. The amount of information transferred depends on file mode:

For binary files, READIN fills the working storage area unless an end-of-logical-record or end-of-information is encountered before the area is full.

For coded files, information is moved to the working storage area until a 12-bit zero byte in the low order bits of a word (end-of-line indicator) is encountered or the working storage area is full. When a zero byte is encountered, two blanks are substituted and the remainder of the area is filled with blanks. If a zero byte is not met before the working storage area is full, the remainder of the line is skipped. The next READIN request obtains the next line rather than the end of the first line.

READIN issues calls to READ through CPC as needed. If the data in the buffer does not satisfy the READIN request, a READ with recall is issued. Therefore, the user does not gain control until his request is satisfied.

If a working storage area is not specified, a READIN request has no effect, except as described below for indexed random files.

READIN makes a check of the I/O progress immediately prior to returning to the user program. A READ without recall is issued if the circular buffer is not already busy and it is more than half empty, so that input/output is buffered with subsequent computing by the user program.

Sequential or random files are read with the following macro:

```
READIN  lfn
```

When an end-of-logical-record or end-of-partition is encountered during a read of a sequential file, the user regains control immediately, with the X1 register showing the state of the request. Filling of the working storage area ceases. The next READIN request begins with the next record.

Status information in the X1 register may be:

positive zero	Requested number of words was read and the function completed normally.
positive non-zero	The working storage area was not filled because the remainder of the logical record contained too few words when the READIN was issued. X1 contains the address of the first unfilled word, or if no data was transferred, the first word address. For coded files, this is always the first word address.
negative non-zero	No data was transferred to the working storage area because an end-of-partition or end-of-information was encountered.

When an indexed random file has named or numbered records, READIN positions the file to the desired record.

```
READIN  lfn,/name/

READIN  lfn,n
```

/name/	Name of record
n	Number of record

When a READIN is issued for such an indexed random file, the current contents of the circular buffer are destroyed when the IN and OUT pointers are set equal. Then, the mass storage address corresponding to the record number or name is copied from the index into FET word 7, and a READ request with recall is passed through CPC. On return from the READ, the procedures for a READIN without a name or number parameter are followed. If a working storage area is specified in the FET, the beginning of the record is copied into it and the FET pointers are adjusted. If no working storage area is specified, no further action occurs; however, the file has been positioned and reading of the desired record has been initiated by READIN.

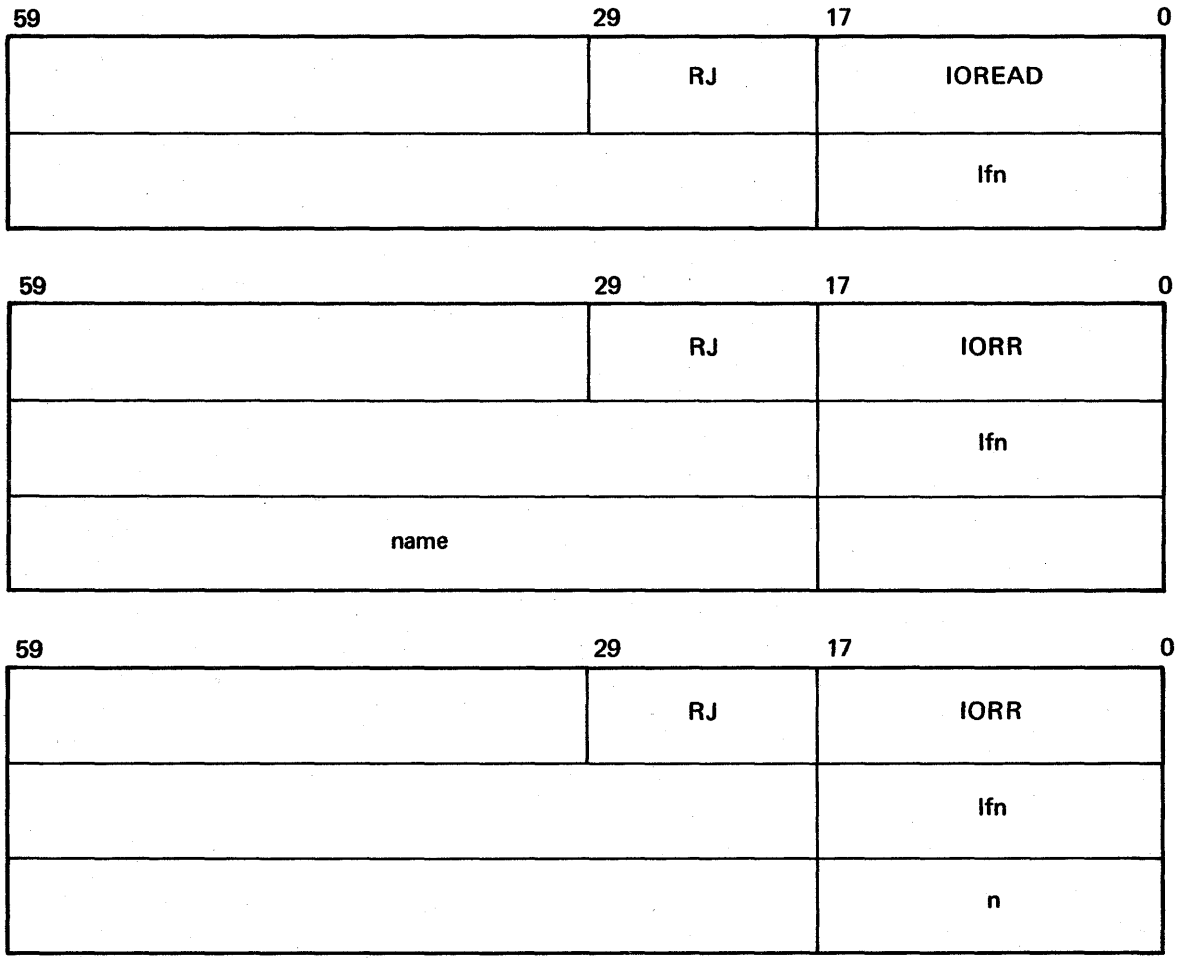
Any remainder of the record can be read by subsequent READIN requests that do not identify the record by name or number. After an end-of-logical-record is encountered on a random file, further READIN requests specifying only a file name will not initiate reading of the next record, as they would on a non-random file. To start reading the next record, or some other record on a random file, a READIN with a record name or number must be issued.

When a record is located by a READIN request containing its name or number, the number of the record is stored in word 8 of the FET, making it possible to read the next record with:

```
READIN  lfn,0
```

The system interprets this statement as record n+1. Consequently, by starting a new record with a request that identifies record number zero, the list of records as given in the index can be read. However, if the calling program did not stop before overshooting the end of the index, there would be an error return from READIN on the last+1 record.

The code generated by the READIN macro depends on the second parameter. For no parameter, a name parameter, and a number parameter, respectively, the code is:



WRITE AND REWRITE FUNCTIONS

Information is transferred from the file circular buffer to a storage device when one of the write functions is issued. These functions, and the main distinctions among them, are:

WRITE	Applicable to mass storage and tape files; writes at end-of-information
WRITER	Applicable to mass storage and SI tapes; writes a short or zero-length PRU to indicate end-of-logical-record

WRITEF	Applicable to mass storage and magnetic tape files; writes an end-of-partition indicator
WPHR	Applicable to magnetic tapes in SI format only; writes a single physical record; the only write function that expects coded data in internal BCD format. No conversion is performed for 9-track coded tapes.
WRITEN	Applicable to S and L data format tapes only
WRITOUT	Applicable to mass storage and tape files; the only write function in which the system, rather than the user, manipulates the buffer pointers of the FET
REWRITE	Applicable to mass storage only; rewrites record of same length
REWRITER	Applicable to mass storage only; writes an end-of-logical-record indicator for a rewritten record
REWRITEF	Applicable to mass storage only; writes an end-of-partition for a rewritten file
WRITIN	Applicable to mass storage file to be rewritten only; analogous to WRITOUT using REWRITE rather than WRITE

The system sets the OUT pointer when data is removed from the buffer. The user must manipulate the IN pointer as he places information in the buffer, as explained under Circular Buffer Use in section 6.

When S and L tapes are being written, the MLRS and UBC fields in the FET must be set by the user to indicate the size of the record before a write is issued.

Status information and error codes are returned to the first word of the FET as the file is written. If the user has the EP bit set, control returns to his program for OWNCODE execution when file action errors occur. Otherwise, operator is notified and given the option to drop the job.

Parameters that appear in the write macros are:

lfn	Logical file name
recall	Optional recall parameter consisting of any non-blank letter/digit character string

WRITE MACRO

The WRITE function transfers information from the file circular buffer to the file storage device. WRITE is applicable to both mass storage files and tapes.

```
WRITE  lfn,recall
```

For mass storage files and tapes in SI format, only full PRU's are written. The size of the PRU depends on the storage device. Writing continues until:

The buffer is empty.

Data in the buffer does not fill a PRU.

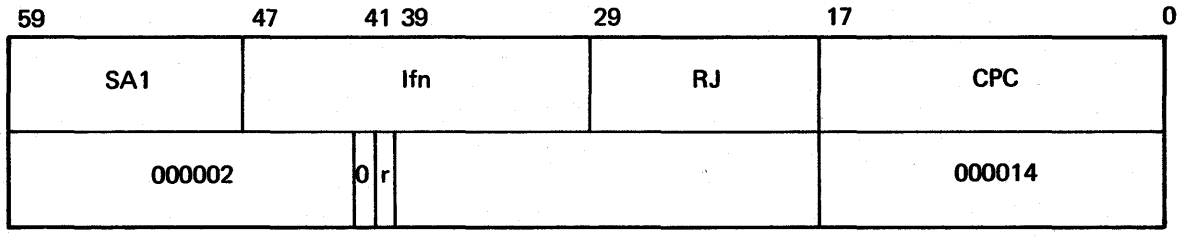
A following WRITER request will empty the buffer.

For tapes in S or L format, only one record is written for each request. The length of the record is determined by the value of the IN and OUT pointers. If the record length exceeds the MLRS field value (bits 0-23 of word 7) in the FET, the job terminates with an error.

When a WRITE function is completed on any type of file, end-of-information (EOI) is established immediately after the position just written. Any information that may have existed beyond that point on the file is lost. When the FET random bit is on, the file is positioned at EOI before writing is done. On permanent files, a WRITE function is permitted only at EOI.

A REWRITE function is used to modify data in the middle of a mass storage file, the WRITE function cannot accomplish such action.

The WRITE macro generates the following code:



WRITER MACRO

The WRITER function causes the circular buffer to be emptied and an end-of-logical-record indicator to be written. For mass storage files and tape files in SI format, a short or zero-length PRU is written. For S and L format tape files, WRITER is equivalent to WRITE.

WRITER lfn,lev,recall

lfn — Logical file name

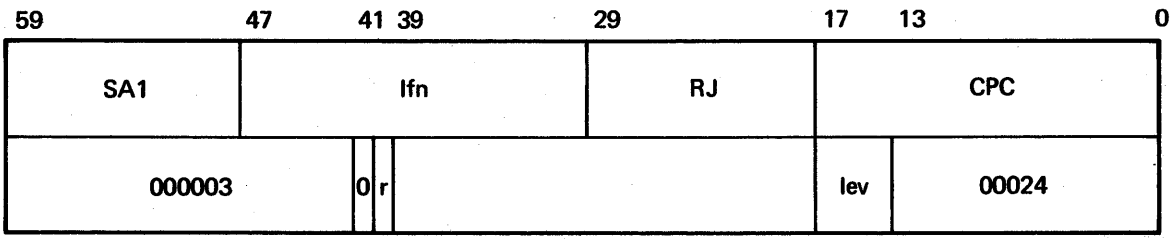
lev — Optional level number 0-17. Default value is 0.

recall — Optional recall indicator

WRITER is processed the same as WRITE, with the following additions:

For mass storage files and tapes in SI format, the data in the circular buffer is written out followed by an end-of-logical-record marker. A zero-length PRU is created if necessary; otherwise a short PRU exists. If the level parameter is present, it is included. If the buffer contains no data when WRITER is issued, a zero-length PRU is created. If the specified level number is 17, the system changes the WRITER request to a WRITEF.

The WRITER macro generates the following code:



WRITEF MACRO

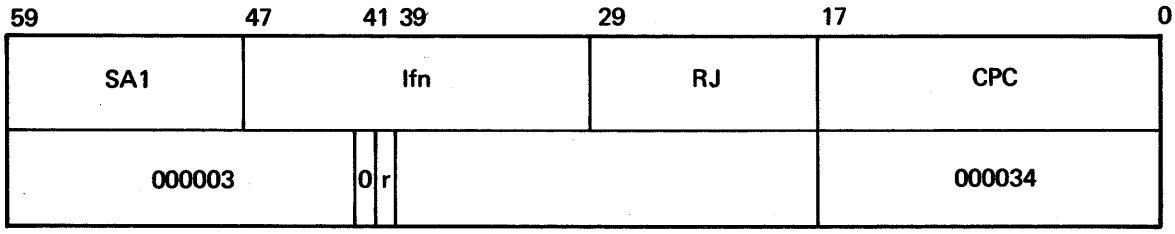
The WRITEF function produces an end-of-partition. Any information in the buffer is written out before the end-of-partition is written.

```
WRITEF  lfn,recall
```

For mass storage files and tapes in SI format, WRITEF produces a zero-length PRU of level 17. Data in the buffer is written out and terminated by a zero level end-of-logical-record before the zero-length PRU is written. If the buffer is empty and the last operation was a write, a zero-length PRU of level 0 is written before the level 17.

For S and L tapes, data in the buffer is written to tape and followed by a physical tape mark.

The WRITEF macro generates the following code:



WPHR MACRO

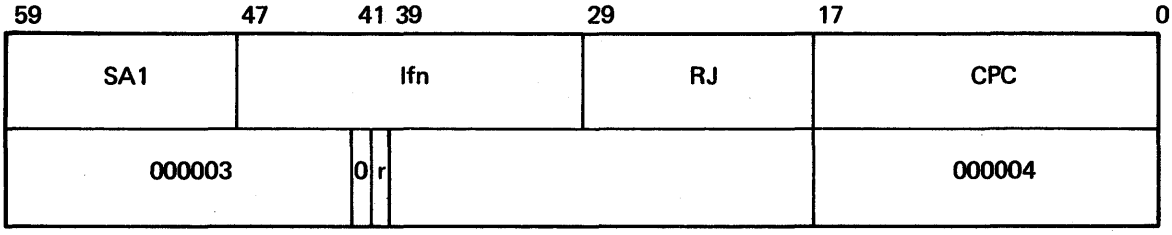
The WPHR function is applicable only to magnetic tape in SI format. It causes all information in the circular buffer, to a limit of 512 words, to be written as a single physical record. Data to be written to 7-track tape must be in internal BCD codes. Only internal to external BCD conversion is performed before writing; no conversion is performed for 9-track tapes.

```
WPHR  lfn,recall
```

If the buffer contains fewer than 512 (decimal) words, the IN and OUT pointers in the FET are set equal when writing is completed to show an empty buffer. If the buffer contains more than 512 words, only the first 512 words are written. The IN and OUT pointers will be set by the system to show that more data exists in the buffer. Status returned is 10, indicating device capacity exceeded.

A WPHR issued for any device other than magnetic tape in SI format is ignored. A 22 status is returned to show an illegal function call, terminating the job.

The WPHR macro generates the following code:



WRITEN MACRO

The WRITEN function is applicable to magnetic tape in S or L format only. It allows maximum use of the interrecord gap time as long as the user provides at least two records and their control words in the circular buffer.

```
WRITEN  lfn,recall
```

Writing continues until:

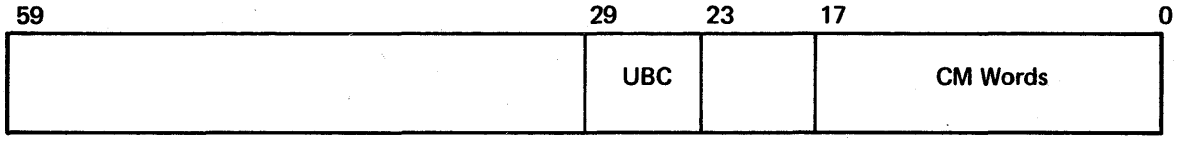
Buffer is empty.

End-of-volume is encountered.

File action error occurs.

No action takes place if the buffer is empty.

The user must provide a header word immediately preceding each record in the buffer. This header is not physically written on the tape. Its format is:



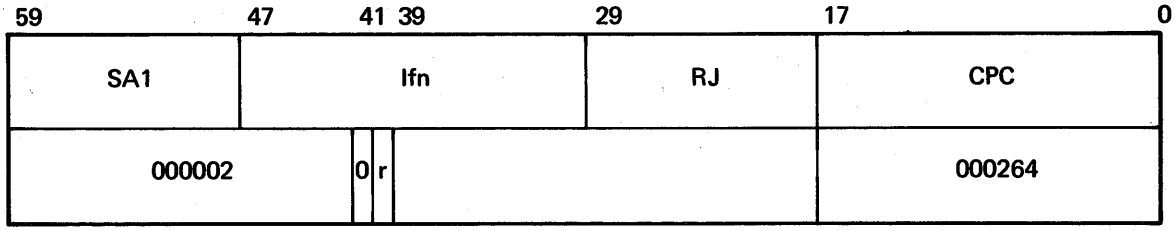
CM words Number of 60-bit words in the physical record

UBC Number of bits that are not valid data in the last word

The system compares the MLRS and UBC fields in the FET using information from this header.

The OUT pointer is not changed to reflect the move until after each complete record has been written to tape. The user should not move the IN pointer beyond the header word until the header and the complete record are in place, or an error will result.

The WRITEN macro generates the following code:



WRITOUT MACRO

The WRITOUT function is applicable to all mass storage and tape files. It employs a user-provided working storage area as well as the file circular buffer; when the buffer is full the system issues a WRITE function to transfer data from the buffer to the file storage device. With random indexed files, the user has the option of using either WRITOUT to position a file and manage the circular buffer himself, or providing a working storage area and letting the system manage the buffer. Otherwise, the user deals only with data in the working storage area; the system handles the circular buffer and both the IN and OUT pointers of the FET.

When WRITOUT is executed, data in the working storage area is transferred to the circular buffer. No record boundaries are assumed, with all data placed in the buffer by WRITOUT being considered a single logical record. Until the user issues a WRITER or WRITEF to empty the buffer, a single record exists. The system empties the buffer as necessary to accommodate new data being moved into the buffer. As with the READIN function, the system buffers input/output with computing by checking the buffer just before returning to the calling program, and issuing a WRITE without recall if the buffer is more than half full. WRITE functions with recall are issued when it is necessary to empty the buffer before carrying out the WRITOUT request, so that the WRITOUT function completes before control returns to the user program.

The amount of data transferred from the working storage area to the buffer depends on the file mode:

For binary mode files, the entire working storage area is transferred.

For coded mode files, trailing blanks are removed and a 12-bit zero byte is inserted in the low order position of a word to indicate end-of-line.

Sequential files are written with:

```
WRITOUT  lfn
```

A WRITER function must be used to terminate a record. If a working storage area does not exist for a sequential or random file, the WRITOUT is ignored with no error indication.

An additional parameter is required when the file has indexed records. To declare the beginning of an indexed record, one of these forms of the macro is used:

```
WRITOUT  lfn,/name/

WRITOUT  lfn,n
```

/name/ Name of record

n Number of record; if n is 0, the number is one greater than the last number, with the first record being numbered 1

To continue writing the same record, this form is used:

WRITOUT lfn

To terminate the record, the WRITER macro should be used, although the system issues WRITER under circumstances noted below.

WRITER lfn

An alternate method of processing indexed records is to use WRITOUT with a record identifier, then fill the circular buffer directly and issue a WRITE request without using the working storage area. A WRITER request is still needed to terminate the record.

When a WRITOUT identifying an indexed record is issued, the system performs the following:

If the buffer contains data from a previous WRITOUT, or the last operation was a completed write rather than write end-of-logical-record, a WRITER occurs.

The IN and OUT pointers are set equal to indicate an empty buffer and the FET status is set to show that write was completed.

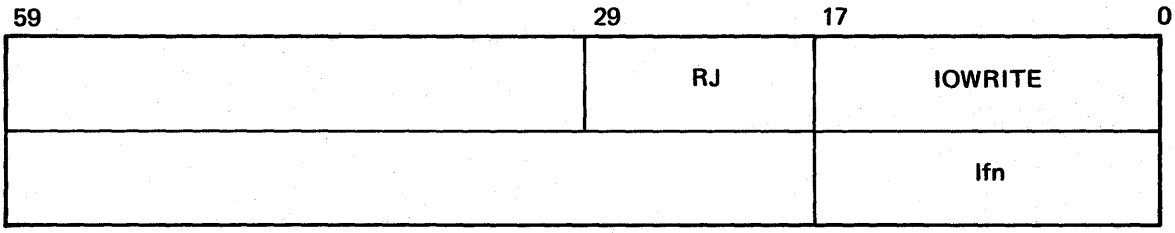
The random file index and the eighth word of the FET are set to the correct record.

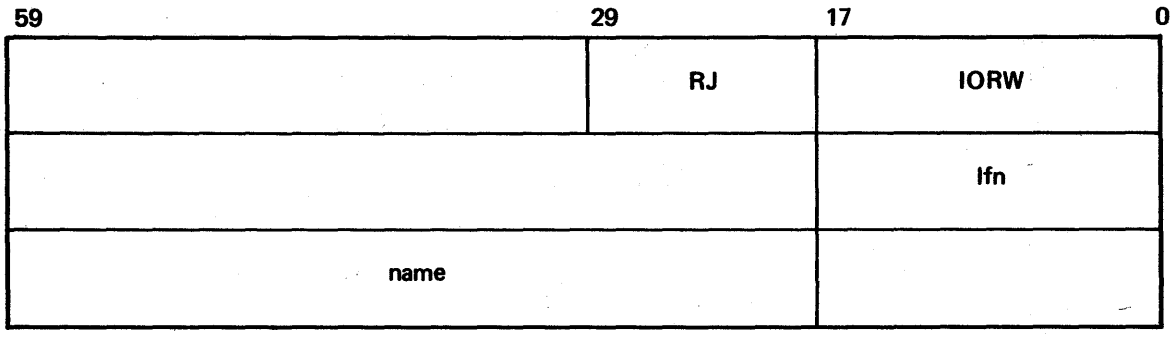
The working storage area is transferred to the circular buffer as the beginning of the new record identified in the WRITOUT.

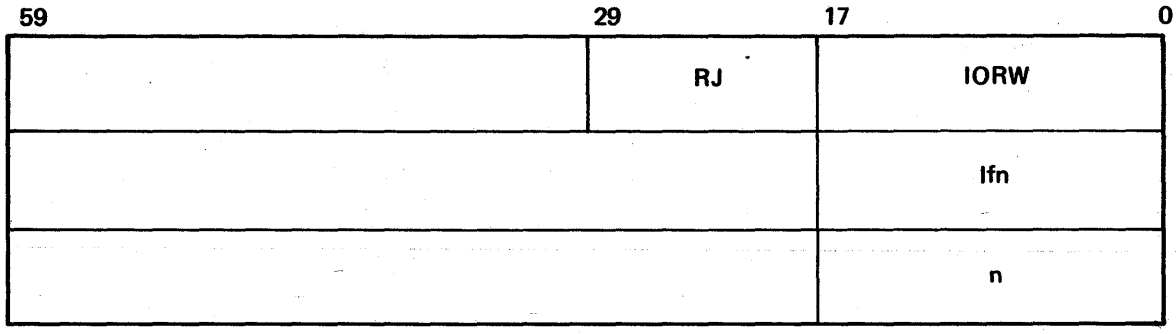
If the buffer contains at least one PRU of data. WRITE is called.

When a working storage area does not exist for an indexed file, or the length of the area is 0, the same procedures occur with the omission of any transfer of data to the buffer.

The code generated by the WRITOUT macro depends on the parameter list. For no second parameter, a name parameter, or number parameter, respectively, the code is:







REWRITE MACROS

The functions REWRITE, REWRITER, and REWRITEF update records in existing mass storage files. A fourth rewrite function, WRITIN, can be used similarly to WRITOUT; it can be used in conjunction with REWRITE, as the WRITOUT function with WRITE, and the REWRITER function should be used to terminate the record rewritten. These functions do not change the total amount of mass storage assigned to the file, nor do they update any index which may be associated with the file.

All of these functions call for writing in place, not writing at end-of-information. Since the system cannot determine the length of the original record, it offers no protection from over-writing or under-writing and does not issue diagnostics when these conditions occur. The system guarantees only that a rewritten record does not extend beyond the file end-of-information, with writing taking place up to that point and a diagnostic issued if the program attempts to go beyond that point. End-of-information is never moved. The index record existing at the end of random file is not protected.

Rewrite functions are similar to WRITE, WRITER, and WRITEF. Parameters for the macros are the same.

REWRITE lfn,recall

REWRITER lfn,lev,recall

REWRITEF lfn,recall

The user is responsible for knowing file structure before and after the rewrite. A minimum of one PRU is transferred from the circular buffer to the file each time a rewrite function is issued. Writing always begins at the current file position. Therefore the user must see that the file is positioned properly before writing takes place.

The amount of information rewritten for each call depends on the amount of information in the circular buffer, with the minimum amount being one PRU which may include a short or zero-length PRU. When a system-logical-record is to be replaced with a record of the same length in a single rewirte operation, RE-WRITER should be used. A longer record may require REWRITE and REWRITER, depending on the buffer size.

When the new record is not the size of the original record, the resulting file may have spurious records. Short replacement records, where the original record was contained in a single PRU, or the replacement record extends into the last PRU of the original record, do not cause difficulties. When the new record occupies fewer PRUs than the original, however, the end of the original record remains in the file. As an example consider an original 120-word record occupying a full PRU of 64 words and extending 56 words into a second PRU. Replacing the record with 60 words produces a short PRU in place of 64 words or original data. The 56 words of the second PRU of the original record remain in the file, since mass storage allocation never is changed by a rewrite.

A similar condition is created when the replacement record extends beyond the PRU's of the original record. Since the beginning of the next record in the file is overwritten, its usefulness is destroyed, but the remainder of the record still resides in the file.

When REWRITEF is issued, a zero-length PRU containing a level 17 is written. If issued when the file is positioned at any point, two level 17 indicators will exist on the file.

When random files are being rewritten, the methods of writing and the results of under-writing or over-writing a logical record are the same as for sequential files. Index integrity can be destroyed by rewriting records of different lengths. The user must position the file properly before each record is rewritten. Otherwise, writing takes place at the current position. Subsequent rewriting operations rewrites the next record in the file, which is not necessarily the next index entry for the file.

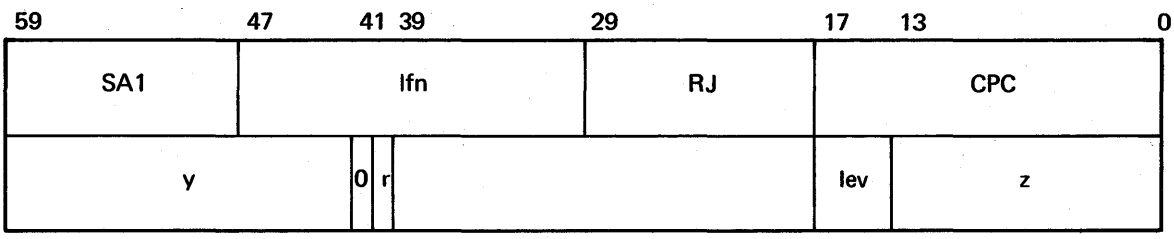
To position a random file for rewriting, the user may use one of two methods:

Set up the FET the same as for a random read and insert the record address found by searching the file index into the record request/return field in the seventh word of the FET.

For an indexed file with records identified by name or number, use the WRITIN function, which causes the system to search the user's index and set the necessary FET fields.

Once the file is positioned to the beginning of a record, a REWRITE and REWRITER sequence or a WRITIN and REWRITER sequence can be executed without further repositioning. The record request/return field in the FET will be cleared by the first REWRITE or REWRITER that is issued by the calling program or WRITIN, and remain cleared until repositioning for another record is required.

The rewrite functions generate the following code:



The y and z fields depend on the specific function. The value of y is:

002 for REWRITE
003 for REWRITER or REWRITEF

The value of z is:

214 for REWRITE
224 for REWRITER
234 for REWRITEF

WRITIN MACRO

The WRITIN function applicable to mass storage files is a rewrite-in-place function similar to the rewrites. It assumes the user has full knowledge of file structure and knows the results of his actions, as explained with the rewrite functions.

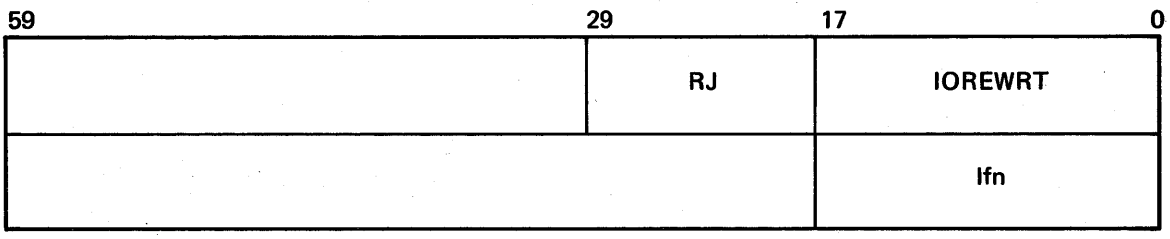
WRITIN is similar to WRITOUT in that it relieves the user of the responsibility of manipulating buffer pointers when a working storage area is provided. When the circular buffer has been filled from the working storage area, WRITIN issues a REWRITE. Handling of binary and coded data is the same as for a WRITOUT. Parameters for WRITIN, and results of its use, are the same as for WRITOUT.

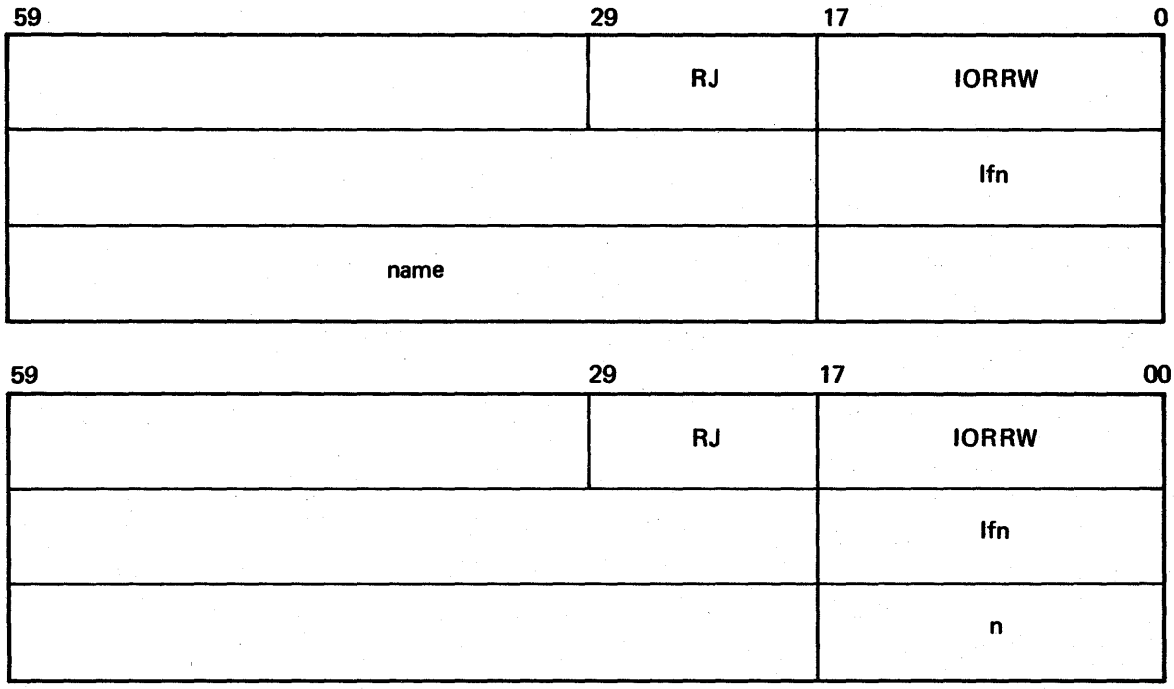
```
WRITIN  lfn

WRITIN  lfn,/name/

WRITIN  lfn,n
```

REWRITER is required to terminate a record, except when WRITIN or WRITOUT names another indexed record. In this case a REWRITER of level 0 is forced before the new record is begun.

If a working storage area does not exist when WRITIN is issued to a random or sequential file, the function is ignored with no error indication. For an indexed file without a working storage area, however, a WRITIN specifying a record name or number causes file repositioning to the beginning of that record. Therefore, the WRITIN function is useful before REWRITE or REWRITER.

The code generated by the WRITIN macro depends on the second parameter. For no parameter, a name parameter, and a number parameter, respectively, the code is:





POSITIONING FUNCTIONS

Files can be repositioned forward with the SKIPF function, or repositioned in a reverse direction with BKSP, BKSPRU, REWIND, SKIPB, and UNLOAD. Any of these commands can be issued at any point in a logical record. If parity errors occur during repositioning, they are ignored.

SKIPF	Skips records forward
SKIPB	Skips records backward
BKSP	Skips back single record
BKSPRU	Skips back single physical record unit
REWIND	Skips back to beginning-of-information
UNLOAD	Skips back to beginning-of-information and unloads

Reverse functions other than REWIND stop at the beginning of the current volume of magnetic tape. No status returned to the FET indicates that beginning-of-volume has been detected before the requested number of backspaces was completed. However, if the XP bit (bit 40 of word 2) is set, the number of skips yet to be made will be stored in the RSC field (bits 24-41) of the FET extension.

If a magnetic file is positioned immediately after a newly written record when a reverse motion function is issued, trailer label procedures are executed before the function is performed. Four tape marks are written if a trailer label format is not defined.

SKIPF MACRO

SKIPF causes one or more system-logical-records, or physical records of an S or L tape, to be bypassed in a forward direction.

```
SKIPF  lfn,n,lev,recall
```

The number of records or record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. The maximum octal value of n is 777776. If n is 777777 and the file is on magnetic tape, it is not repositioned. If n is 777777 and the file is on mass storage, it is positioned at end-of-information. If the CIO call is used instead of the CPC call, whenever n=0 it is treated as if n=1 was given.

For mass storage and SI tapes, the skip count is incremented as each level defined by the lev parameter is passed. Thus, a SKIPF with a count of 1 and lev of 0 issued in the middle of a record positions the file to the beginning of the following record.

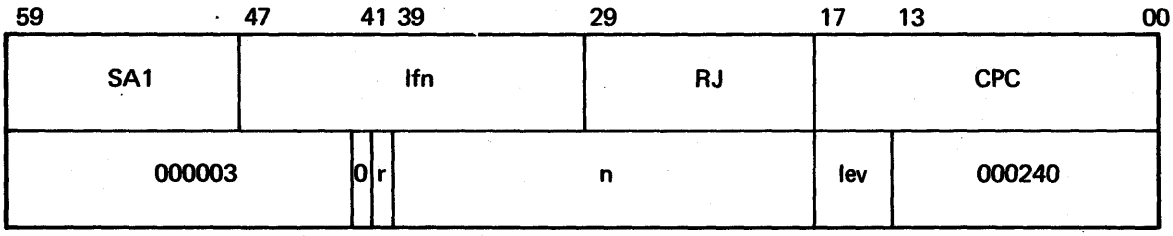
The lev parameter specifies the level defining the record end; logical records are skipped until an end-of-logical record with a level number greater than or equal to the requested level is reached. The file is positioned immediately following this end-of-logical-record mark.

If lev is absent, this field is set to zero, and the file is positioned forward n logical records or parts of records. If end-of-information is encountered before an end-of-logical-record with the specified level is found, the end-of-information status bit will be set in the FET.

Although level numbers do not exist on S and L data format tapes, an lev parameter may be specified for SKIPF requests. If level number 17 is specified, a skip to end-of-partition is performed. Any other level number is assumed to be zero, and one record is skipped.

A SKIPF is continued across volumes when the user processing (UP) bit is 0. If UP is set, the forward skip stops when end-of-volume is detected. If both UP and XP are set when end-of-volume appears before the skip count is fulfilled, the difference between the count requested and count made to that point will be returned to the RSC field in the FET extension.

The SKIPF macro generates the following code:



SKIPB MACRO

SKIPB causes one or more system-logical-records, or physical records of S and L tapes, to be bypassed in a reverse direction.

```
SKIPB  lfn,n,lev,recall
```

The number of records or logical record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. When n is the maximum value of 777777 (octal), the file is rewound.

For mass storage and SI tapes, if the level parameter is used, logical records are read backwards until a short PRU containing the specified level has been read. A forward read is issued, leaving the file positioned after this short PRU. If the file is positioned initially between logical records, the level number immediately preceding the current position is ignored in searching for a record of the specified level. This positioning process is performed n times.

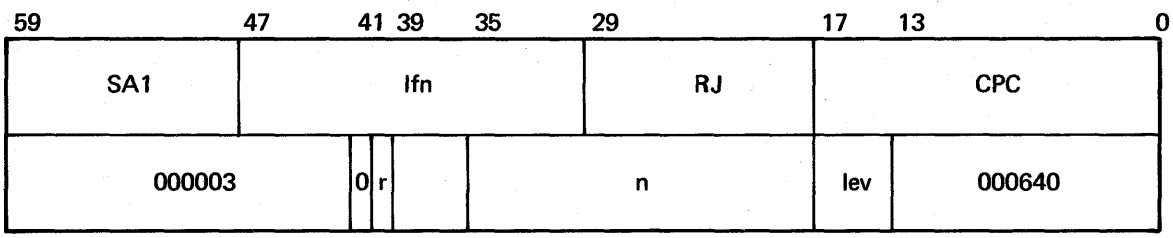
Consecutive system-logical-records within a file may be organized into a group by using level number. The file is composed of one or more groups of logical records. This may be done by choosing a minimum level number other than 0, assigning a larger or equal level number to the last logical record of each group, and assigning a smaller level number to all other logical records. Then SKIPB lfn,,lev skips the file backward to the beginning of the logical record group which immediately follows a logical record of level lev.

If the level parameter is absent, this field is set to zero, and the file is positioned backward n logical records (or partial logical records if the SKIPB is issued in the middle of a logical record).

If the beginning of a volume is encountered on mass storage and the UP bit is set, or if the beginning of a volume on magnetic tape is encountered before the requested level number is found, the request terminates with no indication. However, if XP is set, field RSC in the FET extension contains the count n still required to complete the operation. Parity errors encountered during a SKIPB operation are ignored.

For S and L tapes, only levels 0 and 17 are recognized; any other level specified is assumed 0.

The SKIPB macro generates the following code:

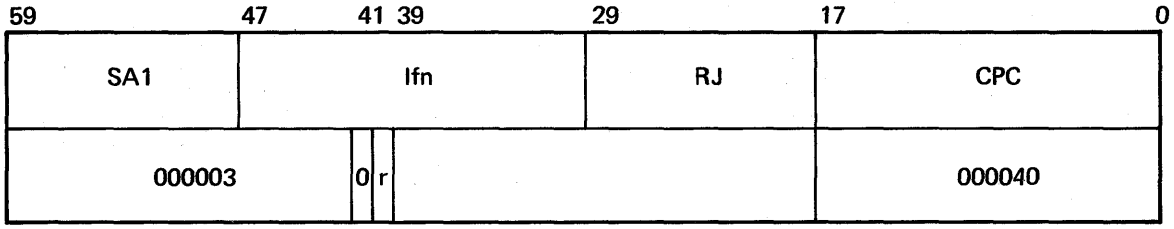


BKSP MACRO

BKSP causes one system-logical-record to be bypassed in a reverse direction. This function is a subset of SKIPB; it is included for compatibility with previous systems.

```
BKSP  lfn,recall
```

The BKSP macro generates the following code:



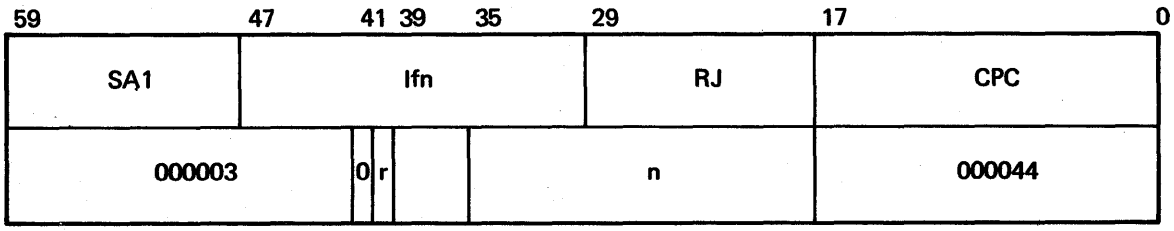
BKSPRU MACRO

BKSPRU causes one or more physical record units to be bypassed in a reverse direction.

```
BKSPRU  lfn,n,recall
```

The number of PRU's to be bypassed is indicated by n. If n does not appear, one PRU is skipped.

The BKSPRU macro generates the following code:



REWIND MACRO

REWIND positions a file to beginning-of-information. A REWIND issued for a file already rewound has no effect. A REWIND request for a file on a device that cannot be rewound causes a 22 status indicating an illegal function to be returned to the FET.

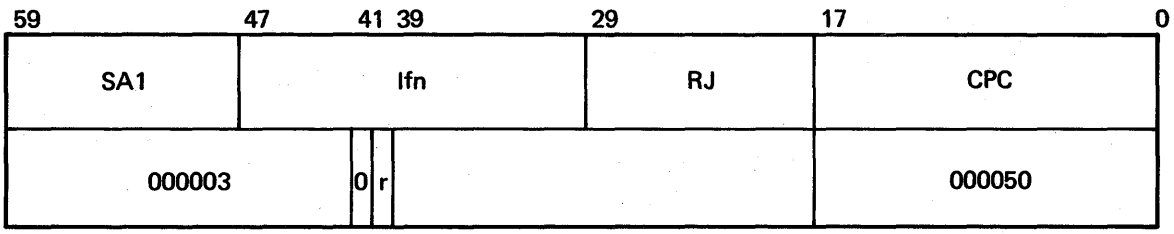
```
REWIND  lfn,recall
```

Labeled tapes are positioned to beginning-of-information ahead of the label group. Subsequent forward motion requests result in the label being skipped before the tape is read or written.

For unlabeled multi-volume tapes, a REWIND causes the current volume to be rewound. For labeled multi-volume, single-file tapes, a REWIND causes the current volume to be rewound and the volume number in the system tables to be set to 1. A subsequent forward motion causes the label to be read and compared with the system tables, and the operator is notified if the current volume is not number 1.

For multi-file labeled tapes, a REWIND issued for a file causes positioning to the beginning of that file. If necessary, the operator is instructed to mount the previous volume. A REWIND that references a multi-file name is illegal; the job terminates.

The REWIND macro generates the following code:

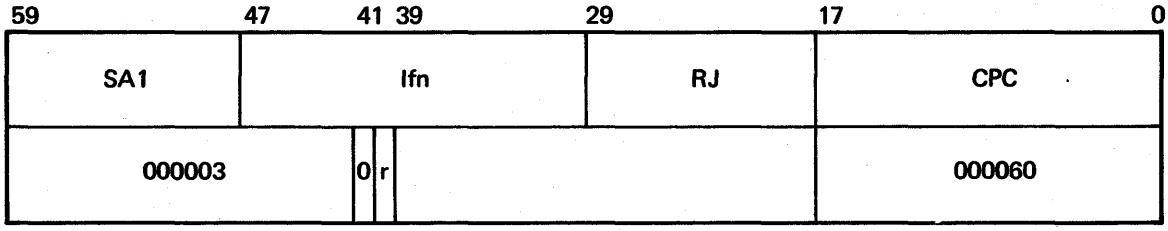


UNLOAD MACRO

UNLOAD operates in a manner similar to REWIND, except that it only affects the current volume of tape. UNLOAD cannot override an IU inhibit unload parameter on a REQUEST control statement. Otherwise, a tape file is rewound and unloaded.

UNLOAD lfn,recall

The UNLOAD macro generates the following code:



FILE DISPOSITION

Files can be disposed of in several ways in addition to the disposition associated with special file names.

The file can be destroyed by the EVICT function.

The file can be routed to an output device at the central site or a remote terminal station with the ROUTE or DISPOSE functions.

Files on public device sets that have not been named in a ROUTE or DISPOSE control statement or macro, or have not been equated to standard output file names such as OUTPUT or PUNCH, disappear upon job termination. Permanent files, of course, are retained under permanent file manager disposition.

It is not possible to dispose of a file by setting a disposition code directly in the FET.

EVICT MACRO

The EVICT function declares that contents of file lfn are to be discarded.

EVICT lfn,recall

When a file on a public device set is evicted, all space occupied by that file is released to the system. The space immediately becomes available for any system purpose or reassignment. An EVICT function directed to a permanent file is ignored; a dayfile message is issued and the job continues normally.

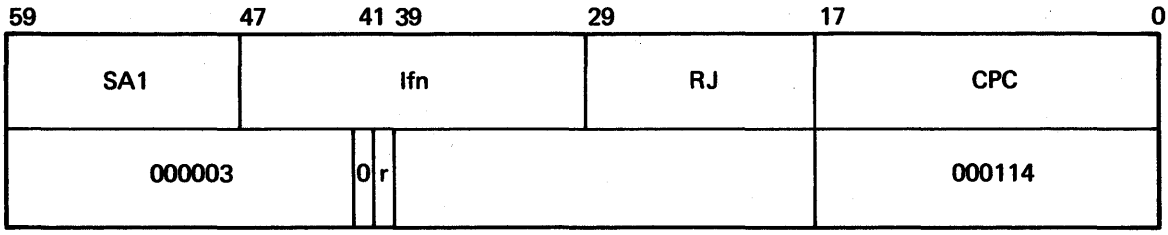
When a file on a magnetic tape is evicted, the tape is rewound and set to new status, thus declaring that the data and label are no longer valid and cannot be read by the job. If the file was declared to be labeled a new header label is written on any subsequent file reference. However, the evicted file is not overwritten without operator authorization if the file expiration date has not passed.

If an EVICT function is directed to a member of a multi-file set, the set already must have been positioned at that file. Eviction of a member file also implies eviction of all files occupying higher numbered positions.

The logical file name used in the EVICT function is retained and cannot be used for a file on another device.

EVICT is undefined and, therefore, illegal on unit record equipment. A fatal error results if it is tried.

The EVICT macro generates the following code:



DISPOSE MACRO

With the dispose function, a central processor program may declare a disposition code and initiate termination processing for a file. Files either can be released or sent to the output queue of completed files, as explained with the DISPOSE control statement. The dispose function can be used only for files that are resident on queue devices.

DISPOSE lfn,*x=ky,recall

lfn — Logical file name

* — Optional end-of-job disposition indicator

x — Two-character disposition mnemonic.

Mnemonic	Definition
PR	Print on any available printer
P2	Print on 512 printer
LR	Print on 580-12 printer
LS	Print on 580-16 printer
LT	Print on 580-20 printer
PB	Punch standard binary format
PU	Punch Hollerith format
P8	Punch free-form binary format
FR†	Print on microfilm recorder
FL†	Plot on microfilm recorder
PT†	Plot
HR†	Print on hardcopy device
HL†	Plot on hardcopy device

†Supporting drivers must be supplied by the installation.

k — Optional site indicator; y must follow

C	Central site
I	INTERCOM terminal

y — Qualifier to k; y cannot be used without k

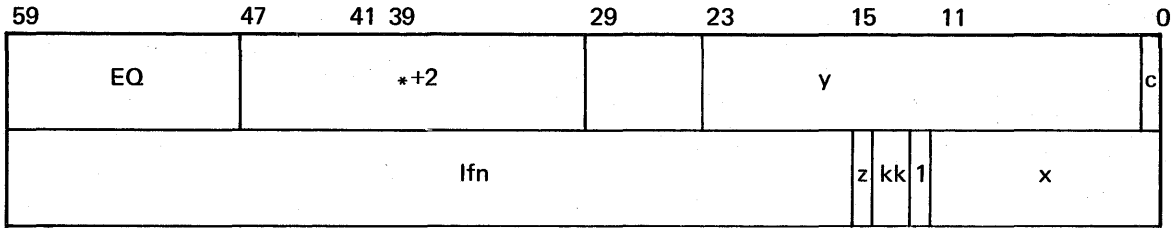
If k is C, two-character alphanumeric installation defined identifier of special forms or paper

If k is 1, two-character user identification

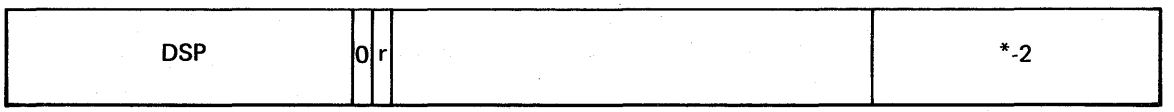
recall — Optional character indicating recall

If only lfn is given, the file is released, with mass storage and table references being removed.

The code generated by the DISPOSE macro is:



The following is set in X6 with a subsequent call to SYS=.



z — set to 1 when * is used

kk — site indicator

00	none
01	central site
10	INTERCOM terminal

The completion bit (C) is set to 1 by DSP when the requested function is complete.

ROUTE MACRO

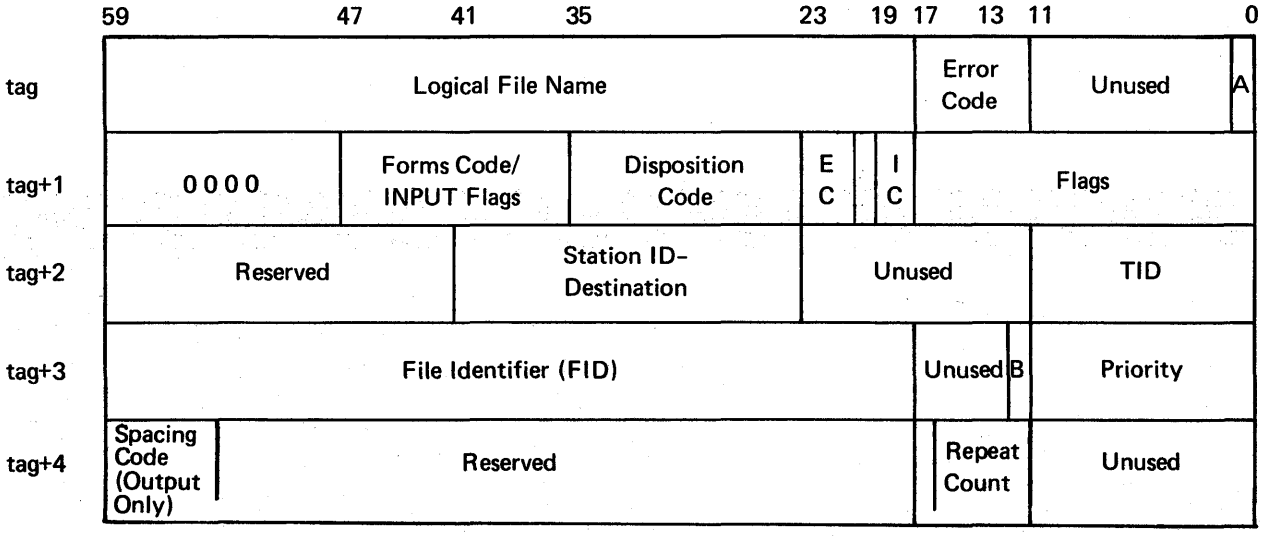
The ROUTE macro places a file in an input or output queue, evicts a file, or specifies attributes the file has when it is placed in an output queue. ROUTE has all the capabilities of DISPOSE. See the ROUTE control statement for a complete description of the ROUTE capabilities. The user must construct a parameter list in the format described below before calling the ROUTE macro. The file being processed must not be the INPUT file, but it must be resident on a queue device.

ROUTE tag,recall

tag Address of the ROUTE parameter list

recall Optional non-blank character indicating auto recall

Parameter List Format:



Word	Bits	Field	Description
tag	18-59	Logical File Name	lfn of file to be routed: must be mass storage file, not a permanent file, cannot reside on a dismountable device, must have at least read permission.
	12-17	Error Code	Code returned by system when bit 12 of Flag field is set, as noted below.
	1-11	Unused	
	0	A	Complete bit. Must be zero when macro is issued; system sets to one when function is complete.
tag+1	48-59	Zeros	Twelve bits of zero. Allows compatibility with previous callers of DSP. The old calling sequence put the lfn in tag+1.
	36-47	Forms Code/ Input Flags	Two display code letters or digits identifying forms to be used for this file. Default is standard forms. If the file is to be routed to an input queue, this field is defined as:

Bit	Definition
47	Unused
46	Unused
45	Do not catalog INPUT file

Word	Bits	Field	Description
tag+1			**Bit** **Definition**
			44 Reserved for use by system jobs
			43 Send file to input queue even if job statement error
			42 Use dependency count
			36-41 Dependency count
	24-35	Disposition Code	Disposition Code mnemonic in display code.
	21-23	EC	External characteristics code translated as follows:

Value (octal)	Print File	Punch File
0	EC (default)	EC (default)
1	- -	EC=SB
2	- -	EC=80COL
3	EC=B4	- -
4	EC=B6	EC=026
5	EC=A6	EC=029
6	EC=A9	EC=ASCII
7	Reserved for installations	

	20	Unused	
	18-19	IC	Internal characteristic code translated as follows:

Value (octal)	Meaning
0	IC or IC=DIS – Display Code (default)
1	IC=ASCII
2	IC=BIN binary
3	Reserved

	0-17	Flag Bits	Indicate specified parameters:

Bit	Meaning
17	File name assigned by system is returned at tag word bit 18-59
16	Unused
15	Spacing code
14	Repeat count
13	Reserved for system job
12	No dayfile message; return error code in bits 12-17 of first-word of parameter list
11	Reserved for system jobs
10	Forms code
9	Priority

Word	Bits	Field	Description
tag+1			**Bit** **Meaning** 8 Internal characteristics 7 External characteristics 6 FID=* System appends two unique sequence characters to file identifier 5 FID – System uses FID specified in tag+3, bits 18-59 4 Disposition code 3 Route to remote station 2 TID is specified. 1 Route to central site. 0 End-of-job (deferred ROUTE)
tag+2	42-59	Reserved	Used by system jobs; otherwise, set to binary zero.
	24-41	Station ID-Destination	Display code destination ID. The file is processed by the system with this logical identifier.
	12-23	Unused	
	0-11	TID	Display code identifier of INTERCOM terminal to receive the file.
tag+3	18-59	FID	If the calling job was not loaded completely from the system library, only a maximum of 5 characters may be used to specify FID. The additional 2-character sequence number is determined by flag bits 5 and 6. Seven characters may be specified by calling jobs loaded completely from the system library.
	13-17	Unused	
	12	B	Use the priority in bits 0-11.
	0-11	Priority	Priority for an interactively routed output file being routed to the routing terminal.
tag+4	18-59	Reserved	For use by system jobs only.
	54-59	Spacing Code (SC)	580 PFC Printer. Spacing array to be loaded with the file (output only).
	17	Unused	
	12-16	Repeat count	Repeat count.
	0-11	Unused	

When an error occurs in processing a ROUTE macro, either a dayfile message explaining the error is issued, or an error code is returned in bits 12-17 of the first word in the parameter list. If bit 12 of the flag field is set, an error code is returned and no dayfile message is issued. If bit 12 is not set, a dayfile message is issued and no error code is returned. If the address of the parameter list is outside the field length of the job or if the complete bit is set when the macro is issued, the job aborts. For all other errors, the ROUTE macro is not executed but processing continues.

When a diagnostic is issued for the ROUTE macro, the message ERROR IN ROUTE FUNCTION LFN= is issued before the message describing the error. If the function completes successfully, no message is issued; the error code field is set to binary zero.

Error Code (octal)	Message
01	INVALID LFN – DSP
02	CANT ROUTE NON ALLOCATABLE EQP
03	CANT ROUTE PERM FILE
04	NO PERMISSION TO ROUTE THIS FILE
05	ROUTE TO INPUT NOT IMMEDIATE – IGNORED
06	IMMEDIATE ROUTING – NO FILE – IGNORED
07	INVALID DISPOSITION CODE – ROUTING IGNORED
10	INVALID FID – ROUTING IGNORED
11	DSP ABORTED BY SYSTEM
12	DSP PARAMETER OUTSIDE FL
13	PRIORITY SPECIFICATION IGNORED
15	EI200 SPECIFIED – INTERCOM USED (DSP)
16	CAN NOT ROUTE INPUT FILE
17	DSP COMPLETE BIT ALREADY SET
20	FILE ON DISMOUNTABLE DEVICE – ROUTING IGNORED
21	TID NOT ALPHANUMERIC – ROUTING IGNORED
22	FORMS CODE NOT ALPHANUMERIC – ROUTING IGNORED
23	INVALID LINK TYPE – ROUTING IGNORED (DSP)
24	FILE NOT ON QUEUE DEVICE – ROUTE IGNORED
25	PRE-DAYFILE LFN AND NO DC=IN – ROUTE IGNORED
26	PRE-DAYFILE FILE NOT FOUND – ROUTE IGNORED

See the Diagnostic Handbook for a description of each message.

PERMANENT FILE FUNCTIONS

Permanent file functions are those defined by control statements with the following names:

ALTER	EXTEND
ATTACH	PURGE
CATALOG	RENAME

Information applicable to a control statement call is also applicable to a call through a permanent file macro. In addition, an FDB and PERM macro are available.

Each permanent file macro expansion contains an RA+1 call to a permanent file program. Parameters necessary for execution of a function are contained in the file definition block (FDB) table within the user's field length.

FDB MACRO

The macro for generating an FDB has the format:

```
fdbaddr    FDB  lfn,pfn,parameter list
```

fdbaddr is the symbol to be associated with the word in the FDB that contains the lfn; it must be present in the location field. Parameters are separated by commas and terminated by a blank. They may include any of those indicated by the two-letter codes described for control statements.

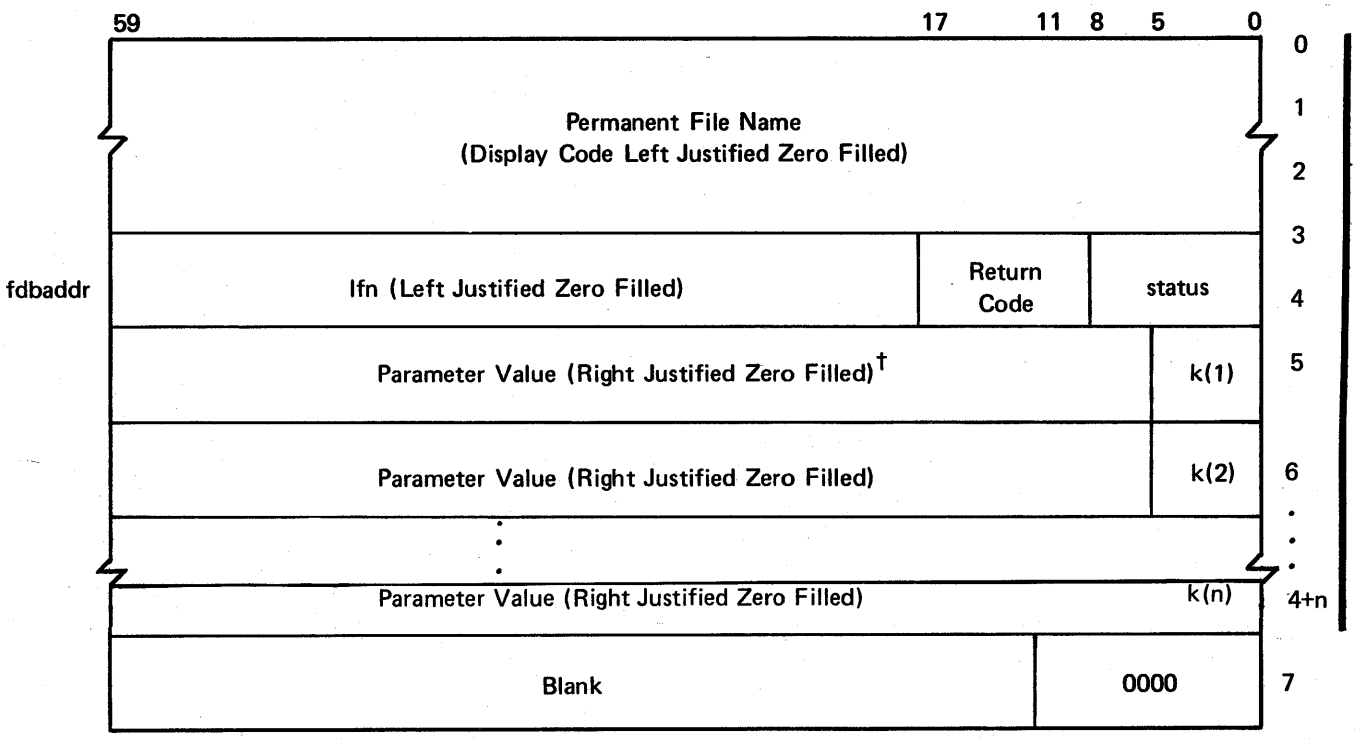
The field to the right of the macro name. FDB, is identical to that which could be on a control statement. Parameters are entered into the FDB as they are encountered in the list. The FDB is generated in-line during assembly whenever the macro is called.

A user specifies the intent of a particular function by specifying parameters. If they do not clearly define the function request, the permanent file manager attempts to inform the user of the unknown information by the following means:

Modification of the File Definition Block will be done when an illegal parameter is correctable. For example, if an incorrect cycle number is encountered on a CATALOG function, the actual cycle number is returned in the FDB. If code is not successful, error codes may be returned in the FDB.

A job dayfile message is issued to the job dayfile unless the RT or RC parameter is specified in the function call that specifies the FDB.

The FDB generated by the macro has the form:



†The SN parameter value is left justified, zero-filled

Field	Description
k(n)	Parameter identifier in octal or display code

Value (Octal)	Keyword Parameter	Description
00		End of FDB list
02	RP	Retention period days
03	CY	Cycle number
04	TK	Turnkey password (display code)
05	DN	Control password (display code)
06	MD	Modify password (display code)
07	EX	Extend password (display code)
10	RD	Read password (display code)
11	MR	Multi-read parameter
13	XR	Control, Modify, Extend password (display code)
14	ID	Owner identification (display code)
16	AC	Account (display code)
17	EC	ECS Buffering (display code)
20 24	PW	Password submitted (display code)
25	FO	File organization (display code)
31	LC	Lowest cycle

Field **Description**

Value (Octal)	Keyword Parameter	Description
32	ST	Station ID (display code)
33	RW	Multi-read with single rewrite
40	SN	Setname (display code, left justified)
41		Reserved for VSN parameter
43	RB	PURGE RB conflict parameter

Status — Status bits

Bit	Meaning
0	Complete bit
1	Unused
2-5	Function code bits 0010 ATTACH 1010 RENAME 0100 CATALOG 1100 PERM 0110 EXTEND 0111 ALTER 1000 PURGE
6	Set if RC or RT not specified. Issue dayfile messages. All errors are fatal.
7	Set if RT specified
8	Set if NR specified

Return Code — Return codes

Value (Octal)	Meaning
000	Function Successful
001	PFN/ID error
002	lfn already in use
003	Unknown lfn
004	No room for extra cycle (limit is five)
005	Permanent file catalog full
006	No lfn or pfn
010	Latest index not written
011	File not on PF device
012	File not cataloged, SN=xxxxxxx (xxxxxxx is the set name of the device set searched)
013	Archive retrieval aborted
015	Cycle number limit reached
016	Permanent File Directory full
017	Function attempted on non-permanent file
020	Function attempted on non-local-file
022	File never assigned to a device
023	Cycle incomplete
024	PF already attached

Field	Description	
	Value (Octal)	Meaning
	025	File unavailable
	027	Illegal lfn
	030	File dumped
	031	Illegal function code
	033	ALTER needs exclusive access
	035	File already in system
	040	Illegal setname specified
	041	Device set not mounted at control point
	042	RBT chain too large for PFC
	043	File resides on unavailable device
	070	PFM stopped by system
	071	Incorrect permission
	072	File Definition Block address invalid (not returned to FDB)
	073	I/O error on PFD/PFC rewrite
	074	ST parameter illegal with private device set

On control statement requests, all errors are fatal; on macro requests, unless the RC parameter is specified, the job is terminated. Any job that attempts a privacy breach is terminated. Codes greater than 70 octal are fatal. All internal permanent file malfunctions are system errors that cause job termination.

FUNCTION MACROS

The macro function call is of the following form:

function fdbaddr,RC,RT,NR

function — Any permanent file function, such as CATALOG.

fdbaddr — Symbol on FDB macro.

RC — Optional parameter that causes return codes to be available in FDB.

RT — Optional parameter that inhibits any permanent file queueing.

NR — Optional parameter that inhibits auto recall.

All permanent file macro calls are issued with auto recall unless NR is present. In this case, it is possible for the central processor program to test the completion bit in the FDB to determine whether the function has completed.

The RT parameter can be used only at the macro level, unless the user constructs the FDB himself. Jobs attempting permanent file attaches queue for the requested file if the RT parameter is not specified when:

File is unavailable (job requesting file wants exclusive access, or job using the file has exclusive access).

Attached permanent file table is full.

Archived file (a permanent file stored on tape rather than mass storage) is temporarily unavailable. ATTACH will set up a LOADPF job to be scheduled. The job requesting the attach will be swapped out until the file is available.

When EXTEND is used on an indexed file, the current index must be rewritten, at the end of the file, by the user to invalidate any prior index. This must be done prior to the EXTEND function. When an EXTEND function is requested for a random file, nothing must have been written on the file since the index was last written.

PERM MACRO

PERM, available only as a system macro, allows a running program to determine what permissions have been granted to a file and whether or not the file is permanent. The macro format is:

```
PERM    fdbaddr,RC,NR
```

The only required parameter is the lfn, which should be supplied in the FDB. This lfn should reference a file available to the job calling PERM.

The 5-bit code is returned to the user as the return code in the FDB (bits 9-13 in fdbaddr). The rightmost 4 bits are the permission bits. The octal codes are:

10 CONTROL
04 MODIFY
02 EXTEND
01 READ

The leftmost bit of the 5-bit code is a flag. If it is 0, the lfn references an attached permanent file unless the entire 5-bit code is equal to 0. If the entire code is 0, the lfn is unknown to the job calling PERM since a permanent file cannot be attached without permissions. A 1 in the leftmost bit indicates a local file.

SYSTEM TEXTS

System texts provide commonly used macro, micro, and symbol definitions for use in COMPASS source programs. NOS/BE 1 provides several system text overlays, which are loaded by COMPASS from the system libraries when specified by S or G parameters on the COMPASS control statement. S or G parameters can also be used on FTN control statements when FORTRAN Extended source programs contain intermixed COMPASS subprograms. Up to seven system texts can be specified, each by a different S or G parameter, for a given assembler run. Most system texts are made up of UPDATE common decks described below. System texts are constructed as part of the installation process described in the Installation Handbook.

COMMON DECKS

ACTCOM–CPU System Action Request Macros:

IXi Xj/Xk	CONTRLC	GETJCI	RECALL	STATUS
IXi Xj/Xk,Bn	DATE	IOTIME	RECOVR	SYSCOM
ABORT	DISPOSE	JDATE	REQUEST	SYSTEM
ACQUIRE	ENDRUN	LOADREQ	ROUTE	TIME
CHECKPT	FILESTAT	MEMORY	RTIME	TRANSF
CLOCK	FILINFO	MESSAGE	SETJCI	VERIFYJ

CIOCOM - Codes, symbols, macros and installation parameters associated with magnetic tape processing and tape scheduling.

CMRDEF - Symbols, macros and installation parameters for Monitor and the integrated scheduler.

COMACIO - CPU Input/Output Macros.

COMAFET - File Environment Table Generation Macros:

FILEB	LABEL	RFILEB
FILEC		RFILEC

COMAREG - Replacement for R= pseudo-instruction

COMSRAS - System Communication Symbols:

Contains definitions of the system communications symbols described in this section under the heading SYSTEM MACRO.

CPSYS - Input/Output Macros using CPC:

BKSP	READC	REWRITEF	WRITE
BKSPRU	READIN	REWRITER	WRITEC
CLOSE	READN	RPHR	WRITEN
CLOSER	READNS	SKIPB	WRITEF
EVICT	READSKP	SKIPF	WRITER
OPEN	REWIND	UNLOAD	WRITIN
POSMF	REWRITE	WPHR	WRITOUT
READ			

ECSCOM - ECS and ECS Link installation parameters; ECS flag register function macros.

ECSDEF - Codes, macros, symbol definitions and storage descriptors for ECS processing and the ECS Link.

IPARAMS - Installation Parameters:

Contains installation parameters as symbol and micro definitions.

LMACOM - Loader Request Macros:

Contains two macros: LOADER and LDREQ.

PFCOM - Permanent File Macros:

ALTER	EXTEND	PURGE	GETPF
ATTACH	FDB	RENAME	SAVEPF
CATALOG	PERM		

PPSYS - Peripheral Processor System Definitions:

Contains many system symbols and micros, and the following macros:

ADK	CRI	LDK
BIT	ENM	PPENTRY
CEQU	JOBCARD	SBK
CMICRO	LDCA	UJK

SCHCOM - Integrated Scheduler Macros:

C1SO	SCHLOK	SCHSTOR
ENTRY34	SCHSAVE	STREQ
LDW		

SISICOM - SCOPE Indexed Sequential Macros:

ACCESSK	OPENOLD	SETBLKI
ACCESSN	REPLACE	SETCOLL
DELETE	REPOS	SETERR
FORCEW	SEEKL	SETFET
INSERT	SEEKS	SETKEY
OPENNEW	SETBLKD	TERMNAT

STATCOM - Station Interface Definitions:

Contains definition of interface to the station control point and definition of codes used in message requests.

6RMCOM - CYBER Record Manager Definitions:

Contains macro, micro, and symbol definitions for user programs that use CYBER Record Manager.

SSYS - System Control Point macros and definitions.

TEXT OVERLAYS

The system text overlays contain various combinations of the common decks, as shown below. When the multimainframe module is present and IP.SRMS=1, an additional system text, SRMSTXT, is cataloged.

CMRTEXT	System text for assembling Central Memory Resident segments separately from CMR. Common decks IPARAMS, SSYS, ECS COM, CIOCOM, CMRDEF, and ECS DEF.
CPCTEXT	System text for central processor programs using CPC. Common decks ACTCOM, COMAFET, COMSRAS, CPSYS, and SISICOM.
CPUTEXT	System text containing all system macros, micros, and symbols for COMPASS CPU programs that use the CIO= communication routine for I/O and run under the CYBER operating system. Common decks ACTCOM, COMACIO, COMAFET, COMAREG, and COMSRAS.
IOTEXT	System text for central processor programs using CYBER Record Manager. Common decks ACTCOM, COMSRAS, and 6RMCOM.
IPTEXT	Installation parameter system text. Contains a single macro, IPARAMS, whose body is the IPARAMS common deck.
LDRTEXT	System text for central processor programs using CYBER Loader. Common deck LMACOM.
PFMTEXT	System text for central processor programs using permanent files. Common deck PFCOM.
PPTEXT	System text for peripheral processor programs. Common decks COMSRAS and PPSYS.
SCHTEXT	System text for central and peripheral processor programs interfacing with the integrated scheduler. Common deck SCHCOM.
SCPTEXT	System text for central and peripheral processor programs in the operating system. Common decks ACTCOM, COMAFET, COMSRAS, CPSYS, and PPSYS.
SDDTEXT	System text containing two macros, PPUDMP and C1DD, that provide the interface between PP programs and the dynamic dump feature.
SSYTEXT	System text for System Control Point subsystem programs. Common deck SSYS.
STATEXT	System text of station interface definition for DSD and INTERCOM. Common deck STATCOM.
SYSTEXT	System text for central processor programs. This is the default system text used by COMPASS when no S or G parameters are specified. It can be identical to either CPCTEXT or IOTEXT, at installation option. In the released system, SYSTEXT is equal to IOTEXT.
SRMSTXT	Cataloged as an additional system text when multi-mainframe shared RMS is installed.

The following additional system texts are provided by product set members.

ALGTEXT	Contains COMPASS coded macros used to expand application areas of ALGOL.
FTNMAC	Contains macros used by COMPASS object programs produced by the FORTRAN Extended compiler (FTN).
SMTEXT	Contains macros for central processor programs that call Sort/Merge.
RMERTXT	Contains CYBER Record Manager error dictionary.

STANDARD CHARACTER SETS A

CONTROL DATA operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE 1, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

STANDARD CHARACTER SETS

CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code	CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code
:†	:	00††	8-2	00	8-2	072	6	6	41	6	06	6	066
A	A	01	12-1	61	12-1	101	7	7	42	7	07	7	067
B	B	02	12-2	62	12-2	102	8	8	43	8	10	8	070
C	C	03	12-3	63	12-3	103	9	9	44	9	11	9	071
D	D	04	12-4	64	12-4	104	+	+	45	12	60	12-8-6	053
E	E	05	12-5	65	12-5	105	-	-	46	11	40	11	055
F	F	06	12-6	66	12-6	106	*	*	47	11-8-4	54	11-8-4	052
G	G	07	12-7	67	12-7	107	/	/	50	0-1	21	0-1	057
H	H	10	12-8	70	12-8	110	(	(	51	0-8-4	34	12-8-5	050
I	I	11	12-9	71	12-9	111	)	)	52	12-8-4	74	11-8-5	051
J	J	12	11-1	41	11-1	112	$	$	53	11-8-3	53	11-8-3	044
K	K	13	11-2	42	11-2	113	=	=	54	8-3	13	8-6	075
L	L	14	11-3	43	11-3	114	blank	blank	55	no punch	20	no punch	040
M	M	15	11-4	44	11-4	115	, (comma)	, (comma)	56	0-8-3	33	0-8-3	054
N	N	16	11-5	45	11-5	116	. (period)	. (period)	57	12-8-3	73	12-8-3	056
O	O	17	11-6	46	11-6	117	≡	#	60	0-8-6	36	8-3	043
P	P	20	11-7	47	11-7	120	[	[	61	8-7	17	12-8-2	133
Q	Q	21	11-8	50	11-8	121	]	]	62	0-8-2	32	11-8-2	135
R	R	22	11-9	51	11-9	122	%	%	63††	8-6	16	0-8-4	045
S	S	23	0-2	22	0-2	123	≠	" (quote)	64	8-4	14	8-7	042
T	T	24	0-3	23	0-3	124	→	_ (underline)	65	0-8-5	35	0-8-5	137
U	U	25	0-4	24	0-4	125	∨	!	66	11-0 or 11-8-2†††	52	12-8-7 or 11-0†††	041
V	V	26	0-5	25	0-5	126							
W	W	27	0-6	26	0-6	127	∧	&	67	0-8-7	37	12	046
X	X	30	0-7	27	0-7	130	↑	' (apostrophe)	70	11-8-5	55	8-5	047
Y	Y	31	0-8	30	0-8	131	↓	?	71	11-8-6	56	0-8-7	077
Z	Z	32	0-9	31	0-9	132	<	<	72	12-0 or 12-8-2†††	72	12-8-4 or 12-0†††	074
0	0	33	0	12	0	060							
1	1	34	1	01	1	061	>	>	73	11-8-7	57	0-8-6	076
2	2	35	2	02	2	062	≤	@	74	8-5	15	8-4	100
3	3	36	3	03	3	063	≥	\	75	12-8-5	75	0-8-2	134
4	4	37	4	04	4	064	¬	^ (circumflex)	76	12-8-6	76	11-8-7	136
5	5	40	5	05	5	065	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	073

†Twelve or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

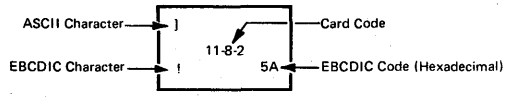
††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations from ASCII/EBCDIC % yield a blank (55_8).

†††The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

	b_8 b_7 b_6 b_5	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
b_4 b_3 b_2 b_1	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 I (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

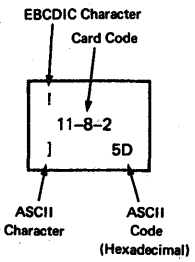
LEGEND



EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC) WITH PUNCHED CARD CODES AND ASCII TRANSLATION

BITS 0 1 2 3	BITS 4 5 6 7 / 1ST HEX 2ND	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
		0	1	2	3	4	5	6	7	8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	DS 11-0-9-8-1 80	12-11-0-9-8-1 90	SP no punch SP 20	& 12 & 26	- 11 - 2D	12-11-0 BA	12-0-8-1 C3	12-11-8-1 CA	11-0-8-1 D1	12-11-0-8-1 D8	{ 12-0 { 7B	} 11-0 } 7D	` 0-8-2 \ 5C	0 0 0 30
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	SOS 0-9-1 81	9-1 91	12-0-9-1 A0	12-11-9-1 A9	/ 0-1 / 2F	12-11-0-9-1 BB	a 12-0-1 a 61	j 12-11-1 j 6A	~ 11-0-1 ~ 7E	12-11-0-1 D9	A 12-1 A 41	J 11-1 J 4A	11-0-9-1 9F	1 1 1 31
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	FS 0-9-2 82	SYN 9-2 SYN 16	12-0-9-2 A1	12-11-9-2 AA	11-0-9-2 B2	12-11-0-9-2 BC	b 12-0-2 b 62	k 12-11-2 k 6B	s 11-0-2 s 73	12-11-0-2 DA	B 12-2 B 42	K 11-2 K 4B	S 0-2 S 53	2 2 2 32
0 0 1 1	3	ETX 12-9-3 ETX 03	TM 11-9-3 DC3 13	0-9-3 83	9-3 93	12-0-9-3 A2	12-11-9-3 AB	11-0-9-3 B3	12-11-0-9-3 BD	c 12-0-3 c 63	l 12-11-3 l 6C	t 11-0-3 t 74	12-11-0-3 DB	C 12-3 C 43	L 11-3 L 4C	T 0-3 T 54	3 3 3 33
0 1 0 0	4	PF 12-9-4 9C	RES 11-9-4 9D	BYP 0-9-4 84	PN 9-4 94	12-0-9-4 A3	12-11-9-4 AC	11-0-9-4 B4	12-11-0-9-4 BE	d 12-0-4 d 64	m 12-11-4 m 6D	u 11-0-4 u 75	12-11-0-4 DC	D 12-4 D 44	M 11-4 M 4D	U 0-4 U 55	4 4 4 34
0 1 0 1	5	HT 12-9-5 HT 09	NL 11-9-5 85	LF 0-9-5 LF 0A	RS 9-5 95	12-0-9-5 A4	12-11-9-5 AD	11-0-9-5 B5	12-11-0-9-5 BF	e 12-0-5 e 65	n 12-11-5 n 6E	v 11-0-5 v 76	12-11-0-5 DD	E 12-5 E 45	N 11-5 N 4E	V 0-5 V 56	5 5 5 35
0 1 1 0	6	LC 12-9-6 86	BS 11-9-6 BS 08	ETB 0-9-6 ETB 17	UC 9-6 96	12-0-9-6 A5	12-11-9-6 AE	11-0-9-6 B6	12-11-0-9-6 C0	f 12-0-6 f 66	o 12-11-6 o 6F	w 11-0-6 w 77	12-11-0-6 DE	F 12-6 F 46	O 11-6 O 4F	W 0-6 W 57	6 6 6 36
0 1 1 1	7	DEL 12-9-7 DEL 7F	IL 11-9-7 87	ESC 0-9-7 ESC 1B	EOT 9-7 EOT 04	12-0-9-7 A6	12-11-9-7 AF	11-0-9-7 B7	12-11-0-9-7 C1	g 12-0-7 g 67	p 12-11-7 p 70	x 11-0-7 x 78	12-11-0-7 DF	G 12-7 G 47	P 11-7 P 50	X 0-7 X 58	7 7 7 37
1 0 0 0	8	GE 12-9-8 97	CAN 11-9-8 CAN 18	0-9-8 88	9-8 98	12-0-9-8 A7	12-11-9-8 B0	11-0-9-8 B8	12-11-0-9-8 C2	h 12-0-8 h 68	q 12-11-8 q 71	y 11-0-8 y 79	12-11-0-8 E0	H 12-8 H 48	Q 11-8 Q 51	Y 0-8 Y 59	8 8 8 38
1 0 0 1	9	RLF 12-9-8-1 8D	EM 11-9-8-1 EM 19	0-9-8-1 89	9-8-1 99	12-8-1 A8	11-8-1 B1	0-8-1 B9	8-1 ` 60	i 12-0-9 i 69	r 12-11-9 r 72	z 11-0-9 z 7A	12-11-0-9 E1	I 12-9 I 49	R 11-9 R 52	Z 0-9 Z 5A	9 9 9 39
1 0 1 0	A (10)	SMM 12-9-8-2 8E	CC 11-9-8-2 92	SM 0-9-8-2 8A	9-8-2 9A	¢ 12-8-2 [5B	! 11-8-2] 5D	: 12-11 : 7C	: 8-2 : 3A	12-0-8-2 C4	12-11-8-2 CB	11-0-8-2 D2	12-11-0-8-2 E2	12-0-9-8-2 E8	12-11-9-8-2 EE	11-0-9-8-2 F4	ǀ (LVM) 12-11-0-9-8-2 FA
1 0 1 1	B (11)	VT 12-9-8-3 VT 0B	CU1 11-9-8-3 8F	CU2 0-9-8-3 8B	CU3 9-8-3 9B	. 12-8-3 . 2E	$ 11-8-3 $ 24	, 0-8-3 , 2C	# 8-3 # 23	12-0-8-3 C5	12-11-8-3 CC	11-0-8-3 D3	12-11-0-8-3 E3	12-0-9-8-3 E9	12-11-9-8-3 EF	11-0-9-8-3 F5	12-11-0-9-8-3 FB
1 1 0 0	C (12)	FF 12-9-8-4 FF 0C	IFS 11-9-8-4 FS 1C	0-9-8-4 8C	DC4 9-8-4 DC4 14	< 12-8-4 < 3C	* 11-8-4 * 2A	% 0-8-4 % 25	@ 8-4 @ 40	12-0-8-4 C6	12-11-8-4 CD	11-0-8-4 D4	12-11-0-8-4 E4	⌐ 12-0-9-8-4 EA	12-11-9-8-4 F0	⌐ 11-0-9-8-4 F6	12-11-0-9-8-4 FC
1 1 0 1	D (13)	CR 12-9-8-5 CR 0D	IGS 11-9-8-5 GS 1D	ENQ 0-9-8-5 ENQ 05	NAK 9-8-5 NAK 15	(12-8-5 (28	) 11-8-5) 29	¯ 0-8-5 _ 5F	^ 8-5 ' 27	12-0-8-5 C7	12-11-8-5 CE	11-0-8-5 D5	12-11-0-8-5 E5	12-0-9-8-5 EB	12-11-9-8-5 F1	11-0-9-8-5 F7	12-11-0-9-8-5 FD
1 1 1 0	E (14)	SO 12-9-8-6 SO 0E	IRS 11-9-8-6 RS 1E	ACK 0-9-8-6 ACK 06	9-8-6 9E	+ 12-8-6 + 2B	; 11-8-6 ; 3B	> 0-8-6 > 3E	= 8-6 = 3D	12-0-8-6 C8	12-11-8-6 CF	11-0-8-6 D6	12-11-0-8-6 E6	⌐ 12-0-9-8-6 EC	12-11-9-8-6 F2	11-0-9-8-6 F8	12-11-0-9-8-6 FE
1 1 1 1	F (15)	SI 12-9-8-7 SI 0F	IUS 11-9-8-7 US 1F	BEL 0-9-8-7 BEL 07	SUB 9-8-7 SUB 1A	ǀ 12-8-7 ǀ 21	¬ 11-8-7 ^ 5E	? 0-8-7 ? 3F	" 8-7 " 22	12-0-8-7 C9	12-11-8-7 D0	11-0-8-7 D7	12-11-0-8-7 E7	12-0-9-8-7 ED	12-11-9-8-7 F3	11-0-9-8-7 F9	12-11-0-9-8-7 EO FF

LEGEND



CONTROL DATA CHARACTER SETS
SHOWING TRANSLATIONS BETWEEN DISPLAY CODE AND ASCII/EBCDIC

DISPLAY CODE		ASCII				EBCDIC			
		UPPER CASE		LOWER CASE		UPPER CASE		LOWER CASE	
OCTAL	CH	CH	HEX	CH	HEX	CH	HEX	CH	HEX
00	:	:	3A	SUB	1A	:	7A	SUB	3F
01	A	A	41	a	61	A	C1	a	81
02	B	B	42	b	62	B	C2	b	82
03	C	C	43	c	63	C	C3	c	83
04	D	D	44	d	64	D	C4	d	84
05	E	E	45	e	65	E	C5	e	85
06	F	F	46	f	66	F	C6	f	86
07	G	G	47	g	67	G	C7	g	87
10	H	H	48	h	68	H	C8	h	88
11	I	I	49	i	69	I	C9	i	89
12	J	J	4A	j	6A	J	D1	j	91
13	K	K	4B	k	6B	K	D2	k	92
14	L	L	4C	l	6C	L	D3	l	93
15	M	M	4D	m	6D	M	D4	m	94
16	N	N	4E	n	6E	N	D5	n	95
17	O	O	4F	o	6F	O	D6	o	96
20	P	P	50	p	70	P	D7	p	97
21	Q	Q	51	q	71	Q	D8	q	98
22	R	R	52	r	72	R	D9	r	99
23	S	S	53	s	73	S	E2	s	A2
24	T	T	54	t	74	T	E3	t	A3
25	U	U	55	u	75	U	E4	u	A4
26	V	V	56	v	76	V	E5	v	A5
27	W	W	57	w	77	W	E6	w	A6
30	X	X	58	x	78	X	E7	x	A7
31	Y	Y	59	y	79	Y	E8	y	A8
32	Z	Z	5A	z	7A	Z	E9	z	A9
33	0	0	30	DLE	10	0	F0	DLE	10
34	1	1	31	DC1	11	1	F1	DC1	11
35	2	2	32	DC2	12	2	F2	DC2	12
36	3	3	33	DC3	13	3	F3	TM	13
37	4	4	34	DC4	14	4	F4	DC4	3C

DISPLAY CODE		ASCII				EBCDIC			
		UPPER CASE		LOWER CASE		UPPER CASE		LOWER CASE	
OCTAL	CH	CH	HEX	CH	HEX	CH	HEX	CH	HEX
40	5	5	35	NAK	15	5	F5	NAK	3D
41	6	6	36	SYN	16	6	F6	SYN	32
42	7	7	37	ETB	17	7	F7	ETB	26
43	8	8	38	CAN	18	8	F8	CAN	18
44	9	9	39	EM	19	9	F9	EM	19
45	+	+	2B	VT	0B	+	4E	VT	0B
46	-	-	2D	CR	0D	-	60	CR	0D
47	*	*	2A	LF	0A	*	5C	LF	25
50	/	/	2F	SI	0F	/	61	SI	0F
51	(	(	28	BS	08	(	4D	BS	16
52	)	)	29	HT	09	)	5D	HT	05
53	$	$	24	EOT	04	$	5B	EOT	37
54	=	=	3D	GS	1D	=	7E	IGS	1D
55	SP	SP	20	NUL	00	SP	40	NUL	00
56	,	,	2C	FF	0C	,	6B	FF	0C
57	.	.	2E	SO	0E	.	4B	SO	0E
60	≡ =	=	23	ETX	03	=	7B	ETX	03
61	[	[	5B	FS	1C	¢	4A	IFS	1C
62	]	]	5D	SOH	01	!	5A	SOH	01
63	%	%	25	ENQ	05	%	6C	ENQ	2D
64	≠ "	"	22	STX	02	"	7F	STX	02
65	→ _	_	5F	DEL	7F	_	6D	DEL	07
66	∨ !	!	21	}	7D	¦	4F	¦	D0
67	∧ &	&	26	ACK	06	&	50	ACK	2E
70	↑ '	'	27	BEL	07	'	7D	BEL	2F
71	↓ ?	?	3F	US	1F	?	6F	IUS	1F
72	<	<	3C	{	7B	<	4C	¦	C0
73	>	>	3E	RS	1E	>	6E	IRS	1E
74	≤ @	@	40	`	60	@	7C	`	79
75	≥ \	\	5C	¦	7C	\	E0	¦	6A
76	¬ ^	^	5E	~	7E	¬	5F	~	A1
77	;	;	3B	ESC	1B	;	5E	ESC	27

NOTES:

1. The terms "upper case" and "lower case" apply only to the case conversions, and do not necessarily reflect any true "case".
2. When translating from Display Code to ASCII/EBCDIC, the "upper case" equivalent character is taken.
3. When translating from ASCII/EBCDIC to Display Code, the "upper case" and "lower case" characters fold together to a single Display Code equivalent character.
4. All ASCII and EBCDIC codes not listed are translated to Display Code 55 (SP).
5. Where two Display Code graphics are shown for a single octal code, the leftmost graphic corresponds to the CDC 64-character set (system assembled with IP.CSET set to C64.1), and the rightmost graphic corresponds to the CDC 64-character ASCII subset (system assembled with IP.CSET set to C64.2).
6. In a 63-character set system, the display code for the : graphic is 63. The % character does not exist, and translations from ASCII/EBCDIC % or ENQ yield blank (55_8).

GLOSSARY B

Absolute Address

The actual physical location of a word in central memory. Contrast with relative address.

Allocatable Device

A storage device that can be shared by more than one job.

Attach

To make a permanent file accessible to a job by specifying the proper permanent file identification and passwords.

Catalog

To place a file under jurisdiction of the permanent file manager, making it a permanent file.

Central Memory Resident (CMR)

Low core area of central memory reserved for tables, pointers, and subroutines necessary for operation of the operating system.

COMPASS

The assembly language of the CDC CYBER 170, CYBER 70 and 6000 Series computers.

Control Points

The concept by which the multiprogramming capability of CDC CYBER 170, CYBER 70 and 6000 Series computers is exploited. When a control point number is assigned to a job, that job is allocated some of the system resources; and it becomes eligible for assignment to the central processor for execution.

Control Statement

An instruction to the operating system or its loader. It is found in a section at the beginning of a job deck.

CYBER Control Language (CCL)

A group of control statements and commands that manipulate all control statements. With CCL, the user can conditionally skip or process control statements, process and reprocess a group of control statements, and process control statements in a file other than the job file. CCL is common to both NOS/BE and SCOPE 2 and is virtually identical on both systems.

CYBER Record Manager

A software package running under the NOS 1 and NOS/BE 1 operating systems that allows a variety of record types, blocking types, and file organizations to be created and accessed. The execution time input/output of COBOL 4, COBOL 5, FORTRAN Extended 4, Sort/Merge 4, ALGOL 4, and the DMS-170 products is implemented through CYBER Record Manager. Neither the input/output of the NOS/BE 1 operating system nor any of the system utilities such as COPY or SKIPF is implemented through CYBER Record Manger. All CYBER Record Manager file processing requests ultimately pass through the operating system input/output routines. SCOPE 2 record manager performs input/output for the SCOPE 2 operating system and its products. SCOPE 2 record manager is similar in capabilities and use to CYBER Record Manager.

Dayfile

A chronological system permanent file, maintained on a permanent file device, which forms an accounting and job history file. Entries, called dayfile messages, are generated by operator action or by the system when control statements are processed or other significant action occurs. The system dayfile has entries for the entire system. Every job receives a job dayfile with entries pertinent to that job.

Deadstart

The process of initializing the system by loading the system library programs and any of the product set from magnetic tape or a public device. Deadstart recovery is re-initialization after system failure.

Default

A system-supplied parameter value or name used when a value or name is not supplied by the user.

Dependency Count

A number established by the user with the Dyn parameter on a job statement and decremented by other jobs in the dependency string. The job is not run until the count reaches zero.

Dependent Job

A job which depends on the execution of other jobs before it can be run. It cannot be run until its dependency count is zero.

Device Set

A group of rotating mass storage devices. No device can belong to more than one device set. Every file must be contained within one device set, but can be on different devices in that device set.

Device Set Member

A rotating mass storage device belonging to a device set.

Device Type Code (dt)

An optional parameter on REQUEST statement or macro which specifies the type of device on which the named file is to be stored. It can encompass a group of parameters to define the device characteristics in detail.

Directive

A directive is control information that appears on a separate file or in a separate section of the job deck.

Dismountable Device

A rotating mass storage device which can be logically disassociated from the running system.

Display Code

Character code used internally in the computer. Each character consists of 6 bits (2 octal digits).

Disposition Code

A two-character mnemonic indicating device, site, form, and format for processing a file named on a ROUTE control statement and a DISPOSE statement or macro. Also, an octal value returned to the file environment table corresponding to the ultimate disposition of the file.

DMPX

A standard dump which appears on file OUTPUT when a job terminates abnormally. It shows the contents of the exchange package for the program, the contents of central processor registers, and the contents of words before and after the location at which the program stopped.

ECS

Extended core storage containing 60-bit words. ECS has a large amount of storage and very fast transfer rates.

EDITLIB

A utility program which allows creation or maintenance of library files suitable for use by the loader.

End-of-Information

Physical end of data. In card decks, a card with a 6/7/8/9 multiple punch in column one. On SI tapes and on labeled S and L tapes, a tape mark followed by an EOF trailer label followed by two tape marks. On mass storage devices, the position of the last written data. CYBER Record Manager defines end-of-information in terms of file residency and organization.

End-of-Tape Reflective Marker

A reflective strip near the end of a magnetic tape. It is used to signal termination of operations on a particular volume. At least 18 feet of tape must follow this marker.

EST Ordinal

The number designating the position of an entry within the equipment status table established at each installation.

Evict

Evict releases all space occupied by a file to the system, including space occupied by entries in system tables.

Exchange Package

A 16-word package containing information used in exchange jumps during job execution: contents of central processor registers, RA and FL in central memory and in ECS and the program address. It is stored in the control point area and printed as part of the standard output of an aborted job.

Field Length (FL and FE)

FL is the number of central memory words assigned to a job. FE is the number of words in the direct access area of extended core storage assigned to a job. Within central memory or extended core storage, the field length added to the reference address defines the upper address limit of a job.

File

A file is a set of information that begins at beginning-of-information and ends at end-of-information and that has a logical file name. All files have at least one partition, which is delimited by a system-logical-record of level 17 on mass storage files or tapes in SI format, and by a tape mark on S or L tapes.

File Environment Table (FET)

A table used for communication between a user program and the operating system when files are processed. An FET created by a compiler or by the assembly language programmer is required within the user field length for each file in the program.

File Set

One or more related files recorded on one or more volumes.

Full track (FT)

Reading/writing sequential sectors on an 844 disk pack (1:1 interlace).

Half track(HT)

Reading/writing alternate sectors on an 844 disk pack (2:1 interlace).

Hang

A system stop that may be caused by hardware failure or by an error in a system program. An error in a user program could cause that program to hang (go into a loop or abort). but no user program error should cause a system hang.

Job Step

Each individual control statement is a job step. A group of job steps forms a job stream.

Job Stream

A job stream is a group of control statements found at the beginning of a deck.

INPUT

A logical file name assigned by the system to every job. It contains the image of user job deck.

JANUS

A group of system peripheral processor routines which controls the processing of input and output files. JANUS controls up to 4 card readers, 3 card punches, and 12 line printers. It normally functions at control point 1, but can be assigned to another control point by the operator.

L Tape

A labeled or unlabeled magnetic tape containing physical records whose sizes range from one central memory word to an upper limit specified by the size of the buffer for that tape.

Labeled Tape

A magnetic tape with header and trailer labels having the format of the CDC CYBER 170, CYBER 70 or 6000 Series standard labels or the 3000 Series labels; alternately a tape in S or L format with non-standard labels.

Level

An indicator specifying relative position in a hierarchy. For priority considerations, level 0 is the lowest priority. For system-logical-records, octal level numbers 0-17 can be used to organize files. For overlay and segment loading, a pair of numbers specifies level, with (0,0) being the portion of the program remaining in memory.

Level Number

An octal number from 0-17 in a short physical record unit or zero-length physical record unit marker to form system-logical-record groups within files. Level number 17 indicates a logical end-of-partition. Level number 16 is used by checkpoint/restart and should not otherwise be specified by the user. The system creates system-logical-records with a level number of 0 for mass storage files and SI tapes when the user does not specify otherwise.

Library

A file or collection of files containing executable programs and tables needed to locate and load the programs. A system library can contain peripheral processor programs in addition to the central processor programs. A user library is file formatted as a library but is not available to a job until it has been explicitly brought to the job.

Load Point

The reflective marker near the beginning of a magnetic tape. Data, including labels, is written after the load point. A rewind positions a single file volume to the load point. At least 10 feet of tape should precede the load point marker.

Load Sequence

A sequence of load operations which encompasses all of the loader's processing from the time that nothing is loaded until the time execution begins. It includes initialization, specification of specified loader requests, and completion of load.

Logical File Name (1fn)

The 1-7 display coded alphabetic or numeric characters by which the operating system recognizes a file. Every 1fn in a job must be unique and begin with a letter.

Macro

A COMPASS language statement which generates other source language code.

Master Device

The member of a device set designated as the device to contain all device set related tables. Every device set has one device that is a master device.

Mount

A logical operation that associates a device set member with a job.

Monitor

The system routine which coordinates and controls all activities of the computer system. It occupies peripheral processor 0 and part of central memory. It schedules the use of the central processor and the other peripheral processors.

Non-allocatable Device

A device such as a magnetic tape which can be used by only one job at a given time.

NUCLEUS

A system library containing the essential system programs needed to load and execute all other system library programs. It is available to all jobs without explicit call.

OUTPUT

A logical file name assigned by the system to each job to receive information such as assembly listing. diagnostics, load map, dayfile, and program output. It is printed at job termination unless otherwise disposed by the user.

Partition

A partition is a system-logical-record of level 17 on a mass storage file or a tape in SI format. On a S or L tape, it is delimited by a tape mark.

Password

A string of 1-9 letters or digits defining access permission assigned at attach time. Each password implies one type of access permission designated for permanent files, such as read, modify, extend, control or turnkey.

Permanent File

A mass storage file cataloged by the system so that its location and identification are always known to the system. Permanent files cannot be destroyed accidentally during normal system operation (including deadstart). They are protected by the system from unauthorized access according to privacy controls specified when they are created.

Physical Record Unit (PRU)

The smallest amount of information transmitted by a single physical operation of a specified equipment, measured in central memory words. A PRU for mass storage devices is 64 decimal words long; that for SI format binary magnetic tape is 512 decimal words: etc.

Private Device

A mass storage device which can be used only by specific request. It is logically removable, and is a member of a private device set.

PRU Device

A mass storage device or tape in SI format.

Public Device

An allocatable mass storage device available to the operating system for assignment of default residence files.

PUNCH

A logical file name that causes the file to be punched on cards in Hollerith format when the job terminates.

PUNCHB

A logical file name which causes the file to be punched on cards in binary format when the job terminates.

Random File

A file with an index entry to each record in the file. A file on a rotating mass storage device is a random file only when the random bit is set in the file environment table. The last record of the file is an index.

Recall

The state of a program when it has released control of the central processor until a fixed time has elapsed (periodic recall) or until a requested function is complete (auto recall). Recall is a system action request. as well as an optional parameter of some file action requests.

Record

CYBER Record Manager defines a record or a portion thereof as the smallest collection of information passed between CYBER Record Manager and a user program. Eight record types exist, as defined by the RT field of the file information table (FIT). Other parts of the operating systems and their products might have additional or different definition of records.

Reference Address (RA and RE)

RA is the absolute central memory address that is the starting, or zero relative address assigned to a program. Addresses within the program are relative to RA. RA+ 1 is used as the communication word between the user program and Monitor. RE is the absolute extended core storage starting address assigned to file.

Relative Address

All addresses in a relocatable program are relative to a base address of zero. When a relocatable program is loaded for execution, the zero base address is assigned to a reference address. At that time, all addresses in the program become relative to the reference address.

Removable Device

A rotating mass storage device which can be physically detached from the RMS drive.

Retention Period

The number of days a permanent file or a device set is to be valid.

Rolling

The concept of removing jobs from central memory to mass storage before execution is complete so memory can be assigned to a higher priority job.

Rotating Mass Storage (RMS)

Disk storage device.

S Tape (stranger tape)

A magnetic tape (labeled or unlabeled. 7 or 9 track) containing physical records ranging in size from 2 characters to 5120 decimal characters. This tape does not contain any level numbers.

Scheduler

A group of system routines which select jobs for assignment to control points and control swapping and rollout of jobs.

System-Logical-Record

A data grouping that consists of one or more physical record units immediately followed by a short physical record unit or a zero-length physical record unit. These records can be transferred between devices without loss of data or structure. A system-logical-record is equivalent to a CYBER Record Manager S type record.

Sequential File

A file in which records must be located by position, not address.

Short PRU

A physical record unit containing data and a marker with an octal level number to mark the end of a system-logical-record. The amount of user data in a short PRU is less than the PRU size of the storage device. A short PRU defines the end of a system-logical-record. In CYBER Record Manager, a short PRU may have several interpretations that depend upon record and block type.

SI Tape

A magnetic tape created under NOS/BE 1 with fixed length physical record units. For coded tape = 128 decimal central memory words; for binary tape = 512 decimal central memory words. An SI tape can be labeled or unlabeled, and written on 7-track or 9-track tape. Identical to SCOPE tape under SCOPE 3.3 and 3.4 and to SI format tape under NOS 1 and KRONOS 2.1.

Staging

Releasing a tape job from the tape queue for scheduling.

Standard Labeled Tape

A tape with labels conforming to American National Standard Magnetic Tape Labels for Information Interchange X3.27-1969. Also called a system labeled tape.

Swapping

The concept of removing jobs form central memory to mass storage before execution is complete, so control point and memory can be assigned to another job. A job is swapped out when it is waiting for an external event, or when its control point and/or central memory is needed by a higher priority job.

System Device

A system device is a device that holds system information. All system devices are PRU devices but not all PRU devices are system devices.

System Libraries

The collection of tables and object language programs residing in central memory or on mass storage, which are necessary for running the system and its product set.

Tape Mark

A short record written on tapes under operating system control to separate label groups, files, and/or labels. Interpretation depends on the tape format.

Unlabeled Tape

A magnetic tape that does not have a header label. Unlabeled tapes generated by the operating system contain a trailer label similar to the trailer for a standard labeled tape.

Unit Record Device

A standard unit record device such as the line printer, card punch, and card reader runs under control of JANUS. A non-standard unit record device, which includes graphic consoles, plotters, and paper tape readers and punches, run under installation software.

UPDATE

A utility program that allows a source statement program stored on mass storage or tape in UPDATE format to be modified and restored.

User Library

Library file a programmer created through the EDITLIB utility. It contains loader tables referencing the assembled central processor programs, subroutines, text records, or overlays.

Volume

A term synonymous with reel of tape.

Zero-Length PRU

A physical record unit, containing only an octal level number, that is used to terminate a system-logical-record; it does not contain any user data. In CYBER Record Manager, a zero-length PRU with a level designator of 17 is a partition boundary.

Zero-Byte Terminator

The 12 bits of zero in the low order position of a central memory word are used to terminate a line of coded information to be output to a line printer or to represent cards input through a card reader. Files with names INPUT and OUTPUT have such terminators while in storage. Any file to be displayed at a terminal must also have such terminators for each line to be displayed correctly. A record with such a terminator in CYBER Record Manager is a zero-byte record (Z type record).

In display code, two colons create 12 bits of zeros. If two consecutive colons occur in a file that contains zero-byte records, they may be stored in the lower order portion of a word and create a zero-byte record.

Files created at a terminal under the CREATE command contain zero-byte terminated records.

CONTROL STATEMENT SUMMARY C

All operating system requests that can be issued on control statements are noted in the following summary, together with parameters issued as part of each request. In this summary, constants are capitalized and variables are in lower case; the variables are defined below the illustrated format.

A full description of each control statement appears in alphabetic order in section 4.

Required characteristics of parameters appearing frequently are:

lfn	Logical file names of 1-7 letters or digits beginning with a letter
vsn	Volume serial number of 1-6 letters of digits, leading zeros are assumed
lev	Octal level number 0-17 for system-logical-records
setname	Device set name of 1-6 letters or digits beginning with a letter

ABS,from,thru.

from	Beginning address of dump
thru	Ending address of dump

ABS dumps absolute addresses of central memory.

ACCOUNT,parameter list.

ACCOUNT supplies accounting information.

ADDSET,MP=vsn,VSN=vsn,SN=setname,NF=n,NM=m,RP=ddd,*PF,*Q,mode.

vsn	Volume serial number
setname	Device set name
n	Maximum number of permanent files on device set
m	Maximum number of members allowed in device set
ddd	Retention period
*PF	Permanent file device
*Q	Queue file device
mode	Recording mode (HT or FT)

ADDSET establishes a master device; adds members to a device set.

ALTER,lfn.

lfn	Logical file name

ALTER sets EOI to current position.

ATTACH,lfn,pfn,ID-name,AC=act,CY=cy,EC=ec,LC=n,MR=m,PW=pw,RW=p,SN=setname.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account
cy	Cycle number
ec	ECS buffer size
n	Lowest cycle
m	Multi-read access
pw	Passwords
p	Multi-read with single rewrite or single extend access
setname	Device set name

ATTACH assigns permanent file to job.

AUDIT,LF=lfn,MO=m,ID=name,PF=pfn, $\left\{\begin{matrix}AI=F\\AI=P\end{matrix}\right\}$ SN=setname,VSN=vsn,AC=n.

lfn	Logical file name
m	AUDIT mode
name	Creator identifier
pfn	Permanent file name
F	Full two-line output
P	Partial one-line output
setname	Device set name
vsn	Volume serial number
n	Account number

AUDIT provides a status of permanent files.

BEGIN,pname,pfile,p_1,p_2,. . .,p_n.

pname	Procedure name
pfile	Name of the file where procedure pname is located
p_i	A parameter having one of the following forms:
	fk Formal keyword
	v Value
	fk=v Value v is substituted for formal keyword fk

BEGIN calls procedure pname on file pfile.

BKSP,lfn,n.

lfn	Logical file name
n	Number of records (decimal) to be backspaced

BKSP backspaces n system-logical-records.

CATALOG,lfn,pfn,ID=name,AC=act,CY=cy,CN=cn,EX=ex,FO=fo,MD=md,MR=m,PW=pw,RD=rd,RP=rp,RW=p,
TK=tk,XR=xr.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account parameter
cy	Cycle number
cn	Control password
ex	Extend password
fo	File organization IS, DA, or AK
md	Modify password
m	Multi-read access
pw	Password list
rd	Read password
rp	Retention period
p	Multi-read with single rewrite or single extend access
tk	Turnkey password
xr	Password for modify, extend, and control permissions

CATALOG makes mass storage file permanent.

CKP.

CKP establishes checkpoint.

COMBINE,lfn1,lfn2,n.

lfn1	Input file
lfn2	Output file
n	Number of records (decimal)

COMBINE combines input file records into one logical record of level 0 on output file

COMMENT.comment.

comment	Comment characters

COMMENT inserts comments.

COMPARE,lfn1,lfn2,n,lev,e,r.

lfn	Logical file name
n	Number of records (decimal) to compare
lev	Level number of system-logical-records
e	Number of non-comparable words to be written
r	Number of records to be processed during comparison

COMPARE compares pairs of files or records.

COPY,lfn1,lfn2.

lfn	Logical file name

COPY copies all files in a volume lfnl to lfn2.

COPYBCD,lfn1,lfn2,n.

lfn	Logical file name
n	Number of records (decimal)

COPYBCD copies packed output files to tape for subsequent off-line listing.

COPYBF,lfn1,lfn2,n.

lfn	Logical file name
n	Number of partitions (decimal)

COPYBF copies n binary partitions from lfn1 to lfn2.

COPYBR,lfn1,lfn2,n.

lfn	Logical file name
n	Number of records (decimal)

COPYBR copies n binary records from lfn1 to lfn2.

COPYCF,lfn1,lfn2,n.

lfn	Logical file name
n	Number of partitions (decimal)

COPYCF copies n Hollerith or External BCD partitions from lfn1 to lfn2.

COPYCR,lfn1,lfn2,n.

lfn	Logical file name
n	Number of records (decimal)

COPYCR copies n Hollerith or External BCD records from lfn1 to lfn2.

COPYN,f,outlfn,inlfn, ...

f	Record format
outlfn	Output file
inlfn	Input file

COPYN accepts input from up to 10 binary files to be copied to the output file (input directive record required).

COPYSBF,lfn1,lfn2.

lfn1	Input file
lfn2	Output file

COPYSBF copies lfn1 to lfn2 formatting binary file for single space printing.

COPYXS,xlfn,scplfn,n.

xlfn	Input X tape must be requested in SI format
scplfn	Output tape must be requested in SI format
n	Number of partitions

COPYXS converts binary X tapes to SI tape format.

DELSET,SN=setname,MP=vsn1,VSN=vsn2.

setname	Device set name
vsn1	Master device volume serial number
vsn2	Member volume serial number

DELSET deletes a member from a device set.

DISPLAY,exp.

exp	CCL expression

DISPLAY evaluates exp and sends the result to the user dayfile.

DISPOSE,lfn, { *dc / *dc = C / dc = Cff / dc = Iid }.

lfn	Logical file name
*	Defer disposition until EOJ
dc	Disposition code
C	Central site route
ff	Special paper or card
Iid	INTERCOM terminal route

DISPOSE disposes a file immediately or at EOJ.

DMP,from,thru.

from	Beginning address in dump
thru	Last address in dump

DMP dumps specified area of central memory.

DMPECS,from,thru,format,lfn.

from	Beginning dump address
thru	Last dump address
format	Print format
lfn	File to receive dump

DMPECS dumps specified area of ECS.

DSMOUNT,VSN=vsn,SN=setname.

vsn	Volume serial number
setname	Device set name

DSMOUNT logically disassociates device set from job.

DUMPF,PW=pw,MO=n, $\left\{\begin{matrix} I \\ I=lfn1 \end{matrix}\right\}$,LF=lfn2,CL,DP=a,ID=name,PF=pfn,CY=cy,SN=sn,VSN=vsn, $\left\{\begin{matrix} IN=ddd \\ JN=yyddd \\ LA=mmddyy \\ DA=yyddd \\ CD=mmddyy \\ TI=hhmm \end{matrix}\right\}$.

pw	Password
n	Dump mode
I	INPUT file
lfn1	Logical file name
lfn2	Output listing file
CL	Complete list
a	Dump type
name	Creator identifier
pfn	Permanent file name
cy	Cycle number
sn	Set name
vsn	Volume serial number
ddd	Number of days
yyddd	Year-day date
mmddyy	Month-day-year date
hhmm	Hour-minute time qualifier

DUMPF dumps permanent files to tape.

EDITLIB,USER,I=lfndir,L=lfnlist.

lfndir	Directive input file
lfnlist	List file of output

EDITLIB creates a USER library file (directive record required; see section 4 under EDITLAB for directives and their formats).

ELSE,ls.

ls	Label string

ELSE initiates a skip or terminates a skip initiated by an IFE statement.

ENDIF,ls.

ls	Label string

ENDIF terminates a skip initiated by an IFE, ELSE, or SKIP statement.

ENDW,ls.

ls	Label string

ENDW, when used in conjunction with a WHILE statement, brackets a group of control statements to be conditionally reprocessed.

EXECUTE.

EXECUTE completes loading and linking of elements for execution, then executes this program. Also refer to the LOADER RM, section 3, for additional information.

EXIT,$\left\{\begin{matrix}C\\U\\S\end{matrix}\right\}$.

omitted	Execute if non-special error
C	Execute
U	Execute if non-special fatal error
S	Execute if either special or non-special error

EXIT establishes exit path in event of selected errors.

EXTEND,lfn.

lfn Logical file name

EXTEND makes permanent an extension to permanent file.

GETPF,lfn,pfn,ID=name,AC=act,EC=ec, { LC=n / CY=cy } ,MR=m,PW=pw,RW=p,ST=mmf.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account
ec	ECS buffer size
n	Lowest cycle
cy	Cycle number
m	Multi-read access
pw	Password
p	Multi-read with single rewrite or single extend access
mmf	System on which file is cataloged

GETPF assigns a local copy of a permanent file to job.

IFE,exp,ls.

exp	CCL expression
ls	Label string

IFE conditionally causes skipping of following control statements.

LABEL,lfn,{W|R},{Z|Y},{RING|NORING},IB,D=d,F=f,N=n,X=x,L=z,V=v,E=e,T=t,C=c,M=m,P=p,VSN=vsn.

lfn	Logical file name
R	Read file
W	Write file
Y	3000 series label
Z	Standard label
RING	Write-enabled ring required
NORING	Write-enabled ring prohibited
IB	Inhibit system noise brackets
d	Density
f	File data format
n	9-track tape code conversion
x	Tape disposition
z	Label name
v	Volume number
e	Edition number
t	Retention period
c	Creation date
m	Multi-file set name
p	Position number
vsn	Volume serial number

LABEL writes or checks on a tape file.

LABELMS(DT=eq,mode,I=lfn)

eq	Device type
mode	Recording mode (HT or FT)
lfn	Logical file name for input directives

LABELMS blank labels a disk before use in a device set.

LIMIT,n.

n	Octal number representing the multiple of 4096 60-bit words

LIMIT limits amount of mass storage assigned to job.

LISTMF,M=mfn,P=p.

mfn	Multi-file name
p	File position number to begin listing

LISTMF lists contents of a labeled multi-file tape.

LOAD,lfn1/r,lfn2/r, ...

lfn	Logical file name
r	Rewind indicator

LOAD loads programs on lfn into central memory.

LOADPF,LP=x,LF=lfn,CL,SN=setname,VSN=vsn,ID=name,PF=pfn,CY=cy, $\left\{\begin{matrix}\text{I=lfn}\\\text{I}\end{matrix}\right\}$.

x	Load files
lfn	Logical file name of listing
CL	Complete list
setname	Device set name
vsn	Volume serial number
name	Creator identifier
pfn	Permanent file name
cy	Cycle number
lfn	Logical file name of directive
I	INPUT file

LOADPF loads permanent files that have been dumped to tape.

MAP, $\left\{\begin{matrix}\text{OFF}\\\text{FULL}\\\text{PART}\end{matrix}\right\}$.

OFF	No map
FULL	Full map
PART	Partial map

MAP selects map, no map, or partial map option.

MODE(m)

m	Program halt conditions

MODE defines program halt conditions.

MOUNT(VSN=vsn,SN=setname,mode)

vsn	Volume serial number
setname	Device set name
mode	Recording mode (HT or FT)

MOUNT must be used to access a device set.

PAUSE,comment.

PAUSE inserts formal comments – operator must respond.

PURGE,lfn,pfn,ID=name,AC=act,{LC=n / CY=cy},EC=ec,MR=m,PW=pw,RB=1,RW=p,SN=setname,ST=mmf.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account
n	Lowest cycle
cy	Cycle number
ec	ECS buffer size
m	Multi-read access
pw	Password
1	Record block conflict
p	Multi-read with single rewrite or single extend access
setname	Device set name
mmf	System on which file is cataloged

PURGE removes a permanent file from the system.

RECOVER,SN=setname,VSN=vsn.

setname	Device set name
vsn	Master device volume serial number

RECOVER validates a device set; reconstructs device set tables.

REDUCE.

REDUCE reduces field length of job after loading.

RENAME,lfn,pfn,ID=name,AC=act,CN=cn,CY=cy,EX=ex,MD=md,RD=rd,RP=rp,TK=tk,XR=xr.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account parameter
cn	Control password
cy	Cycle number
ex	Extend password
md	Modify password

rd	Read password
rp	Retention period
tk	Turnkey password
xr	Password for control, modify and extend permission

RENAME renames control information for permanent files.

REQUEST,lfn,dt,parameters.

lfn	Logical file referenced
dt	Device or device type; dt field can be expanded

REQUEST assigns file to a device – see section 4.

REQUEST,lfn,AX,EC.

lfn	Logical file name
AX	ECS device type
EC	ECS buffer file

REQUEST assigns file to ECS device.

REQUEST,lfn,dtaa,OV,EC,*PF,*Q, $\left\{\begin{matrix} SN \\ SN=setname \end{matrix}\right\}$,VSN=vsn.

lfn	Logical file name
dtaa	Device type mnemonic and allocation style
OV	Overflow
EC	ECS buffer file
*PF	Permanent file device
*Q	Queue file device
SN	Setname specified by SETNAME control statement
setname	Device set name
vsn	Volume serial number

REQUEST assigns file to permanent file device on private or public device set.

RESTART,name,n,S=xxx.

name	Name of checkpoint file
n	Decimal number of checkpoint where job is to be restarted
xxx	Decimal number of words in smallest physical record (used only with S tape)

RESTART restarts job at check-point.

RETURN,lfn1,lfn2, ...

lfn Logical file or multi-file set names

RETURN releases files and associated devices from job and decrements tape unit required count.

REVERT,ABORT.

omitted Normal return to calling job or procedure

ABORT Abort after return to calling job or procedure

REVERT causes processing to return to the calling job or procedure.

REWIND,lfn1,lfn2, ...

lfn Logical file name

REWIND rewinds file named.

RFL,fl.

fl New field length

RFL redefines job field length.

ROUTE,lfn,DEF, {DC / DC=dc}, {EC / EC=ec}, {FC / FC=fc}, {FID / FID=fid}, {IC / IC=ic}, {PRI / PRI=pri}, {REP / REP=n}, {SC=nn / SC}, {ST / ST=mmf}, {TID / TID=tid / TID=C}.

lfn Logical file name

DEF Defer disposition code

DC Evict file

dc File disposition

EC JANUS print files

ec External characteristics of file

FC Standard forms

fc Forms code

FID Output queue file name same as job name

fid File name in output queue

IC Display code file format

ic Internal characteristics of file

PRI Standard priority

pri Priority level

REP No extra copies

n Repeat count

SC	Spacing code for 580 PFC print files
nn	Spacing code array
ST	Process file on original system
mmf	System on which file is cataloged
TID	Return job to original site
tid	INTERCOM terminal identification
C	Output at central site

ROUTE directs a file to input or output queue.

SAVEPF,lfn,pfn,ID=name,AC=act,CN=cn,CY=cy,EX=ex,FO=fo,MD=md,MR=m,PW=pw,RD=rd,RP=rp,RW=p,
ST=mmf,TK=tk,XR=xr.

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
act	Account parameter
cn	Control password
cy	Cycle number
ex	Extend password
fo	File organization IS, DA, or AK
md	Modify password
m	Multi-read access
pw	Password list
rd	Read password
rp	Retention period
p	Multi-read with single rewrite or single extend access
mmf	System on which file is cataloged
tk	Turnkey password
xr	Password for modify, extend, and control permissions

SAVEPF makes local file permanent.

SET,sym=exp.

sym	CCL symbolic name
exp	CCL expression

SET sets the value of CCL symbolic names.

SETNAME,setname.

setname	Device set name

SETNAME implicitly references device sets.

SKIP,ls.

ls	Label string

SKIP causes unconditional skipping of the control statements that follow it.

SKIPB,lfn,n,lev,mode.

lfn	Logical file name
n	Number of records (decimal) to skip
lev	Level number of system-logical-records to skip
mode	B for binary files – C for coded files

SKIPB skips backward by n system-logical-records of level lev or greater.

SKIPF,lfn,n,lev,mode.

lfn	Logical file name
n	Number of records (decimal) to be skipped
lev	Level number of system-logical-records to skip
mode	B for binary files – C for coded files

SKIPF skips file forward by n system-logical-records of level lev or greater.

SUMMARY.

SUMMARY provides accounting information.

SWITCH,n.

n	Sense switch number

SWITCH sets switch to on or off.

SYSBULL,p1,p2, . . . ,pn.

pi	Bulletin names

SYSBULL copies system bulletins to OUTPUT file.

TRANSF,job1,job2, ...

job	Name of next job for execution

TRANS decreases dependency count of named jobs.

TRANSPF,PW=pw,FS=setname1,TS=setname2,FM=vsn1,TM=vsn2,LF=lfn.

pw	Password list
setname1	Device set name from which permanent file information is transferred
setname2	Device set name to which information is transferred
vsn1	Volume serial number of member from which information is transferred
vsn2	Volume serial number of member to which information is transferred
lfn	Logical file name of output listing

TRANSPF transfers permanent files.

UNLOAD,lfn1,lfn2, ...

lfn	Logical file or multi-file set names

UNLOAD is same as RETURN but does not decrement tape unit count.

VSN,lfn1=vsn1,lfn2=vsn2, ...

lfn	Logical file name
vsn	Volume serial number associated with lfn

VSN equates vsn to file name.

WHILE,exp,ls.

exp	CCL expression
ls	Label string

WHILE, when used in conjunction with ENDW, conditionally reprocesses a group of control statements.

PUNCH CARD AND TAPE FORMAT D

This appendix contains details of the format of punch cards and magnetic tape. Two types of card format are discussed: Hollerith or coded cards and binary cards, which include the separator cards used between sections of a deck and between decks. Magnetic tape format is discussed in terms of the binary and coded formats produced on 7- and 9-track tapes in SI format.

PUNCH CARD FORMATS

Punch card formats can be coded Hollerith, standard binary, and free-form binary.

Hollerith cards are produced when the file name is PUNCH, or the disposition code of the output file is PU (octal 0010). Unused columns at the end of the last Hollerith card are blank; a card with 7/8/9 multipunch follows the last card produced.

Standard binary cards are produced when the file name is PUNCHB, or the file has a disposition code of PU, IC=BIN, and EC=SB (octal 1210).

Free-form binary cards are produced when the file name is P80C, or the file has a disposition code of PU, IC=BIN, and EC=80COL (octal 2210). If the number of words to be punched in free-form is not an even multiple of 16, the unused columns at the right of the last punched card are blank. A card with 7/8/9 is produced following the last free-form binary card. The flag cards are not punched as part of the output.

HOLLERITH FORMAT

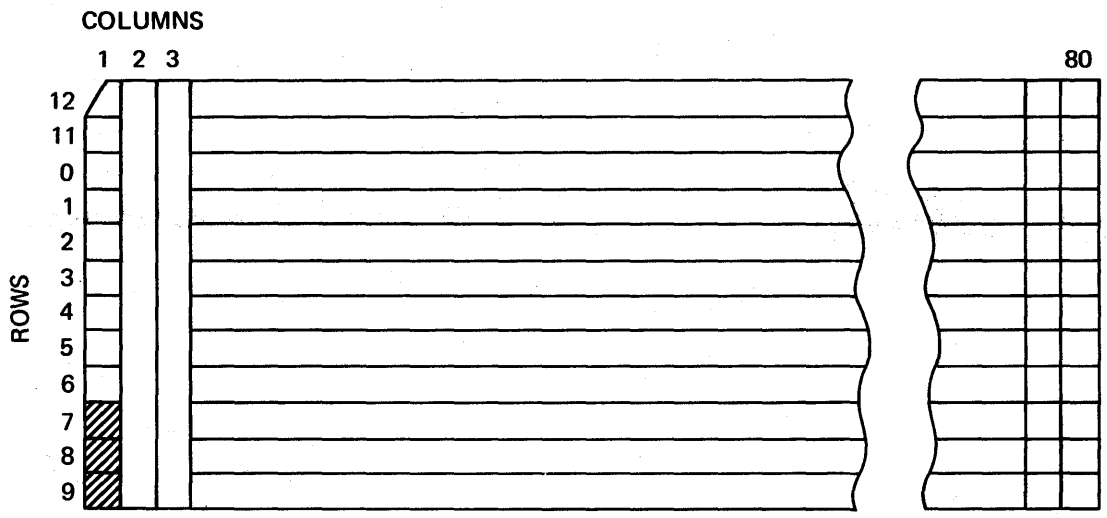
Hollerith cards are often called coded cards. Each column can be punched to represent codes of any given character set (see appendix A). The hole code is translated by card reading devices into the binary code for the character. Blank columns are translated into a binary code representing a blank space.

Hollerith punch cards can be in 026 or 029 format. 026 mode is a 63- or 64-character set defined by Control Data. 029 mode is a Control Data 64-character subset of the codes defined by the American National Standard Code for Information Interchange, X3.4-1968 (ASCII mode).

Each installation selects the default mode for cards to be read into the system, but cards in an alternative mode can be read when the job indicates another card mode. Appendix A shows card codes for 026 and 029 modes and discusses how to change modes within a job deck.

END-OF-SECTION CARD

A card containing octal 0007 (7/8/9) in column 1 separates sections in a job deck. Level numbers associated with the record are punched in Hollerith code in columns 2 and 3. The level number may be 00,01,02,03,04,05,06,07, 10,11,12,13,14,15,16, or 17. If columns 2 and 3 are blank, the level number is assumed to be 00. Level numbers 1-7 may be punched with a trailing blank in the form nb, where n is the level number and b is a blank. The format of this card is as follows:



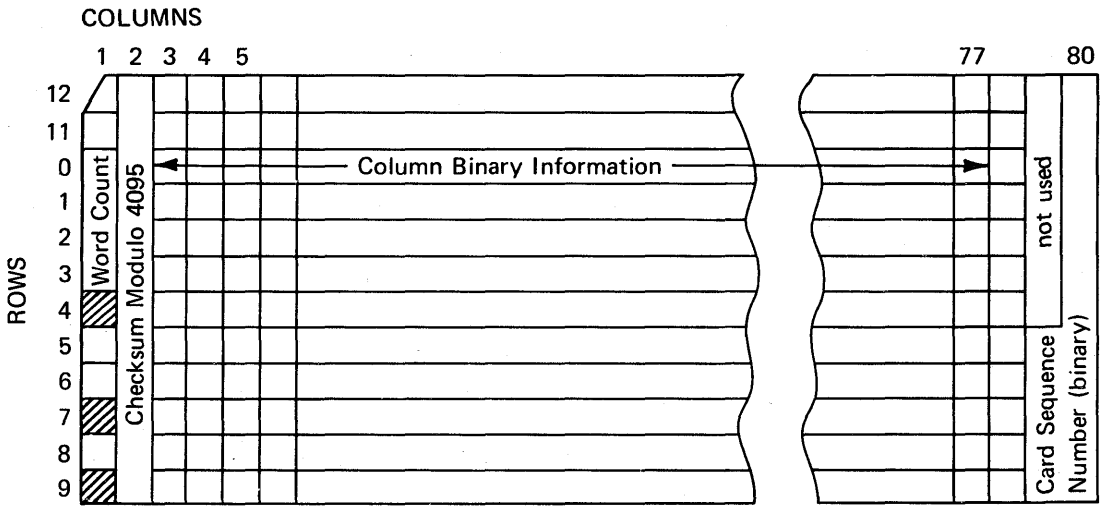
STANDARD BINARY CARDS

All standard binary cards must have punches in rows 7 and 9 of column 1; thus, any four octal digits ending with 5 or 7 would act as a binary card marker. Any card without a 7/9 punch in column 1 is considered to be a Hollerith card; no legal Hollerith code contains a 7/9 punch combination. Any Hollerith card column containing an illegal Hollerith punch combination is read as a blank, and a message is produced for output giving the card number and the number of the record containing the card.

Binary subprogram or data cards can contain the binary representation of up to 15 central memory words. This card type contains a 7/9 punch with a word count in rows 0/1/2/3 and a checksum flag in row 4, all in column 1. The word count indicates the number of binary words in the card, starting in column 3 and not extending beyond column 77. Column 2 contains a checksum of the binary words in columns 2 thru 77; the value of the checksum is a ones-complement sum, modulo 4095 (2**12 - 1=4095). If the checksum flag in row 4 of column 1 is punched, the checksum is ignored by the system. Columns 79 and 80 contain a card sequence number in binary. The lower five bits in column 79 and all 12 bits in column 80 make up the 17 bit serial number of the card record within the logical record that contains it. If cards are not read in sequential order, a warning message is produced for output; however, the cards are read and accepted.

Columns 1, 2, 78, and 80 are produced when a binary punch file is punched through a remote terminal or JANUS controlled device. These columns are removed when the deck is read into the system, so that a card has only 15 central memory words of information internally.

The format of a binary subprogram card is as follows:



FREE-FORM BINARY CARDS

Free-form binary cards are unique since they can be read as sixteen 60-bit words per card (eighty 12-bit columns) with no checksum or sequence number. For example, a card having 6/7/8/9 punched in column 1 and at least one punch in one other column can be read as a free-form binary card. Normally, it would be treated as an end-of-information card.

Free-form binary cards must be preceded by a flag card with all 12 rows punched in column 1 and any other column and no other punches. This flag card is not read as containing information; it signals that free-form binary cards follow

Any number of cards may follow; none may have the same form as the free-form flag card or a 6/7/8/9 end-of-information card. The free-form binary cards are read into memory in 16-word increments. After the free-form binary cards, another flag card with 12 rows punched in colum 1 and the same column as the first flag card must appear. This card signals the end of the free-form binary deck and standard binary or Hollerith cards follow. The operator's console displays TRAY EMPTY until a matching flag card is read.

If it is necessary for a free-form binary card with the same appearance as the flag card to appear in the deck, it is possible to create a flag card of a different form. Any card having 12 rows in column 1 punched and 12 rows in any other column punched with no other punches on the card is recognized as a free-form flag; therefore, 79 variations are possible for the flag card.

Normally, a series of free-form binary cards and their flag cards are organized into one record in an input file. However, they can be preceded and/or followed by standard binary and/or Hollerith cards within the same record. The different cards in the record are accepted; however, a message indicating a change in mode is produced for the record. A valid record might consist of the following:

1. A series of Hollerith cards.
2. A start free-form flag card (7777 in columns 1 and 80) with no other punches.
3. A series of free-form binary cards without a standard 6/7/8/9 card or any card identical with 2.
4. An end free-form flag card identical with 2.
5. A start free-form flag card, which might be the same as or different from 2 and 4.
6. A series of free-form binary cards as in 3.
7. An end free-form flag card identical with 5.
8. A series of standard binary cards which should be in order according to sequence numbers. If not, a sequence number check message and a mode change message are issued for the record.
9. A 7/8/9 card.

TAPE FORMAT

7-TRACK CODED SI FORMAT

For coded data being output on 7-track tape, the PP converts display code to internal BCD codes if a 6684 converter is not available. The tape controller converts internal BCD to the external BCD codes recorded on the tape. In the 63-character set display code, characters 33 and 63 convert to an external BCD 12. However, if the last two characters of a central memory word have a display code representation of 0000 (end-of-line delimiter byte), they become an external BCD 1632.

For 7-track coded tapes being read in, the tape controller converts external BCD to internal BCD codes. The PRU converts the internal BCD to display codes (if a 6684 converter is not available) before transferring data to the file buffer. On input, the external BCD 12 is converted to a display code 33 (zero). The end-of-line delimiter byte, which must occur at some multiple of five bytes, is converted to a 0000 display coded end-of-line byte.

Peculiarities for coded tape for the 64-character set:

	OUTPUT		INPUT		
	Display Code	Internal BCD	External BCD	Internal BCD	Display Code
	00	16	16	16	00
	33	00	12	00	33
	63	12	12	00	33
Line Terminator	0000	1672	1632	1672	0000

Display code 00 is not a valid character; display code 63 (colon) is lost. Line terminators (byte of all zeros in lowest byte of a central memory word) will not result in the loss of zeros.

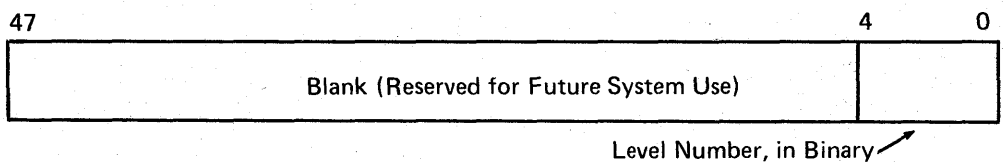
Peculiarities for coded tape for the 64-character set:

	OUTPUT		INPUT		
	Display Code	Internal BCD	External BCD	Internal BCD	Display Code
	00	12	12	12	33
	33	00	12	12	33
	63	16	16	16	63
Line Terminator	0000	1672	1632	1672	0000

Display code 00 (colon) is lost; display code 63 is now a valid character. An exception exists when up to nine 0 characters precede a line terminator. They are changed in the PP buffer to 63B. On tape, they result in external BCD 16. When tape is read, a 63 preceding a line terminator is converted to display code zero. This substitution ensures preservation of all zeros preceding a line terminator, regardless of the graphic character set used.

Appendix A contains the conversion tables for these codes. Conversion is performed by a 6684 if it is part of the hardware configuration.

The system-logical-record terminator on 7-track coded tape is eight characters long. Its format in external BCD is:

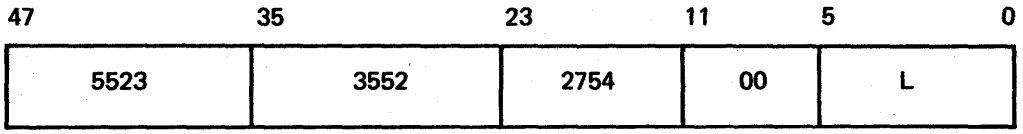


The level number is the low-order 5 bits of the last character. The upper 2 bits of this character are always zero except for level zero which is represented by 010000 (binary). For example, in external BCD, level 5 would be represented by 2020202020202005 and level 0 would be represented by 2020202020202020.

A record terminator marker is appended to the record data, if possible, or written as the only information in the following tape block.

7-TRACK TAPE BINARY SI FORMAT

The system-logical-record terminator on 7-track binary tape is 48 bits long. Its format is:



The marker immediately follows record data if it can be contained within the tape block; otherwise, it is written as the only information in the following tape block.

9-TRACK TAPE CODED OR BINARY SI FORMAT

When SI format 9-track tapes are written or read, information is not converted by the system. All 60-bits of a word in central memory are written to the tape; each pair of 12-bit bytes is written as three 8-bit characters. Conversely, each three 8-bit characters on a tape are written as two 12-bytes when the tape is read. Partial central memory words cannot be read or written on SI tapes. SI tapes can be written or read in packed mode.

The system-logical record terminator has the same format as that for 7-track coded tapes.

Table D-1 summarizes tape file characteristics.

TABLE D-1. TAPE FILE CHARACTERISTICS

Track/Density	Mode	Type	Tape Parity	Maximum Block Size	Data Format on Tape	Noise Size*	End-of-Record	End-of-File	End-of-Information
9 TRACK	Binary	SI	Odd	5120 characters	Packed§	⩽6	†	††	TM EOF1 TM TM
HD=800 bpi	Binary	S	Odd	5120 characters	Packed§	⩽6	Interblock gap (IBG)	Tape mark(TM)	†††
GE=6250 cpi									
PE=1600 cpi	Binary	L	Odd	No maximum**	Packed§	⩽6	IBG	TM	†††
Hardware selects density on read.	Coded	SI	Odd	1280 characters	Packed§	⩽6	†	††	TM EOF1 TM TM
	Coded	S	Odd	5120 Characters	EBCDIC or ASCII conversion	⩽6	IBG	TM	†††
	Coded	L	Odd	No maximum**	EBCDIC or ASCII conversion	⩽6	IBG	TM	†††
7 TRACK	Binary	SI	Odd	5120 characters	No conversion	⩽6	†	††	TM EOF1 TM TM
LO=200 bpi	Binary	S	Odd	5120 characters	No conversion	⩽6	IBG	TM	†††
HI=556 bpi	Binary	L	Odd	No maximum**	No conversion	⩽6	IBG	TM	†††
HY=800 bpi	Coded	SI	Even	1280 characters	External BCD	⩽6	†	††	TM EOF1 TM TM
Hardware can read short records at wrong density.	Coded	S	Even	5120 characters	External BCD	⩽6	IBG	TM	†††
	Coded	L	Even	No maximum**	External BCD	⩽6	IBG	TM	†††

§ Packed means four 6-bit characters in memory are changed to or from three 8-bit characters on tape.

† Short or zero length PRU with a 48-bit marker containing a level number $\leq$ 16B.

†† Zero length PRU with a 48-bit marker containing a level number of 17B.

††† For unlabeled tapes: on a write, 4 tape marks; on a read, undefined so that the user must determine. For labeled tapes: TM EOF1 TM TM.

* May be changed by installation option. Defined in 6-bit characters.

** Maximum size on read (except READSKP) or write is determined by size of user data buffer.

CYBER 170 MODEL 176 DIFFERENCES E

Major hardware differences between CYBER 170 Model 176 and other CYBER 170 models are as follows:

CYBER 170 Model 176 extended memory is analogous to the CYBER 70 Model 76 large central memory (LCM) or large central memory extended (LCME). Extended memory cannot be shared between mainframes and does not have a distributive data path (DDP) access. Shared mass storage (not 819 disk) and coupler linkage multimainframe (MMF) modes are supported; the ECS MMF link is not supported. The maximum extended memory block copy size is 1023 decimal words.

The instruction word stack has a 2-word read-ahead and is not voided by a jump out of the stack or 02 (JP) instruction. When instructions are modified, a return jump is required to void the stack before the modified instructions are executed.

Because of these differences, products have been modified to execute and compile based on a MODEL=176 value in the MODEL micro (refer to the NOS/BE Installation Handbook). Binaries generated by other model settings will not necessarily run under the new models and vice versa.

CYBER 170 Model 176 is not compatible with the CYBER 170 Model 175 in the following ways:

Model 176 systems always execute in CEJ/MEJ mode; the switch, if present, has no effect.

Model 176 peripheral processor subsystem (PPS) can cause exchange jumps in the CPU only when the monitor flag is clear.

Model 176 instruction word stack (IWS) is not degradable.

Model 176 CPU has an instruction word step mode.

Model 176 02 instruction does not void the IWS.

Model 176 jump out of stack does not void the IWS.

CYBER 70 Model 76 LCM/LCME memory replaces ECS as extended memory on CYBER 170 Model 176. The maximum number of 60-bit words that can be transferred in the block copy instruction is 1023 decimal. CYBER 170 Model 176 does not support flag register operations. Extended memory has single error correction/ double error detection (SEC/DED).

The 011, 012, and 013 instructions are legal on a model 176 in any word parcel. NOS/BE forces a half exit, as performed on model 175 if the instruction is in the upper word parcel.

The 014-017 instructions are legal on model 176.

The 464-467 instructions are legal no-op instructions on model 176

30-bit instructions in parcel 3 are legal on model 176.

Model 176 shift unit tests bits 6 through 11 of Bj to determine if the shift count exceeds 63 decimal.

Model 176 shift unit returns negative zero when a negative number is right shifted by more than 63 decimal places.

Model 176 divide unit enters a 4000. . . . pattern below the least significant bit of the dividend on round operations. Overflow or underflow on exponent subtract returns overflow or underflow.

Model 176 floating add unit returns a positive zero if the shift count exceeds 128.

Model 176 branch instructions sense infinite and indefinite as out of range.

A central exchange (013 instruction) exchanges to RAS + K on a model 176.

Model 176 CPU has no breakpoint capability.

INDEX

COMMENT SHEET

MANUAL TITLE CDC NOS/BE 1 Reference Manual

PUBLICATION NO. 60493800 REVISION D

FROM: NAME:

BUSINESS ADDRESS:

CUT ALONG LINE

AA3419 REV. 7/75 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

OLD

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
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

OLD

FOLD

MACRO INDEX

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION