60493800

CONTROL DATA CORPORATION

NOS/BE VERSION 1 REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
CYBER 170 SERIES
CYBER 70
MODELS 71, 72, 73, 74
6000 SERIES

CONTROL STATEMENT INDEX

60493800

CONTROL DATA CORPORATION

NOS/BE VERSION 1 REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
CYBER 170 SERIES
CYBER 70
MODELS 71, 72, 73, 74
6000 SERIES

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.
E	Updated to reflect NOS/BE 1.3 at PSR level 473 and to make editorial and technical corrections.
(6-13-78)	Support of permanent file utilities PFLOG and GENLDPF, GETACT macro, user capability to
	assign universal password and permissions to private sets, user reprieve processing, schedule-by-density
	option for tapes, hardware GE write error correction option, 677/679 tape units, and INTERCOM 5
	is also included. This edition obsoletes all previous editions.
F	Updated to reflect NOS/BE 1.3 at PSR level 481 and to make editorial and technical corrections.
(10-13-78)	The REQUEST control statement and the FILINFO macro have been modified.
G	Updated to reflect NOS/BE 1.3 at PSR level 488 and to make editorial and technical clarifications.
(2-16-79)	New features documented include the following: added formats for the REDUCE and RFL control
	statements for use with ECS; new parameters on the GETPF, SAVEPF, and PURGE control
	statements; system ability to swap ECS.
Publication No. 60493800	

REVISION LETTERS I, O, Q, S, X, AND Z 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.

REVISION RECORD (Cont'd)

REVISION	DESCRIPTION
H	Updated to reflect NOS/BE 1.4 at PSR Level 508 and to make editorial and technical
(12-21-79)	corrections. Support of 885 disk drives is included.
J	Updated to reflect NOS/BE 1.4 at PSR level 518, and to make editorial and technical
(5-19-80)	corrections. COPYL, COPYLM, and ITEMIZE utilities formerly in the Common Utilities
	Reference Manual are included in this revision, as are expanded descriptions of
	permanent file macros formerly in the NOS/BE System Programmer's Reference Manual.
	Interpretive mode processing of ECS read and write operations is also included in this
	revision.
K	Updated to reflect NOS/BE 1.4 at PSR level 530 and to make editorial and technical
(11-17-80)	corrections.
L	Updated to reflect NOS/BE 1.5 at PSR level 538 and to make editorial and technical
(4-20-81)	corrections. Support of EOV2 tape label and a revised CDC CYBER control language
	section are also included. This edition obsoletes all previous editions.
M	Updated to reflect NOS/BE 1.5 at PSR Level 552 and to make editorial and technical
(11-23-81)	corrections. The new feature is interactive CCL procedures. All references to
	INTERCOM 4 have been deleted.
Publication No. 60493800	

LIST OF EFFECTIVE PAGES

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

PAGE	REV
Front Cover	-
Inside Front Cover	M
Title Page	-
ii	L
ii-a/ii-b	M
iii	M
iv	M
v	L
vi	L
vii	L
viii	M
ix	M
x	M
1-1	E
1-2	E
1-3	A
1-4	H
1-5	K
1-6	H
1-7	K
1-8	A
1-9	E
1-10	E
1-11	C
1-12	A
1-13	J
2-1	A
2-2	E
2-3	E
2-4	F
2-5	H
2-6	L
2-7	K
2-8	L
2-9	A
2-10	H
2-11	K
2-12	L
2-13	L
2-14	E
2-15	E
2-16	F
2-17	E
2-18	L
3-1	L
3-2	E
3-3	F
3-4	J
3-5	E
3-6	J
3-7	E
3-8	L
3-9	F
3-10	L
3-11	L
3-12	E
3-13	M
3-14	M

PAGE	REV
3-15	M
3-16	M
3-17	L
3-18	L
3-19	L
3-20	L
3-21	L
3-22	M
3-23	M
3-24	L
3-25	L
3-26	L
3-27	L
3-28	L
3-29	L
3-30	L
3-31	M
3-32	L
3-33	M
3-34	L
3-35	M
3-36	M
3-37	L
3-38	L
3-39	L
3-40	L
3-41	L
3-42	L
3-43	L
3-44	L
4-1	L
4-2	E
4-3	J
4-4	J
4-5	E
4-6	M
4-7	M
4-8	H
4-9	G
4-10	H
4-11	E
4-12	K
4-13	M
4-14	E
4-15	M
4-16	E
4-17	E
4-18	F
4-19	F
4-20	E
4-21	M
4-22	L
4-23	L
4-24	M
4-25	L
4-26	L
4-27	L
4-28	L
4-29	L

PAGE	REV
4-30	L
4-31	L
4-32	L
4-33	L
4-34	L
4-35	L
4-36	L
4-37	L
4-38	L
4-39	L
4-40	L
4-41	L
4-42	L
4-43	L
4-44	L
4-45	L
4-46	L
4-47	M
4-48	L
4-49	L
4-50	L
4-51	L
4-52	L
4-53	L
4-54	L
4-55	L
4-56	L
4-57	L
4-58	L
4-59	L
4-60	L
4-61	M
4-62	L
4-63	L
4-64	L
4-65	M
4-66	L
4-67	L
4-68	M
4-69	L
4-70	L
4-71	L
4-72	L
4-73	L
4-74	L
4-75	M
4-76	M
4-77	L
4-78	L
4-79	L
4-80	L
4-81	L
4-82	L
4-83	L
4-84	L
4-85	L
4-86	M
4-87	L
4-88	L

PAGE	REV
4-89	L
4-90	L
4-91	L
4-92	L
4-93	L
4-94	L
4-95	M
4-96	M
4-97	L
4-98	L
4-99	L
4-100	L
4-101	L
4-102	L
4-103	L
4-104	L
4-105	L
4-106	M
4-107	L
4-108	L
4-109	L
4-110	L
4-111	L
4-112	L
5-1	L
5-2	M
5-3	M
5-4	M
5-5	M
5-6	M
5-7	M
5-8	M
5-9	M
5-10	M
5-11	M
5-12	M
5-13	M
5-14	M
5-15	M
5-16	M
5-17	M
5-18	M
5-19	M
5-20	M
5-21	M
5-22	M
5-23	M
5-24	M
5-25	M
5-26	M
5-27	M
5-28	M
5-29	M
5-30	M
5-31	M
5-32	M
5-33	M
5-34	M
5-35	M

PAGE	REV
5-36	M
5-37	M
5-38	M
5-39	M
5-40	M
5-41	M
5-42	M
5-43	M
5-44	M
5-45	M
5-46	M
5-47	M
5-48	M
5-49	M
5-50	M
5-51	M
5-52	M
5-53	M
5-54	M
5-55	M
5-56	M
5-57	M
5-58	M
5-59	M
5-60	M
5-61	M
5-62	M
6-1	C
6-2	L
6-3	H
6-4	E
6-5	H
6-6	E
6-7	H
6-8	E
6-9	J
6-10	E
6-11	H
6-12	C
6-13	H
6-14	K
6-15	L
6-16	L
6-17	L
6-18	L
6-19	L
6-20	L
6-21	L
6-22	L
6-23	L
6-24	L
6-25	L
6-26	L
6-27	L
6-28	L
7-1	J
7-2	L
7-3	L
7-4	L

7-5	L
7-6	H
7-7	E
7-8	L
7-9	H
7-10	F
7-11	L
7-12	M
7-13	L
7-14	L
7-15	L
7-16	L
7-17	L
7-18	M
7-19	L
7-20	L
7-21	L
7-22	L
7-23	L
7-24	L
7-25	L
7-26	L
7-27	L
7-28	L
7-29	L
7-30	L
7-31	L
7-32	L
7-33	L
7-34	L
7-35	L
7-36	L
7-37	L
7-38	L
7-39	L
7-40	L
7-41	L
7-42	L
7-43	L
7-44	L
7-45	L
7-46	L
7-47	M
7-48	L
7-49	L
7-50	L
7-51	L
7-52	L
7-53	L
7-54	L
7-55	L
7-56	L
7-57	L
7-58	L
7-59	L
7-60	L
7-61	L
7-62	L
7-63	L
7-64	L
7-65	L
7-66	L
7-67	L
7-68	L
7-69	L
7-70	L
7-71	L
7-72	L
7-73	L
7-74	L
7-75	L
7-76	E
7-77	H
7-78	J
7-79	L
7-80	L
7-81	L
7-82	L
7-83	K
7-84	K
7-85	K
7-86	L
7-87	M
7-88	M
7-89	L
7-90	L
7-91	L
7-92	L
7-93	L
7-94	L
7-95	M
7-96	L
7-97	L
7-98	L
A-1	K
A-2	K
A-3	L
A-3	L
A-4	L
A-5	L
A-6	L
A-7	L
A-8	L
A-9	L
A-10	L
B-1	L
B-2	L
B-3	E
B-4	L
B-5	H
B-6	H
B-7	H
B-8	E
B-9	E
C-1	E
C-2	E
C-3	H
C-4	E
C-5	H
C-6	M
C-7	M
D-1	E
D-2	E
E-1	J
F-1	M
F-2	M
Index-1	M
Index-2	M
Index-3	M
Index-4	M
Index-5	M
Index-6	M
Index-7	M
Comment Sheet	M
Inside Back Cover	M
Back Cover	-

PREFACE

This manual describes the Network Operating System/Batch Environment (NOS/BE) Version 1.5 Operating System for the CONTROL DATA® CYBER 70 Models 71, 72, 73, 74; CDC®CYBER 170 Series; and CDC 6000 Series Computer Systems. It contains general information about files, job flow, and execution; it gives detailed descriptions of the full array of control statements available. Sections 1 through 5 are intended for application programmers who write in higher level languages; sections 6 and 7 are for system programmers and others who write in COMPASS assembly language.

It is assumed the user of this manual has a basic familiarity with the NOS/BE operating system. The user who is unfamiliar with this system, or operating systems in general, should first study the NOS/BE Version 1 Batch User's Guide.

Extended memory for the CYBER 170 Model 176 is large central memory extended (LCME). Extended memory for all other NOS/BE computer systems is extended core storage (ECS) or extended semiconductor memory (ESM).

In this manual, the acronym ECS refers to all forms of extended memory unless otherwise noted.

Programming information for the various forms of extended memory can be found in the COMPASS Reference Manual and in the appropriate computer system hardware reference manual.

CONVENTIONS

Conventions for central memory word formats are as follows:

- Crosshatching indicates a field is not used by or is not applicable to a function processor. However, Control Data reserves the right to assign these fields to system use in the future.
- Fields reserved for system use are so labeled.
- Fields with numeric identifiers indicate the actual value that is used or returned for a particular function. Numeric identifiers are octal unless otherwise noted.

RELATED PUBLICATIONS

The following manuals contain additional information about NOS/BE that may prove useful to the system user.

The NOS/BE Manual Abstracts is a pocket-sized manual containing brief descriptions of the contents and intended audience of all NOS/BE and NOS/BE product manuals. The abstracts can be useful in determining which manuals are of greatest interest to a particular user.

Control Data also publishes a Software Publications Release History of all software manuals and revision packets it has issued. This history lists the revision level of a particular manual that corresponds to the level of software installed at the site.

Control Data Publication	Publication Number
CYBER Loader Reference Manual	60429800
CYBER Record Manager Advanced Access Methods Version 2	60499300
CYBER Record Manager Basic Access Methods Version 1.5	60495700
INTERCOM Version 4 Multi-User Job Capability	60494700
INTERCOM Version 4 Reference Manual	60494600
INTERCOM Version 5 Multi-User Job Capability Reference Manual	60456070
INTERCOM Version 5 Reference Manual	60455010
NOS/BE Manual Abstracts	84000470
NOS/BE Version 1 Batch User's Guide	60494000
NOS/BE Version 1 Diagnositc Handbook	60494400
NOS/BE Version 1 Diagnostic Index	60456490
NOS/BE Version 1 Installation Handbook	60494300
NOS/BE Version 1 Operator's Guide	60493900
NOS/BE Version 1 System Programmer's Reference Manual, Volume 1	60494100
NOS/BE Version 1 System Programmer's Reference Manual, Volume 2	60457370
On-Line Maintenance Software Reference Manual	60453900
SCOPE Version 2 Operator's Guide	60455090
Software Publications Release History	60481000
Update Reference Manual	60449900

DISCLAIMER

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

NOS/BE is the operating system for the CDC 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 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 operating system. All products run under the control of the operating system.

NOS/BE is a multi-programming, multi-processing 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 CDC 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 CDC CYBER 170; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems have the following 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 (ECS)

Graphics terminals and plotters

Different types of line printers and magnetic tape units

All of the previously mentioned hardware usually resides at a central site. However, the CDC CYBER hardware and NOS/BE 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 CDC CYBER 170; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer System 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 65K, 98K, or 131K words. Memory sizes of 198K or 262K are available with the CDC CYBER 170 series.

A CDC 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 is a multi-programming 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.

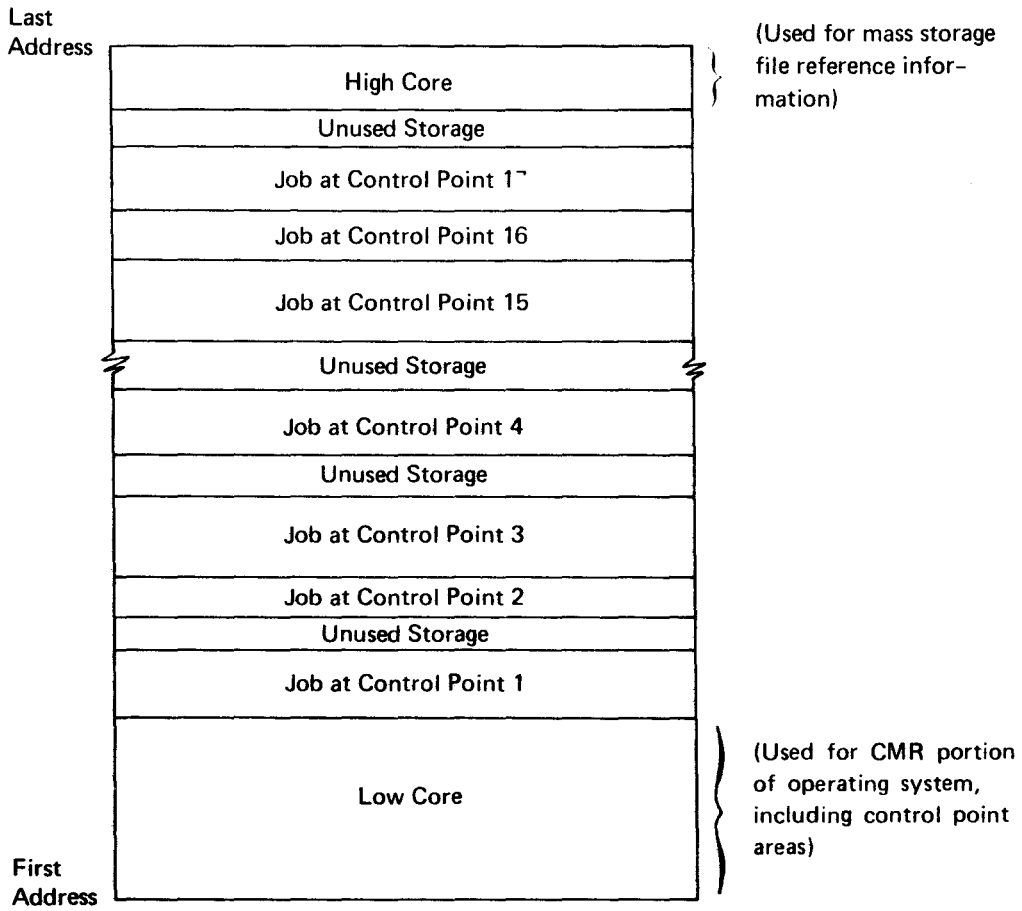


Figure 1-1. Central Memory Allocation

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, the monitor routine (MTR) suspends all user program activity at the control point, waits for all peripheral processors (PPs) 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 is not affected by this change in location. Since all program locations are relative to the beginning of the job field length, only the reference address (RA) 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. The job moves into and out of memory in relation to its needs for system resources, such as tapes or the central processor, 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 memory (CM and/or ECS) 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 a 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, 173, 720, 730, and 740; 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 CDC 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 CDC CYBER 170 Models 175, 176, 750, and 760; CYBER 70 Model 74-1x; and the 6600 Computer Systems have a single processor composed of 9 or 10 arithmetic 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. (Refer to appendix D for a more detailed discussion of CDC CYBER 170 Model 176 differences.)

The CDC 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 PROCESSORS

Peripheral processors (PPs) are small computers with 4096 12-bit words of memory. Any given system might have 7 to 20 peripheral processors. PPs are independent computers; they all can simultaneously process programs. In addition, a CDC CYBER Model 176 can have up to six first-level peripheral processors (PPUs) that are used to transfer data to mass storage.

One of the purposes of the PPs 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 PP. 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.

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, and 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 produced 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 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 (bpi).

A 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

ECS is a second, supplementary form of memory that has two main uses. It can be 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.

CDC CYBER 170 Model 176 systems have a form of extended memory different than other CDC CYBER 170 models but functionally similar. The CDC 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 D of this manual. References to ECS in the remainder of this document apply to extended memory of all CDC CYBER 170 Models except as limited by the CDC CYBER 170 Model 176 differences described in appendix D.

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 cannot 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 in the following paragraphs.

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.

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; some 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.

CDC CYBER RECORD MANAGER

CDC 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 CDC CYBER Record Manager manuals.

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

CDC 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 CDC CYBER Record Manager functions often result in an identical sequential file.

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

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 statement 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

CDC CYBER LOADER

CDC 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. CDC CYBER Loader is a common product described in full in the CYBER Loader Reference Manual.

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 (figure 2-1) 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.†

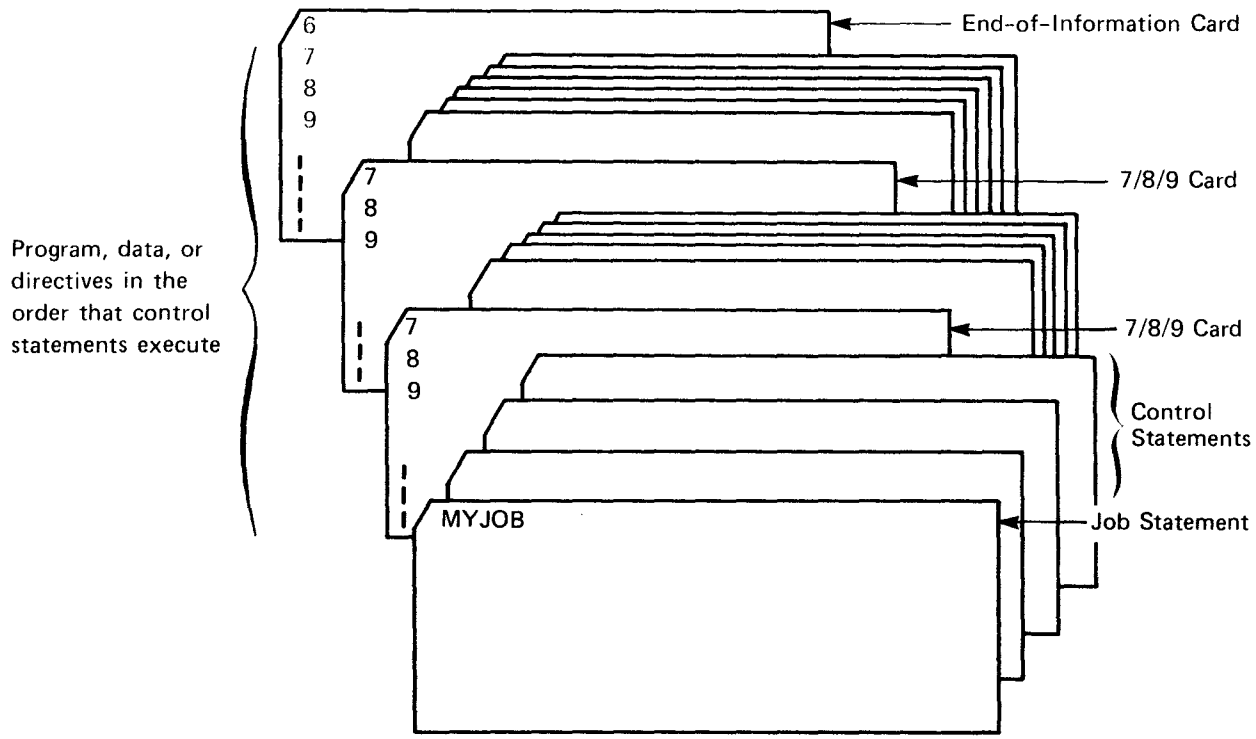


Figure 2-1. Sample Deck Structure

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 the INTERCOM EDITOR, 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
```

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
6/7/8/9
```

The preceding two decks 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 (EOI) 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 an end-of-partition (EOP) or an end-of-file (EOF).

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. JANUS, the system routine that controls standard unit record equipment, converts a 7/8/9 level 17_8 card to the equivalent of a 6/7/8/9 end-of-information card.

Separator cards can be used to indicate whether the cards following them are punched in O26 or O29 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 CDC CYBER 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 in the following paragraphs.

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 attach the file. 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 7.

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 CDC 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.

Global libraries

Local 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 CDC CYBER 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.
SEGLOAD	Specifies segmentation, dividing large programs into sections.

Refer to the CYBER 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 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. (Refer to the CYBER 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:

```
FTN5,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. For instance, the FORTRAN compiler 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

The following names should be used on the program execution call statement to assemble or compile a user program.

Source Language	lfn	Source Language	lfn
FORTRAN Version 5	FTN5.	SYMPL	SYMPL.
FORTRAN Extended Version 4	FTN.	Sort/Merge	SORTMRG.
COBOL Version 5	COBOL5.	PERT/TIME	PERT66.
COBOL Version 4	COBOL.	APT	APT.
ALGOL	ALGOL.	QUERY UPDATE Version 2†	QU.
ALGOL Editor	ALGEDIT.	QUERY UPDATE Version 3†	QU.
COMPASS	COMPASS.	FORM	FORM.
SIMSCRIPT	SIMS.	Data Definition Language 2	DDL.
BASIC	BASIC.	Data Definition Language 3	DDL3.

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 FTN5)

Parameters for many products are the default I=INPUT, B=LGO, and L=OUTPUT. Refer to 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 unprocessed section of the job deck must contain the program.

†Only one version is active on a system. The call is the same regardless of the version.

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 following examples presume a job compiles a FORTRAN 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 two tape units required.
REQUEST,DATA,MT.  ASSIGN 3456.
REQUEST,DATA2,MT.   ASSIGN 3457.
FTN5.
LGO.
LGO.
```

The same operations performed more efficiently are:

```
EFFICIENT,MT1.
FTN5.
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 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 following options.

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

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 is 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 also be rolled out, 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 that follows, the tape has a label containing 1972 as the volume serial number. The job would be structured as illustrated in figure 2-2.

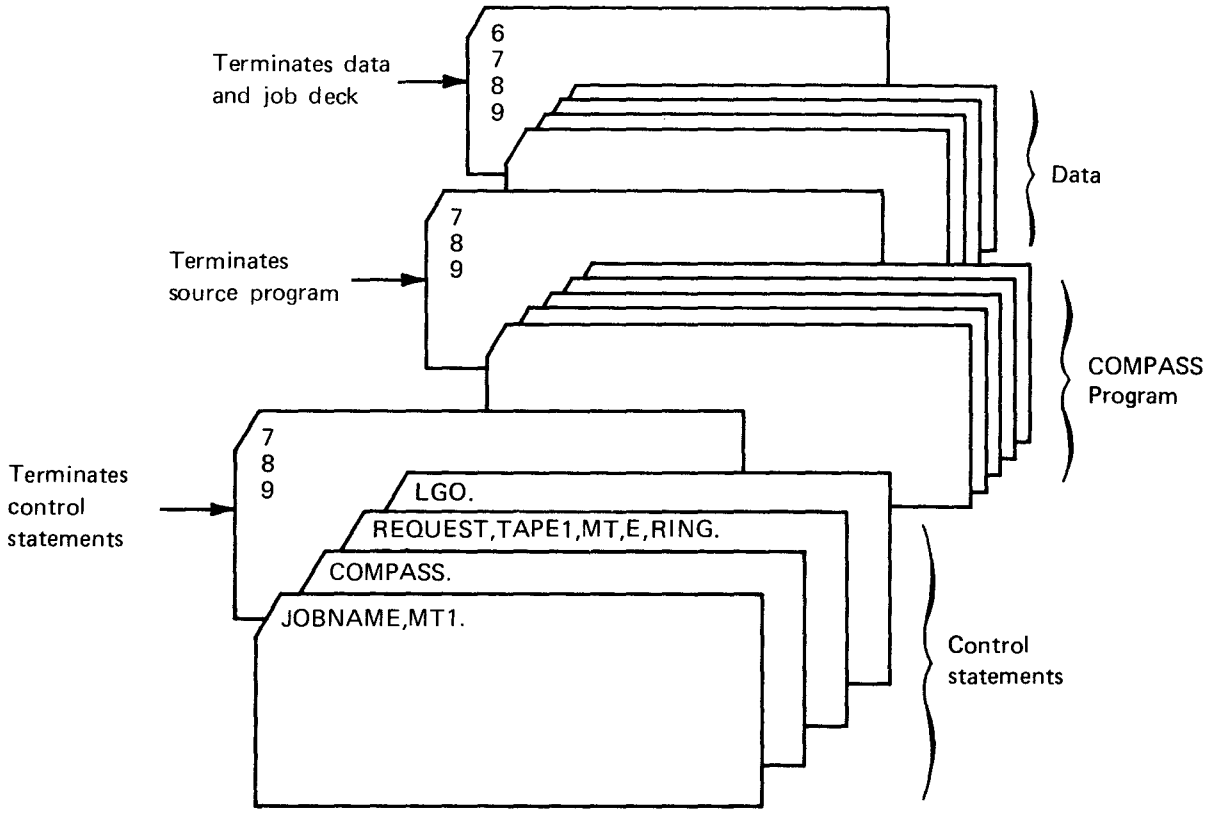


Figure 2-2. Sample COMPASS Job

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).

When the job is in the input queue of jobs awaiting execution, it comes under control of a 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, and 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. The system 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.

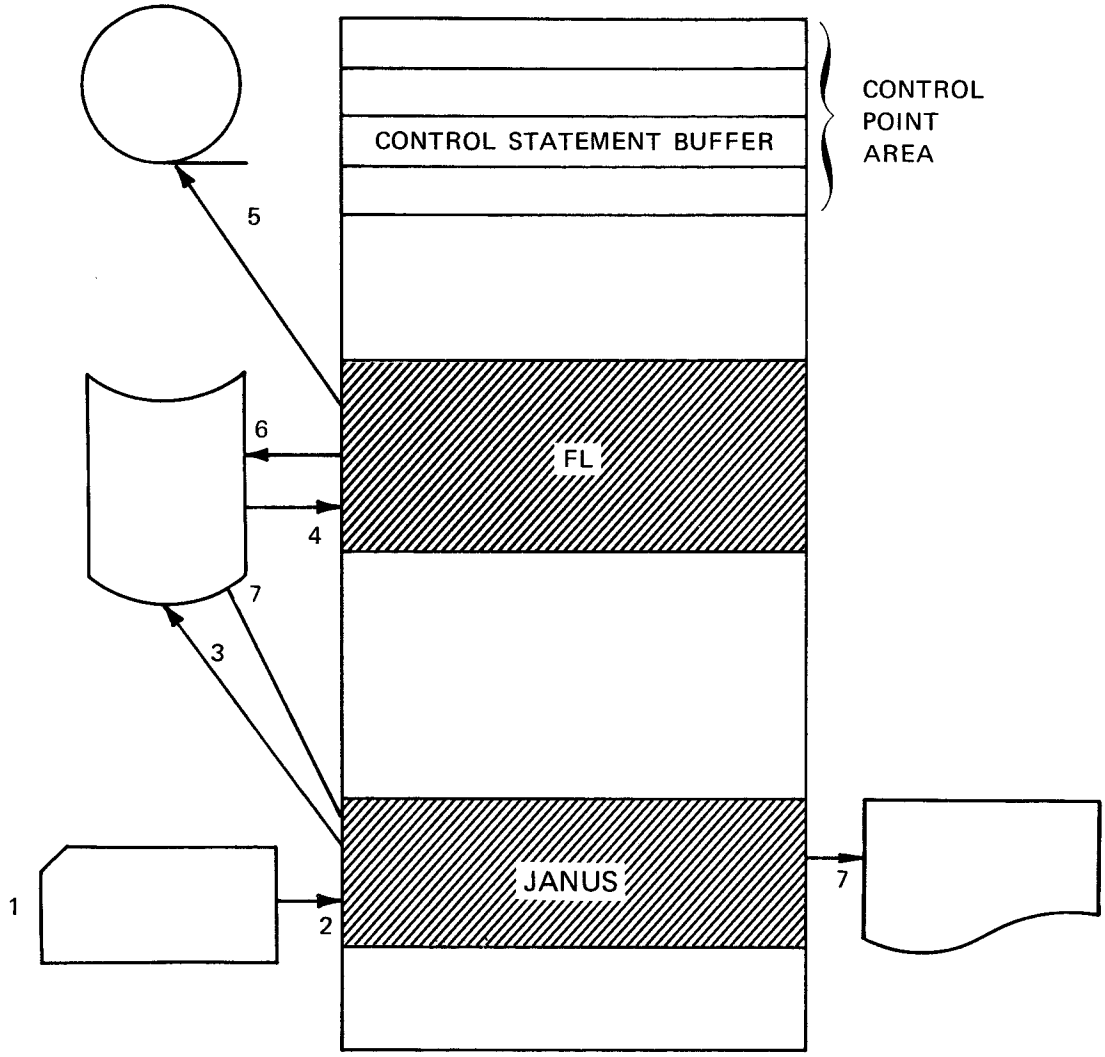
After accounting processing, the system copies the BATCH system bulletin to the job OUTPUT file. If the installation has not specified BATCH system bulletin information, no information is written to the OUTPUT file. The installation can specify other standard procedures to be executed at this time.

Upon completion of all standard procedures, job control is advanced to the second statement, COMPASS, which directs assembly of the user's program. The system 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. However, 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 then advances to the REQUEST statement. The VSN parameter provides the volume serial number for the tape label. The system automatically assigns the tape if it is mounted. (If the installation does not choose the automatic assignment feature, 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 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 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. A generalized description of the job flow is shown in figure 2-3.



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 2-3. Job Flow at Central Site

EXAMPLES OF JOB DECK ARRANGEMENTS

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

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

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

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

```
JOBB.
FTN5.
LGO.
7/8/9
FORTRAN Program
6/7/8/9
```

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

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

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

```
JOBD.
FTN5.
LGO.
LGO.
7/8/9
FORTRAN Program
7/8/9
First Data record
7/8/9
Second Data record
6/7/8/9
```

5. 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.

```
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
```

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 following files 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; 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.

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 previously described for normal job termination.

†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.

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, earlier in this section.

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 preoutput 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, rename, or extend of a permanent file, modified a permanent file, or 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 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 2-4 shows a typical dayfile.

```
  mfi        system 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.CP8      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 2-4. Sample Dayfile

The system header identifies the system on which the job executed. Installations might change the information given on this line.

mfi	Mainframe identifier.
system level	Operating system level.
mm/dd/yy	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 origin in the following format.

jobname FROM s s s / t t

jobname	Unique name assigned by the system.
s s s	Source mainframe ID (blank if sss is the same as mfi).
t t	Terminal ID (blank unless the job was sent from an INTERCOM terminal).

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

IP nnnnnnnn WORDS – FILE lfn, DC dc
or
OP nnnnnnnn WORDS – FILE lfn, DC dc

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.
lfn	Logical file name.
dc	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. Figure 2-5 shows sample accounting messages.

```
MS  aaaaaaaa WORDS (bbbbbbbb MAX WORDS USED)
CPAcccccccc.ccc SEC. dddddddd.ddd ADJ.
CPBcccccccc.ccc SEC. dddddddd.ddd ADJ.
IOeeeeeeee.eee SEC. ffffffff.fff ADJ.
CMgggggggg.ggg KWS. hhhhhhhh.hhh ADJ.
ECiiiiiiii.iii KWS. jjjjjjjj.jjj ADJ.
SS                  kkkkkkkk.kkk ADJ.
PPmmmmmmmm.mmm SEC.      DATE mm/dd/yy
```

Figure 2-5. Sample Accounting Messages

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.
dddddddd.ddd	Adjusted central processor time for each processor. The time is multiplied by an installation selected weighting constant.
eeeeeeee.eee	Input/output time.
ffffffff.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 I/O time are weighted, to compensate for overlapped I/O 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, I/O time, central memory kilo-word seconds, and ECS kilo-word seconds.
mmmmmmmm.mmm	Peripheral processor time.
mm/dd/yy	Date job was run.

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 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 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 file name. The operating system associates two files with each job, one with the file name INPUT and another with the file name OUTPUT. All other file names must be specified by the job. The 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 file name must be unique within a job and must not duplicate the name of a multi-file tape set associated with the job. File names are one through seven letters or digits and must begin with a letter.

RESERVED FILE NAMES

File names that begin with ZZ are reserved for use by the system. User jobs are not prevented from creating or reading files with the name ZZxxxxx, 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 following paragraphs contain descriptions of some of these files.

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 or ROUTE control statement. However, 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 private 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 with a CATALOG statement. Such 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 the following.

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 are assigned by the operating system to a mass storage device designated for input and output queue files.

DISPOSING OF FILES AND EQUIPMENT

Temporary or permanent status 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 described in DISPOSE and ROUTE control statements and macros, and Special-Named Files.

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. All files currently associated with the job are called local files. 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 file name and the resident device available for other uses. 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 scope internal (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 terms used throughout this manual, such as:

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

Level terminators

Physical record units

Partitions

SYSTEM-LOGICAL-RECORDS AND PHYSICAL RECORD UNITS

A physical record unit (PRU) 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_8 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_8 establishes a partition boundary for the file. Level 17_8 always is recorded in a zero-length PRU. Level 17_8 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_8 terminator marks the logical end of the file. However, a file can contain any number of partitions defined by level 17_8 before the physical end of the file.

The following lists 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_8.	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_8.	Partition; sets end-of-partition bits in system tables; end-of-file exits occur.
End of mass storage allocated in system record block table (RBT).	End-of-information; sets end-of-information bits, if any, in system tables or 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_8 has special meaning to the checkpoint/restart feature of the operating system. Consequently, level 16_8 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 CYBER Record Manager manuals.

FILE DIVISIONS

The physical representation of beginning-of-information and end-of-information depends on the storage device as follows:

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_8. 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_8 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 previous divisions. Individual products that are supported by the operating system have different definitions of the term record. For instance, CDC CYBER Record Manager defines eight types of records, only one of which (S type) is equivalent to a system-logical-record. CDC 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:

Attribute	Significance
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 that can hold permanent files.
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 encountered.

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

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.

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

2. 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, the maximum number of permanent files that can exist in the set, the universal password, the universal permissions, the public password, and the default file retention period for this 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.

3. Members of the device set are added by additional ADDSET control statements that specify the device set name, the master device volume serial number (VSN), 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.

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.

1. 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.

2. 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.

3. 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 EXAMPLES

1. NEW DEVICE.

```
LABELMS(DT=AY)   PLEASE USE PACK 844A
LABELMS.         PLEASE USE PACK 844B
ADDSET(VSN=844A,MP=844A,SN=MORE,*PF,UV=MYUNIV,UP=C,PB=MYPUBLIC,FR=360)
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)
FTN5.
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 jobs makes a permanent file on the device set MORE.

5. USE A SET.

```
MOUNT(VSN=844A,SN=MORE)                  Mounts the master device.
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.
FTN5.
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)	Mounts master device.
MOUNT(VSN=844A,SN=MORE)	Mounts master device.
SETNAME(MORE)	
ATTACH(TAPE5,PERMFILE,ID=FRIEND)	File is taken from device set MORE because of preceding SETNAME.
REQUEST(A,PF,SN=OTHER)	File directed to device set OTHER since explicitly requested.
COPY(TAPE5,A)	
FTN5.	
LGO.	FORTRAN job creates file TAPE6 on system device as no REQUEST card used.
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	

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. Refer to Record Request/Return Information of the FET in section 6 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 sections 6 and 7 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 various languages, see the reference manuals for those products or languages.

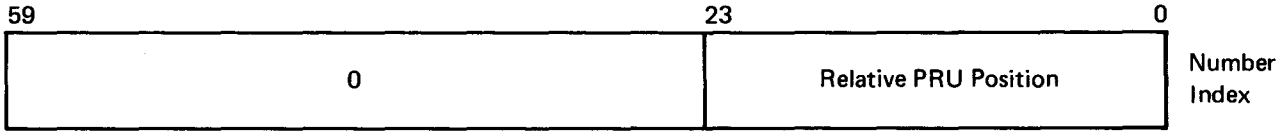
NAME/NUMBER INDEX FILES

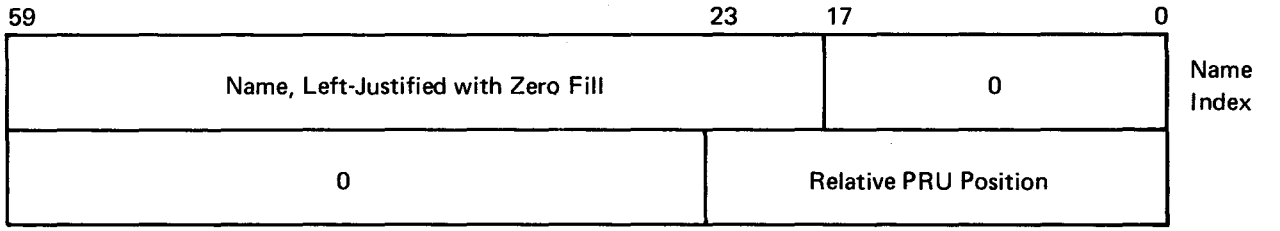
Name/number indexed files can be created, read, written, and rewritten using the COMPASS macros OPEN, CLOSE, READIN, WRITOUT, WRITIN, and WRITER. Management of a single index level is provided through macros OPEN and CLOSE.

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 nonzero 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. Following are the formats of index entries.





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 contains addresses of sub-index records interspersed throughout the file. The master index record is processed by OPEN/CLOSE as is a single-level index record. The user routine needs to ensure that READIN/WRITOUT references 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 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 the permanent file directory and catalog. 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 7.

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 local file name, the permanent file name can serve as both the pfn and the 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 if any of the following occur.

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.

PERMISSIONS AND PASSWORDS

All user files have a 4-bit permission code. Each bit represents an access permission as defined by the following.

Permission	Significance
READ	Required to read, load, or copy a file.
MODIFY	Required to rewrite existing data or to eliminate part of a file.
EXTEND	Required to eliminate part of a file or to increase the amount of mass storage allocated to a particular file.
CONTROL	Required to purge a file, or to 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 turnkey 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 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. On a RENAME, the public password must be specified with the PW parameter to change a permanent file to ID = PUBLIC.

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

The universal password, universal permission, and public password for private device sets are defined on the ADDSET control statement when the master device is created. For public device sets, they are defined by the installation (refer to the NOS/BE Installation Handbook).

The universal password is a string of 1-9 letters or digits. When specified for a function that references a permanent file, such as ATTACH, it grants the universal permission defined for that set. Universal permission is any non-null combination of control, modify, extend, and/or read permissions. The universal password takes precedence over any password defined by CATALOG or RENAME, as explained in the following examples.

PURGE(pfn,ID=id,SN=MYSET,UV=MYUNIVPW)

If the universal password is MYUNIVPW and the universal permission is control permission on device set MYSET, then the universal password can be used to purge any permanent file on MYSET even though a CN= password has been defined to restrict access to that file.

ATTACH(pfn,ID=id,SN=DSET,UV=U)

If the universal password is U and the universal permission is read permission on device set DSET, then the universal password can be used to attach and read any permanent file on DSET even though an RD= password has been defined to restrict access to that file.

The public password is a string of 1-9 letters or digits. On a CATALOG of the initial cycle of a file with ID=PUBLIC, the public password for this device set must be specified using PW=.

MULTIPLE ACCESS

A permanent file can be attached to more than one job at the same time. Many jobs can read a file at the same time, but only one at a time 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.

An alternate method of allowing multiple attaches with read only permission is initially to catalog the file with EX=, MD=, and CN= (or XR=) specified. Subsequent attaches without PW= or MR= specified default to multi-read access.

Table 3-1 lists the cases of multiple access in which access by a second job is either granted or the job is put into a waiting queue. In the latter case, the attach request is not honored until all of the requested permissions can be granted. If the second job is of batch origin and is placed in the waiting queue, a wait message is issued to the job dayfile. If the second job is of INTERCOM origin and is placed in the wait queue, the wait message is issued to the terminal. The user can wait until the attach is honored or bypass the attach by entering %A.

TABLE 3-1. MULTIPLE ACCESS PERMISSIONS

		Second job issues an ATTACH requesting the following permissions:			
		Read	Extend	Modify	Control
First job has file attached with these permissions:	Read	Granted	Granted	Granted	Wait
	Extend	Granted	Wait	Wait	Wait
	Modify	Granted	Wait	Wait	Wait
	Control	Wait	Wait	Wait	Wait

Granted: File is immediately attached to the second job with the requested permission.

Wait: System places the job in the permanent file queue until the ATTACH request can be honored.

QUEUED AND ARCHIVED FILES

Job requests to attach a permanent file usually are executed immediately. If a job cannot attach a file immediately, the system places the request in the permanent file queue. Four conditions can cause a permanent file request to be placed into the permanent file queue.

TRANSPF utility is running.

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 request 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. Therefore, 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.

A request for an 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 the following 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.

1. Enter command to attach file. Wait until WAITING FOR ARCHIVED FILE message appears.
2. Enter abort command.
3. Continue with other operations.
4. 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 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 previous ones.

SAVEPF	Create a permanent file on a public or private 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 3-2 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 3-2. PERMANENT FILE PARAMETERS

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

\+ Required. * Optional. o Ignored with message.

** Optional. If used with ST, both SN and VSN must be specified.

The following 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.
GENLDPF	Generates LOADPF jobs according to the permanent file catalog (PFC) entries on the tape produced by PFLOG.
LOADPF	Loads permanent files that have been dumped by DUMPF.
PFLOG	Dumps the PFC to tape.
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:

By using 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 and 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, 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=vsn,SN=setname

INTERCOM PERMANENT FILE USE EXAMPLES

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 (CR) denotes carriage return.

1. COMMAND-STORE,MYFILE (CR)

   ID=RKC (CR)

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

2. COMMAND-FETCH,MYFILE,RKC (CR)

   COMMAND-DISCARD,MYFILE (CR)

   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:

CATALOG	CT ID=name PFN=pfn CT CY= cy SN=setname n WORDS
EXTEND/ALTER	EX ID=name PFN=pfn EX CY= cy SN=setname n WORDS
PURGE	PR ID=name PFN=pfn PR CY= cy SN=setname n WORDS
RENAME (old permanent file)	NM ID=name PFN=pfn NM CY= cy SN=setname n WORDS
RENAME (new permanent file)	RN ID=name PFN=pfn RN CY= cy SN=setname n WORDS

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

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

EXAMPLES

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

CATALOG EXAMPLES

The first set of examples demonstrates 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 local 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=ART)

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

   These control statements demonstrate that the XR parameter has the same effect as the MD, CN, and EX parameter combination. ART 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 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 file name. If the 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 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 file name, PERMANE. The first seven characters must be letters or numbers and begin with a letter if the file name is omitted in the attach call.

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

   or

   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.

5. ATTACH(WATER,LILLIES,ID=CMONET,MR=1)
   ATTACH(WATER,LILLIES,ID=CMONET)

   Assuming catalog example 4 was successful, these two control statements perform the same function of attaching 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
   ATTACH(LFN,ID=RODIN)
   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 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 permanent file cycle specified on the PURGE control statement was already attached (with a different file name), 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 permanent file 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 more	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 error processing (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 set; otherwise, 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, which 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 C 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.

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 667, 669, 677, and 679 tape drives only)

SI format is assumed unless an F=S or F=L parameter appears in a LABEL control statement or 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.

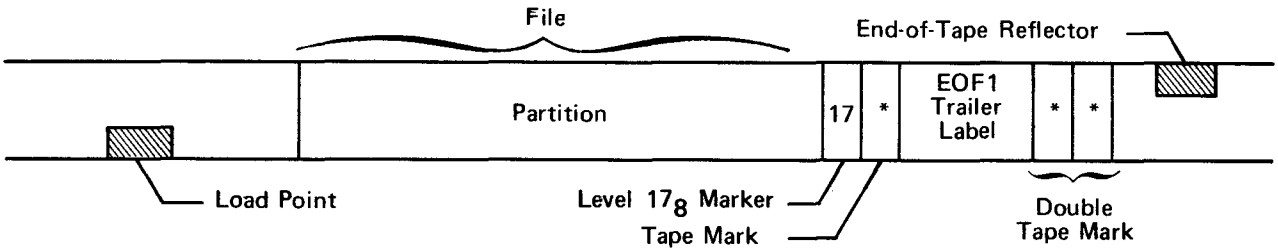
The short or zero-length PRU that terminates a record is less than full PRU size.

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_8 indicates an end-of-partition. Level 17_8 is always written as a zero-length PRU.

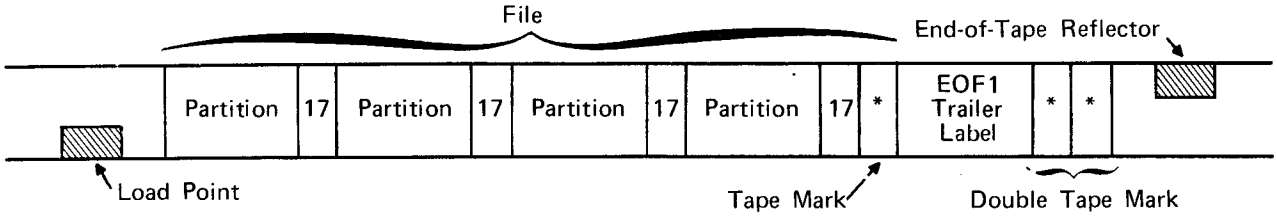
When an output file on an SI tape is closed, the operating system appends up to four items: a level 17_8 zero-length PRU,† a single tape mark, trailer label information for both labeled and unlabeled tapes, and a double tape mark. The file is then positioned to the beginning of the single tape mark. If more information is written to the tape, only the level 17_8 marker indicating an end-of-partition remains. If the tape is 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 as follows:



†The presence of a level 17_8 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_8 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 as follows:



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 three more times, then closes the file.

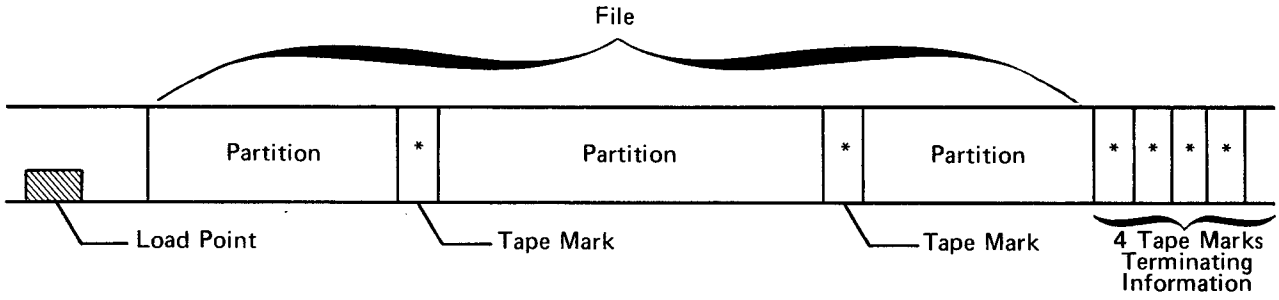
Coded information is written on 7-track SI tape in external BCD format shown in appendix A. On a 9-track SI tape, data is written in packed (binary) mode for both coded and binary data. Only full central memory words can be written or read on 7- or 9-track SI format tapes. (Refer to Tape Format in appendix C.)

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 three partitions written before the file is closed, is as follows:



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 (lfn+6) 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 6 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.

SEVEN-TRACK VERSUS NINE-TRACK TAPES

Both seven-track and nine-track 0.50-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 nine-track L tapes are supported only on 669 and 679 tape drives.

SEVEN-TRACK TAPE

Seven-track tapes are processed by 667 and 677 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 seven-track; LO, HI, or HY provides an implicit definition. On a LABEL statement, seven-track is assumed unless nine-track is specifically declared.

NINE-TRACK TAPE

Nine-track tape is processed on Control Data 669 and 679 tape units. On a REQUEST control statement, an NT parameter explicitly specifies a nine-track tape. On both REQUEST and LABEL control statements, a density specification of HD, PE, or GE implicitly specifies a nine-track tape and the NT parameter can be omitted.

---

†Data cannot be written at 200 bpi on 667 or 677 tape drives although both drives can read 200 bpi tapes.

Under hardware control, nine-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 669 and 679 tape drives
PE	1600 cpi (phase encoded) applies to 669 and 679 tape drives
GE	6250 cpi (group encoded) applies to 679 tape drives

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

When S or L format nine-track tapes are written or read, 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 as follows:

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. The tape label characters are written in 6-bit external BCD on 7-track tapes, and in 8-bit ASCII or EBCDIC on 9-track tapes.

Two types of labels are recognized:

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 overwritten 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.

Multifile 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 the following 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	End-of-volume label	End of volume	Required when appropriate
EOV	3-9	End-of-volume label	End of volume	Optional
UTL	†	User trailer label	End of file	Optional

Table 3-3 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 desired 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 6. 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.

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

TABLE 3-3. 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 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 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 characters	Spaces	Yes
	22-27	4	Set Identification (Multi-File Set Name)	6	Any characters	Volume Serial Number of first volume of set	No
	28-31	5	File Section Number (Volume Number)	4	4 digits indicating number of volume in file	0001	Yes
	32-35	6	File Sequence Number (Position Number)	4	4 digits 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 digits indicating the edition of file	00	Yes
	42-47	9	Creation Date	6	Space followed by 2 digits for year, 3 digits 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 characters	Space	No
	55-60	12	Block Count	6	Zeros	Zeros	Yes
	61-73	13	System Code	13	Any characters	Spaces	No
	74-80	14	Reserved	7	Spaces	Spaces	No

TABLE 3-3. 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 digits: 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.						
Second † End-of-Volume Label	1-3	1	Label Identifier	3	EOV	EOV	Yes
	4	2	Label Number	1	2	2	Yes
	5-10	3	Next VSN	6	6 characters	No default	Yes
Additional End-of-Volume Labels	1-3	1	Label Identifier	3	EOV	EOV	Yes
	4	2	Label Number	1	3-9	3-9	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 characters. Content of these fields is not checked on input; content is written as received from the user.					

†The second end-of-volume label conforms to ANSI standards but is not a standard ANSI label.

STANDARD LABELED TAPE STRUCTURE

The first four ANSI labels are required and the fifth ANSI label is written by the system but is not required to read the file. They 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 multifile 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 multifile set.

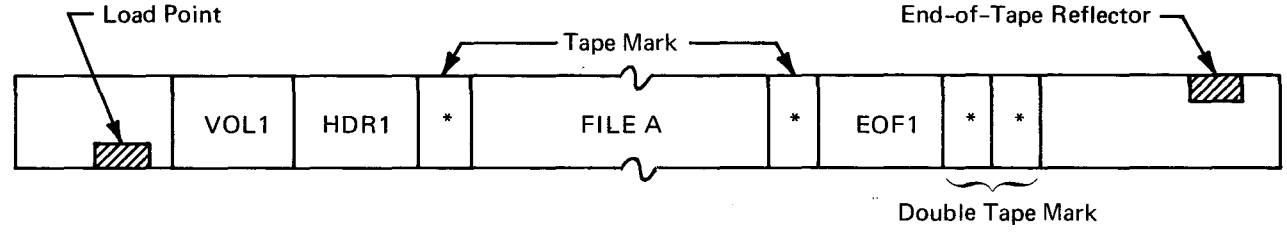
EOV1 — Required only if physical end-of-tape reflector is encountered before an EOF1 is written or if a multifile set is continued on another volume. It is preceded by a single tape mark and followed by an EOV2 label or a double tape mark.

EOV2 — Written after the EOV1 label by the NOS/BE operating system. This label is not a required ANSI label. When the EOV2 label is present, it contains the VSN of the next reel in the multireel set. The user can either specify the VSNs for the set with a VSN statement before the tape is written or let the operator assign the VSNs when the tape is written.

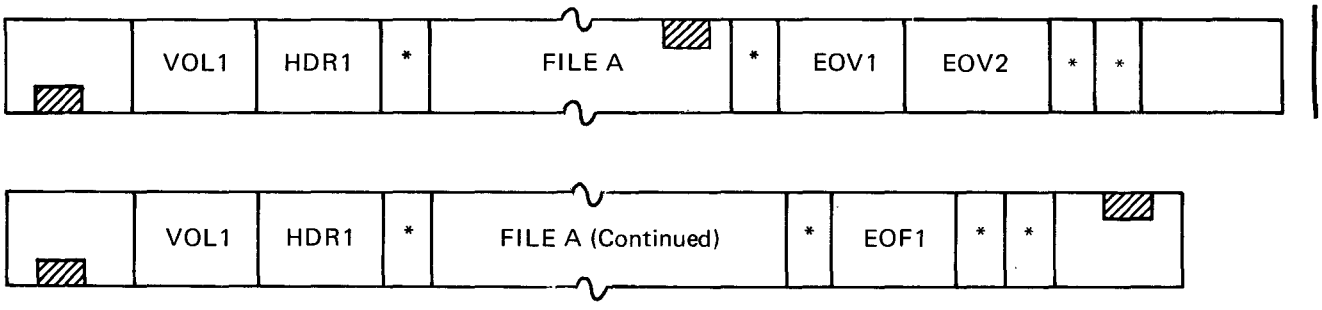
The EOV2 label is preceded by an EOV1 label and followed by a double tape mark. An EOV2 label is not required to read a multireel set.

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

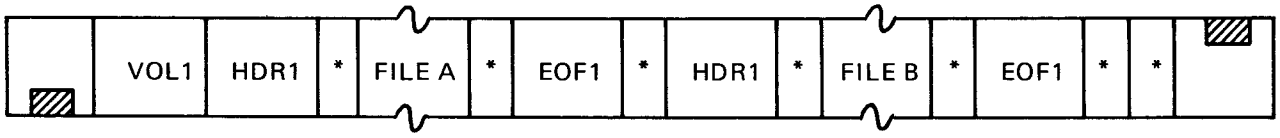
Single volume file:



Multi-reel file:



Multifile volume structure that results from a request for a multifile set is:



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

LABELED MULTIFILE SETS

A multifile 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 multifile 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 multifile 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 multifile 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 or as the name of a multifile set on a RETURN or UNLOAD control statement.

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

On REQUEST, the MF parameter designates the first parameter to be a multifile name rather than a file name. The U parameter causes standard labels to be produced. Other parameters should establish tape density and format for the entire multifile 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 multifile 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, use the following statements.

```
REQUEST(MANY,MF,U,E,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 multifile can be done at the end of the existing set. At some point prior to the end, existing files can be overwritten. For example, to create a new file LASTONE, use

```
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 multifile can proceed automatically with the use of a VSN under the following conditions:

VSN statement or parameter equates the multifile name to the physical volume of tape.

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

A REQUEST statement is used to assign the multifile 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 multifile set member.

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

Once the multifile 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. The three control statements involved are REQUEST, LABEL, and VSN.

The REQUEST statement can be used for all tape files (labeled, unlabeled, single file, or multifile 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 multivolume 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 multifile 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 multifile volume is to be labeled, a REQUEST statement must first establish the multifile 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 multifile 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.

For a multivolume file EOV2 labels, a VSN statement, or operator commands, identify the VSNs of the continuation reels. A labeled tape or an unlabeled SI format tape may have an EOV2 label placed after the EOV1 label. This label contains the VSN of the next reel. The VSN in the EOV2 label is the VSN specified by a previous VSN statement or the VSN specified by the operator. If the user wants to override existing EOV2 labels or if no EOV2 labels exist, the user should enter a VSN statement to identify the VSNs of the continuation reels. When the job's tape file requirements change frequently, the user should specify the VSN statement so the operator knows the required tapes. An operator can specify a VSN and it will override both an EOV2 label and a VSN statement.

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 (seven-track) and NT, HD, PE, and GE (nine-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. Depending on installation options for tape scheduling and default density (refer to the NOS/BE Installation Handbook), for nine-track tapes, the job statement density request and the density specified on the LABEL or REQUEST statement must be the same.

PRINT FILES

Print files contain a disposition code indicating printer output. The file OUTPUT always is a print file.

Print files must have the following characteristics.

Characters must be in 6-bit display code (IC=DIS) or 8-bit ASCII (IC=ASCII). Display code files contain ten 6-bit characters per 60-bit CM word. Eight-bit ASCII files contain five 8-bit characters right justified in each 12-bit byte. Bits 7-11 of each 12-bit byte are ignored. 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.

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.

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

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.

Character	Action
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	Output remainder of line (up to 30 characters) on the B display and the dayfile and wait for the JANUS entry /OKxx.
V	Specifies a new carriage control array to be loaded for a 580 printer.

The remaining carriage control characters to not inhibit printing. Only the carriage control character is not printed. Any preprint skip operation of 1, 2, or 3 lines that follows a postprint 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.

TABLE 3-4. 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)	Space 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.

The V function can be used when assigning output to a 580 printer with programmable format control. Such a printer does not use carriage control format tapes; instead it contains a microprocessor plus memory. Programmable format 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, there are no restrictions as to how the array is split, but 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 are specified to indicate the following.

Character	Significance
A	Top of forms code; the array must begin with an A.
B through K	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.
L	Bottom of forms code.
O	End of the array; must be specified as the last character in the array. However, it does not correspond to any line on the form.
blank	No channel. Blanks increase the number of lines on the form.

Any other characters are illegal and invalidate the array.

Regardless of whether the array is contained on one or two lines, 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 printer with programmable format control, the printer page ejects and does not print the line(s).

The following examples illustrate typical carriage control output and its effect.

1. column 1 23 4567 8901 23 4567 8 9 01 2 34
   array V6A B C D EFGHI JK O

   This causes the 6-line per inch buffer to be loaded with a 22-character array, implying a 21-line form.

2. column 1 23 4567 8901 234
   array V8ABCDEFGHI JKO

   This causes the 8-line per inch buffer to be loaded with a 12-character array, implying an 11-line form.

3. column 1 2345678 901 2345 67
   array VCA B D C
   VC E F G H I JK O

   This causes the 8-line per inch buffer to be loaded with a 22-character array, implying a 21-line form.

4. column 1 2345 6
   array V6BCDO

   This is invalid because the array does not begin with an A.

5. column 1 23456789
   array VBA C DEO

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

6. column 1 2345 6
   array V8ABWC

   This is invalid because W is an illegal character and the array does not end with an O.

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, except possibly on the ACCOUNT statement (depending on the installation).

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 44_8 except *) . $ and blank. (A blank can be used to separate the verb from the first parameter.) The comma and left parenthesis are preferred separators. Refer to appendix A for display code values.

Parameters — Parameter format and order depends on the individual control statements. 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 ore 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 allowscontinued control statements.) In general, the last nonblank character of the card or line to be continued must be a separator, and 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 7.

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.

A 26 or 29 can be punched in columns 79 and 80 of the job statement to indicate whether the statements following are punched in O26 or O29 character codes. The default mode depends on an installation option (see appendix A).

All numbers on job statements (except 26 or 29 in columns 79 and 80) 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:

name,Tt,IOt,CMfl,ECfl,Pp,Dym,MTk,NTk,HDk,PEk,GEk,CPp,STmmf.

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

name — 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, for 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 since certain kinds of program errors, such as an infinite loop, can result in great waste.

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 (request field length) 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 value must not exceed the installation-defined limit unless STmmf is also specified (EC will be ignored if the installation limit is zero and STmmf is not specified). An installation default amount (typically zero) is assigned if the

parameter is omitted and subsequent MEMORY and RFL requests from user programs are not allowed to exceed that amount. The installation can specify a default amount to be assigned when EC is specified without fl.

The EC parameter is applicable only to user programs in which ECS is accessed. If the ECS parameter is specified, the job will start either with no ECS assigned or with the assigned ECS equal to the parameter, depending on the option selected by the installation. In the case in which no ECS is assigned, it is the same as if a REDUCE,ECS. control statement had just been processed. In the case in which the ECS assigned is equal to this parameter, it is the same as if an RFL,EC=fl control statement had just been processed.

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
HDk
PEk
GEk

MT specifies seven-track tape. GE, PE, HD, and NT specify nine-track tapes with the following densities.

GE	6250 cpi [679 group coded recording (GCR) unit only]
PE	1600 cpi
HD	800 cpi
NT	Installation-selected default density

k is the maximum number of seven-track or nine-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.

Depending on the installation option for tape scheduling, the following rules for specifying density and k apply. If the installation has selected the schedule-by-density option, three separate counts according to density are maintained for each job (for example, the number of GE tape units is counted separately from the number of HD tape units). If the installation has not selected the schedule-by-density option, only one count of nine-track tape units is maintained.

A job can use more than a total of k tape units as long as their use is not simultaneous. For example, if MT3 is specified, seven-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 four 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; usually, both processors will be used during the execution of any program. p can be A or B.

On a 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. However, it is useful for running CPU diagnostic programs.

On a CYBER 74-2x or 6700 system, the two processors operate at different speeds. CPA restricts the job to the faster processor, and 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 multimainframe environments, ST should be used to ensure that a string of interdependent jobs is executed in the same mainframe.

Examples of job statements:

```
JOBA100,T400,CM45000,EC2,P1,DAB3,MT5,CPA.  THE JOB NAME IS TRUNCATED TO JOBA1

K2S1.  ALL DEFAULT VALUES ARE AUTOMATICALLY ASSIGNED

TLS,T777,IO777,CM50000,EC5,NT2,P1,MT1.

JOB4,T77777,IO0,NT1.  THIS JOB HAS INFINITE CENTRAL PROCESSOR AND I/O TIME
```

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. Installations can prohibit absolute dumps.

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 from.

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 7.

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 and 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 (CREATE MASTER DEVICE OR ADD DEVICE TO PRIVATE 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,SN=setname,MP=mpvsn,VSN=vsn,UV=uv,UP=up,PB=pb,FR=fff,NF=n,NM=m,RP=ddd,*PF.

Parameters SN, MP, and VSN are required. If parameters MP and VSN are equal, parameters UV, UP, PB and FR are required unless the installation defines defaults. All parameters are order independent.

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.

MP=mpvsn — 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.

UV=uv† — Universal password; 1-9 letters or digits.

UP=up† — Universal permission; any non-null combination of the characters C, M, E, and R, which specify the following permissions.

C	Control permission
M	Modify permission
E	Extend permission
R	Read permission

PB=pb† — Public password; 1-9 letters or digits.

†This parameter applies only when a master device is being added.

FR=fff† Permanent file default retention period specifying the number of days permanent files on this private set are to be retained; 0-999. The private set owner determines the future of each file once the retention period expires.

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 tables for the member devices on the master device system. 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.

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.

†This parameter applies only when a master device is being added.

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	Local 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 multimainframe environment, the permanent file must reside on a device 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,CY=cy,EC=ec,LC=n,MR=m,PW=pw,UV=uv,RW=p,SN=setname.

The first parameter establishes the local file name by which the file is known to the job. Parameter pfn is required. Parameters lfn (if present) and pfn are order dependent. All other parameters are optional depending on how the file was cataloged. They are order independent. The ATTACH statement can be continued from one line to the next. The first line must not be terminated by a period or a right parenthesis. To be consistent with other control statements that require such a format, the last nonblank character on the line should be a separator. The continuation begins in column 1 of the next line.

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.
ID=name	ID parameter by which the file was cataloged. Required unless the file was cataloged with ID=PUBLIC.
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	Buffer Size
K	Installation standard number of blocks of ECS.
nnnn	Number of 1000 (octal) word blocks to be allocated.
nnnnK	Same as EC=nnnn.
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 — Multiread permission.

m	Significance
0 or omitted	File may be attached with all the permissions established by the creator of the file.
Nonzero digit	File can be attached only with read permission.

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.

UV=uv — Universal password; 1-9 letters or digits. Grants universal permission. Password and permission for public sets are defined by the installation; for private sets, they are defined on the ADDSET statement when creating the master device. If this parameter is specified, PW parameters are ignored.

RW=p — Rewrite request.

p	Significance
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 (refer to SETNAME statement).

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, $\left\{\begin{matrix}AI=F\\AI=P\end{matrix}\right\}$,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 — Name of file 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.

m	Mode
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 2-line output for each file audited. Default.

AI=P — Partial 1-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 with leading zeros assumed. 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 CDC 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. If n is set to zero, the system treats it as n=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	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; 1-9 letters or digits. Required unless the installation is cataloging the file with ID=PUBLIC.
AC=act	Account parameter, 1-9 letters or digits. Installation determines the procedure if act is incorrect or is not specified.

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 — Multiread indicator.

m	Significance
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. All permissions except read are cancelled.

PW=pw — Password list to obtain permissions. Control password is required to catalog a new cycle of the same pfn/ID. Public password is required to catalog the initial cycle of a file with 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.

p	Significance
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. (Refer to the RESTART utility.)

Checkpoint/restart cannot handle the following items.

Rolled-out jobs

Random files (except random permanent files)

Multifile 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 have 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_8 since checkpoint uses level 16_8 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 the first level 17_8 system-logical-record (partition) boundary.

The format of COMBINE is:

COMBINE,lfn_1,lfn_2,n.

Parameters lfn_1, and lfn_2 are required.

lfn_1	File from which one or more system-logical-records is read, 1-7 letters or digits beginning with a letter.
lfn_2	File to which one system-logical-record is written, 1-7 letters or digits beginning with a letter. (lfn_2 cannot be the same file as lfn_1.)
n	Number (decimal) of system-logical-records in lfn_1 to be written onto lfn_2. Default is 1. If n is zero, COMBINE terminates at a level 17_8 system-logical-record (partition) boundary.

The job is responsible for positioning of both files.

COMMENT (ADD COMMENT TO DAYFILE)

COMMENT inserts a formal comment into the job dayfile. Since the comment is displayed at the operator console as part of the job dayfile and the job continues, 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,lfn_1,lfn_2,n,lev,e,r.

Parameters lfn_1 and lfn_2 are required; all others are optional. All parameters are order dependent.

lfn_i	Name of file 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 lfn_1, to be compared to lfn_2. Default is 1.
lev	Record level number (octal). Default is 0.
e	Number (decimal) of nonmatching word pairs to be written to the OUTPUT file for each nonmatching record. Default is 0.
r	Number (decimal) of nonmatching records to be processed during the comparison. Included in nonmatching 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. If a parity error occurs, the file is backspaced and re-read in coded mode. If another parity error 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 parity error, 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 error, following records of GREEN are read in the same mode until a change is forced by a parity error.

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-paritions 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,lfn_1,lfn_2.

Parameters are order dependent and optional.

lfn_1 File to be copied onto lfn_2, 1-7 letters or digits beginning with a letter. Default is INPUT.

lfn_2 File onto which lfn_1 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. lfn_1 is copied through end-of-information or a double end-of-partition. File format is not changed, and FILE control statements are ignored (refer to CYBER Record Manager manuals).

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 lower order position of the last word of the line image. COPYBCD writes each line image as a 140-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,lfn_1,lfn_2,n.

All parameters are positional and optional.

lfn_1 Name of input file to be copied onto lfn_2, 1-7 letters or digits beginning with a letter. Default is INPUT.

lfn_2 Name of output file onto which lfn_1 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_8 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 multifile 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, 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,lfn_1,lfn_2,n.

All parameters are order dependent and optional.

lfn_1	Name of file from which information is to be copied, 1-7 letters and digits beginning with a letter. Default is INPUT.
lfn_2	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,lfn_1,lfn_2,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 lfn_2, 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_8 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_8 lose their level indicator on an S or L tape.

For the record and block types indicated below, CDC 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 CDC 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.

The format of COPYBR is:

COPYBR,lfn_1,lfn_2,n.

Parameters are order dependent and optional.

lfn_1	Name of file from which information is to be copied, 1-7 letters or digits beginning with a letter. Default is INPUT.
lfn_2	Name of file to which information is to be copied, 1-7 letters or 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,lfn_1,lfn_2,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 does not abort) 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, CDC 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 CDC CYBER Record Manager 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 lfn_1. 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.

COPYL/COPYLM (BINARY COPY WITH REPLACEMENT)

The COPYL and COPYLM control statements copy an old file to a new file substituting records from a replacement file for the matching records on the old file. Records on the replacement file which do not match records on the old file are ignored unless the user specifies that they be appended to the new file. Records are considered matching if they have the same type and the same name; however, the user may specify that the record type be ignored. COPYL and COPYLM are commonly used to maintain files of procedures or subroutines.

COPYL and COPYLM differ only in the handling of multiple occurrences of a record on the old file. COPYL uses each record on the replacement file only once, replacing the first matching record from the old file. COPYLM uses the first matching record encountered on the replacement file to replace each matching record from the old file. COPYL can be used to replace multiple occurrences of the same record if multiple occurrences of the record are in the replacement file.

The old file and the replacement file must reside on mass storage or a system-logical-record format tape. Only a single file terminated by an end-of-file marker is processed by a single call to COPYL or COPYLM unless the user requests processing to the end-of-information by using the E parameter. When working with multifile files, the user must be sure to position the multifile file to the file that is to be processed.

The order of the records on the replacement file is not significant. The system copies the records to the new file in the same order as on the old file.

COPYL and COPYLM issue dayfile messages during processing; no other printed output is produced unless the command is issued from an interactive terminal. The dayfile messages list which replacement records were copied and which replacement records were not copied to the new file. These messages are issued immediately to interactive terminals.

COPYL and COPYLM replace only the types of records listed in table 4-3. Any record on the old file that is not recognized as one of the listed types is copied to the new file without further processing. Any replacement file record type that is not listed in table 4-3 is ignored without comment.

The formats of COPYL and COPYLM are:

COPYL(oldlfn,replfn,newlfn,last,flag)	Single replacement.
or	
COPYLM(oldlfn,replfn,newlfn,last,flag)	Multiple replacement.

All parameters are optional and position dependent. A user denotes an omitted parameter by consecutive commas.

oldlfn	File name of the old file; default name is OLD.
replfn	File name of the replacement file; default name is LGO.
newlfn	File name of the updated file; default name is NEW.
last	Name of the last record on oldlfn to be processed. If last is not specified, all records on oldlfn are processed from its current position to the next end-of-file (or end-of-information if the E parameter is used).
flag	Processing parameters.

Flag	Description
R	Rewind oldlfn and newlfn files before processing. (replfn file is always rewound before and after processing. Oldlfn and newlfn are not necessarily rewound to beginning of information in multifile files. Refer to explanation below.)
A	Append to the end of newlfn all replfn records that do not match any records on the oldlfn. If A is not selected, records on the replacement file that do not match any records on the oldlfn are ignored and a dayfile message is issued.
T	Check for matching name of record, but omit check for matching type of record. If T is not selected, records match only if both the type and name of the records are the same.
E	Process oldlfn until the end-of-information.

These parameters can be specified by combining one or more letters in any order, such as TRA, AR, RTEA, or TR.

COPYL and COPYLM check only the first four flag parameters; if more than four are specified, the remaining characters are ignored.

The R parameter affects file positioning of the old and new files before processing. If R is specified, the old and new files are rewound before processing. In a multifile file, if there is one or more end-of-file markers between the current position of the file and the beginning-of-information, the R parameter rewinds the file to the first preceding end-of-file. In the absence of R, the user is responsible for positioning the oldlfn and newlfn files. The R parameter does not affect the file of replacement records, since the current file of the replacement file is always rewound to the beginning-of-information before and after processing.

The E parameter causes the old file to be processed to the end-of-information. Each end-of-file encountered on the old file causes a matching end-of-file to be written on the new file. Records added to the new file as a result of an AE parameter combination are appended with an end-of-file prior to the end-of-information. Here, users should note that such appended records follow an end-of-file if both end-of-file and end-of-information existed at the end of the old file.

Processing stops after an end-of-file, end-of-record, or end-of-information is reached, depending on the structure of the old file and the processing parameters selected. If processing stops because an end-of-file or end-of-record is reached, the old file will be positioned after that end-of-file or end-of-record. If processing stops because end-of-information is reached, the old file will be positioned just prior to the end-of-information.

COPYL and COPYLM add an end-of-file to the new file even if no end-of-file is encountered on the old file. No further positioning of the new file takes place.

TABLE 4-3. TYPES† OF RECORDS REPLACED BY COPYL AND COPYLM

Type	Description	Type	Description
ABS	Central processor overlay with one or more named entry points	REL	Relocatable central processor program
CAP	Capsule	TEXT	Text record
OVL	CP overlay with one unnamed entry point including system texts	6PP	6000 Series peripheral processor program
PROC	CYBER Control Language procedure file	7PP	7000 Series peripheral processor program

†For additional information about how these types are determined, refer to appendix F.

COPYN (CONSOLIDATE FILE)

COPYN consolidates or merges files. System-logical-records from up to 10 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,inlfn$_1$, ...

Parameters are order dependent and required. Up to 10 inlfn parameters can be specified.

f	Format of output record.	
	0	Copy records verbatim.
	non-zero	Omit ID from record.
outlfn	File name of output file, 1-7 letters or digits beginning with a letter.	
inlfn$_i$	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 the statement in error followed by an error message is printed on OUTPUT. 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, and if it is set, the job is terminated. Otherwise, control is given to the next control statement.

REWIND (REWIND FILE)

The REWIND directive rewinds the named file. This file must be one of the input or output file names given on the COPYN control statement, not 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 file. 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. Levels 1 through 16 are not recognized by the skip.

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. n skips forward n records; -n skips backward n records.

WEOF (WRITE FILE MARK)

WEOF writes a partition boundary on the named file.

The format of the WEOF directive is:

WEOF(lfn)

lfn	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:

p_1,p_2,p_3

p_1 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.

p_2 Last record to be copied in a set of records:

name	System-logical-records p_1 through p_2 are copied. p_2 must be located between p_1 and end-of-information.
decimal integer	Number of records to be copied, beginning with p_1. Zero-length records and file marks are counted. Copying stops when the file end is encountered, even if the count has not been reached.
*	p_1 through an end-of-partition are copied.
**	p_1 through a double end-of-partition are copied.
/	p_1 through a zero-length record are copied.
0 or blank	Only p_1 is copied.

p_3 Input file to be searched. If p_1 is a name, and p_3 is omitted, all input files declared on the COPYN statement are searched until the p_1 record is found. If it is not located, a message is issued. If p_1 is a number and p_3 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 p_1 is positioned at the record following p_2. Other files remain effectively in the same position.

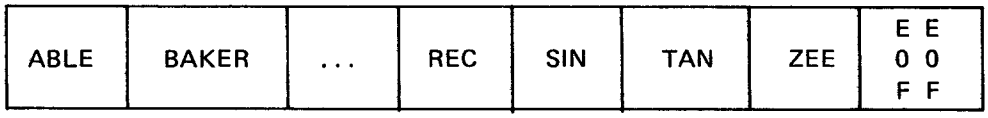
When COPYN is searching for a named record and p_3 has been omitted, each input file is searched in turn until either the named record is found or the original position of the file is reached. The job INPUT file, however, is not searched end-around.

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 p_2 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.

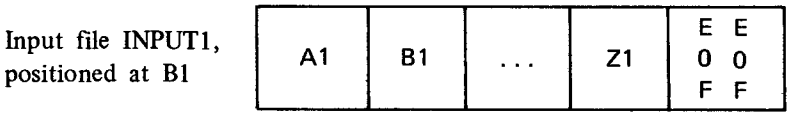
Examples of file positioning:

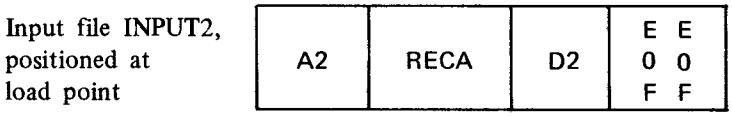
1. Record identification statement: REC,,INPUT1

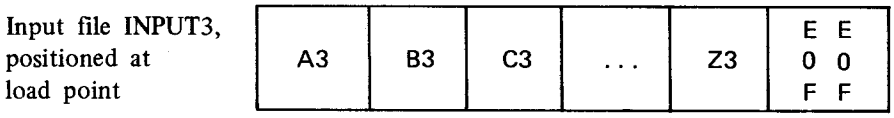


If INPUTA were positioned at TAN, TAN and ZEE would be examined for REC. The double EOP would cause ABLE to be the next system-logical-record examined, continuing until REC is read and copied to the output file. INPUT1 would then be positioned at SIN.

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,lfn_1,lfn_2.

Parameters are order dependent and optional.

lfn_1 — Name of input file to be copied onto lfn_2, 1-7 letters or digits beginning with a letter. Default is INPUT.

lfn_2 — Name of output file onto which lfn_1 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	File name of input X tape, 1-7 letters or digits beginning with a letter.
scplfn	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 and blank-labels a member device 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. The member device must be on-line (not necessarily mounted) before DELSET is issued so that it can be blank-labeled and the flaw table updated. 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=mpvsn,VSN=vsn.

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=mpvsn	Volume serial number of master device for the device set, 1-6 letters or digits with leading zeros assumed.

VSN=vsn 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 either 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=Cfc / 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. If * is used to dispose the file OUTPUT to the central site (*dc=C) for a job that originated elsewhere, a copy of the dayfile is sent to the job's origin at end-of-job.

dc Disposition code.

SC	Evict the file (default)	P8	Punch 80-column free-form binary format
PR	Print on any available printer		
PE	Print on ASCII 95-character print train	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	IN	Place file in the input queue

†Supporting drivers must be supplied by the installation.

C File is to be routed to the central site.

Cfc Forms code for special card or paper forms. 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.

DISPOSE EXAMPLES

1.
```
JOB.
COBOL.
LGO.
DISPOSE,OUTPUT,PR.          Prints OUTPUT on any available printer.
REWIND(LGO)
FTN5.
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. Three 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 are equal in value and not 0.
Relative dump	Parameters specify address within field length.

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 within the field length. If the P register is 0, the P address in bits 30-47 of RA+0 determines 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 also given 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 pair of identical, nonzero octal values indicates the control point area is to be dumped.

This control statement dumps the entire (200 octal word) control point area of the job. The actual octal value specified is not significant. If the two octal values are nonzero and identical, the control point area of the job will be dumped.

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.

DMP EXAMPLES

1. DMP,1,1. Dumps the control point area of the job.
2. DMP,0,0. Dumps the exchange package of the job.
3. DMP,100,200. Dumps from address 100 through 200 of the job's field length.
4. DMP,100. Dumps from the beginning of the job's field length through address 100.
5. DMP. Dumps the exchange package of the job.

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 backup 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	Backup 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 and defined passwords for RD, MD, CN, and EX. If a DUM permanent file with TK=DUMPF already exists (earlier systems required this), it must be purged and replaced as described above. 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=lfn_1,LF=lfn_2,CL,DP=a,ID=name,PF=pfn,CY=cy,SN=setname,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. Refer to CATALOG control statement for password definitions.

MO=n — Dump mode:

n	Mode
1	Backup 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. MD 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=lfn_1 — Name of directive file for MO=1 dump; 1-7 letters or digits beginning with a letter. If lfn_1 is not specified, directives for MO=1 are on INPUT. If a directive file is used, the following parameters are not allowed on the DUMPF statement: ID=name, PF=pfn, CY=cy, VSN=vsn, IN=ddd, JN=yyddd, LA=mmddyy, DA=yyddd, CD=mmddyy, and TI=hhmm.

LF=lfn_2 — 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	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 or full dump.

ID=name — Dump files with this owner.

PF=pfn — Dump files with this permanent file name. ID should be specified also; if it is not specified, ID=PUBLIC is assumed.

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=setname — 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 not attached within this number of days; 1-3 digits. Can be qualified by a TI parameter.

JN=yyddd	Dump files not attached on or after this date; five-digit ordinal date format. Can be qualified by TI parameter.
LA=mmddyy	Dump files not attached on or after this date; six-digit month-day-year format. Can be qualified by TI parameter.
DA=yyddd	Dump files created, modified, renamed, or extended after this date; five-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; six-digit month-day-year format. Can be qualified by TI parameter.
TI=hhmm	Time qualifier for date parameters; four-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 the same mode and type (DP parameter). If an attempt is made to run a DUMPF with a different mode or type than one already running, all except the first DUMPF abort. Several copies of DUMPF can execute at the same time on different sets and the modes and types need not match.

If a group of files is to be dumped for backup 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 formats are as follows.

```
ID=name
ID=name, PF=pfn
ID=name, PF=pfn, CY=cy
```

Parameters are order independent and ending punctuation is not required. The PF and CY parameters are optional. The ID parameter should be specified. However, if the PF parameter is specified without an ID parameter, ID=PUBLIC is assumed.

DUMPF EXAMPLES

1.
```
DAYDMP, . . .
REQUEST(DUMTAPE,NT,PE,S,N)
DUMPF(PW=PERM1,DA=78164)
6/7/8/9
```

   The job DAYDMP dumps all files cataloged, modified, renamed, or extended after the 164th day in 1978.

2.
```
SELDMP, . . .
REQUEST(DUMPTAPE,MT,HY,S,N)
DUMPF(PW=PERM1)
7/8/9
ID=DEVCTR
PF=FILE1,ID=LER
PF=FICHE,ID=GFS,CY=1
6/7/8/9
```

Job SELDMP dumps the files specified in the input section of the control statement record. All files with ID DEVCTR are dumped.

3. ARCHIVE, . . .
PAUSE. BRING UP P DISPLAY TO INSURE DUMP TAPE HAS A VSN.
REQUEST(DUMTAPE,MT,HY,S,N)
DUMPF(MO=2,IN=10,PW=PERM2)
6/7/8/9

Job ARCHIVE illustrates a 10-day archive dump.

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, replacement of routines, and statistics about library contents can be listed.

A user library can only contain assembled central processor routines, CCL procedures, 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, overlays, or CCL procedures. 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 a 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

CCL procedures

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 to or 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=lfn_1,L=lfn_2)

All parameters are optional.

USER	Distinguishes user library definition from system library. Default is USER.
lfn_1	File name containing directives, 1-7 letters or digits beginning with a letter. Default is INPUT. I is identical to I=INPUT.
lfn_2	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.
FTN5.
ATTACH(ALIB,ID=SMITH)
EDITLIB(USER)
EXTEND(ALIB)
7/8/9
     COMPASS program to be assembled
7/8/9
     FORTRAN 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 as follows:

LIBRARY(libname, {OLD / NEW})	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 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.

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 and REPLACE directives only, several individual programs can be stated. In a file with records A,B,C,D,E, the parameter D/B/E represent D and B and E. EDITLIB searches the entire file specified to find the named program.

Program names must not exceed seven 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.

†The definition of current file position depends on the file format. The current file position of a library file is always defined to be the beginning of the file. The current position of other files is the beginning of the next record in the file. The user can control the current position of these other files with file manipulation directives.

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 ZZ 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 the ADD directive is:

ADD(prog,lfn,AL=level,FL=fl,FLO=$\left\{\begin{matrix}0\\1\end{matrix}\right\}$,LIB)

Parameters prog and lfn are required; all others are optional.

prog	Name of program or range of programs to be added.
lfn	Name of file 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 lfn 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. Current values in source library or existing library tables are not preserved. 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-partition of TREES are 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(SEND/MONEY,PROCFIL)	Procedure file PROCFIL is searched as needed and procedures SEND and MONEY are added to the user's library.
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 the CONTENT directive is:

CONTENT(prog,lfn)

prog	Program or range of programs to be listed.
lfn	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 the DELETE directive 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 either to construct a new library using the old one as the source or to use RANTOSEQ followed by SEQTORAN, which compacts the library and preserves attributes of programs in the library.

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 the ENDRUN directive is:

ENDRUN.

FINISH (STOP FILE MANIPULATION)

FINISH indicates the end of library construction.

The format of the FINISH directive is:

FINISH.

LIBRARY (IDENTIFY 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 the LIBRARY directive is:

LIBRARY(libname, $\left\{\begin{matrix}\text{OLD}\\\text{NEW}\end{matrix}\right\}$)

libname	Library name and name of file containing library during this job.
OLD	Used when libname is an existing library to be modified.
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 the LISTLIB directive is:

LISTLIB(prog,lfn)

prog	Program or range of programs to be listed.
lfn	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 the RANTOSEQ directive is:

RANTOSEQ(rlfn,slfn)

rlfn	Disk resident random library that is to be converted.
slfn	Sequential library created from rlfn; slfn is not rewound after the copy.

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 the REPLACE directive is:

REPLACE(prog,lfn,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 the REWIND directive is:

REWIND(lfn) or REWIND(lfn/lfn/ . . . lfn)

lfn 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 the SEQTORAN directive 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 the SETAL directive 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 the SETFL directive 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 the SETFLO directive is:

SETFLO(prog, $\left\{\begin{matrix}0\\1\end{matrix}\right\}$)

prog	Name of program or range of programs.
0	Does not allow override; 0 is the default value.
1	Allows 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 the SKIPB directive 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	File name containing prog, 1-7 letters or digits beginning with a letter.

The format of the SKIPB directive for files is:

SKIPB(n,lfn,F)

n	Number (decimal) of files to be skipped backward; cannot be zero.
lfn	File name of multi-file, 1-7 letters or digits beginning with a letter.
F	Indicates files, not records, are to be skipped.

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 the SKIPF directive 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	File name containing prog, 1-7 letters or digits beginning with a letter.

The format of the SKIPF directive for files is:

SKIPF(n,lfn,F)

n	Number (decimal) of files to be skipped forward; cannot be zero.
lfn	File name of multifile, 1-7 letters or digits beginning with a letter.
F	Indicates files, not records, are to be skipped.

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.

FTN5.
EDITLIB(USER)
7/8/9
      .          FORTRAN 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,MT,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. Refer to the CYBER 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)

The EXIT control statement establishes the conditional processing of sequences of control statements when certain fatal errors occur. If an error causes a job step to terminate (table 4-4), the system aborts the job and searches the job control statement file for EXIT control statements, skipping other control statements in the process. If the system finds no EXIT statement, the job is terminated as described in Job Processing and Deck Structure, section 2. If the system finds two consecutive EXIT statements, the job is terminated.

The formats of the EXIT statement are:

```
EXIT.

EXIT,C.

EXIT,U.

EXIT,S.
```

C	Conditional processing option.
U	Unconditional processing option.
S	System processing option.

The type of error that occurs dictates the type of EXIT processing to be performed. Some error conditions bypass EXIT processing and terminate the job immediately. Error conditions are classified as follows:

Job step abort	Terminates the current job step and starts the search for any of the four types of EXIT control statements. Most error conditions in the system are in this classification.
Special abort	Terminates the current job step processing and starts the search for an EXIT,S control statement.
Terminal abort	Terminates the current job step and the job immediately. No EXIT processing takes place.

Table 4-4 describes the type of EXIT processing performed when various errors occur.

EXTEND (PERMANENT FILE EXTENSION)

EXTEND makes information written at the end of an existing permanent file permanent. Information can be written at the end of any attached permanent file. However, in the absence of an EXTEND or ALTER control statement, 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	Name of permanent file attached with extend permission, 1-7 letters or digits beginning with a letter.

TABLE 4-4. EXIT PROCESSING

Condition Causing Job Step Termination	Type of Termination and Action Taken on Occurrence	Action Taken When EXIT Encountered			
		EXIT.	EXIT,C.	EXIT,U.	EXIT,S.
Successful completion (no error or only non-fatal errors). ENDRUN macro.	Normal job step advance; advances to next control statement and processes it. Terminates job if end of control statement record encountered.	Terminates job.	Resumes processing after EXIT,C.	Resumes processing After EXIT,U.	Terminates job.
Peripheral processor encountered improper I/O request. Time limit exceeded (first time only). Operator DROP. User arithmetic error not negated by a MODE control statement. ECS parity error.	Job step abort; aborts job step and skips all control statements until an EXIT statement is found. Terminates job if no EXIT found before end of control statement record encountered.	Resumes processing after EXIT.	Terminates job.	Resumes processing after EXIT,U.	Resumes processing after EXIT,S.
Loading program with compilation or assembly errors. ABORT,NODUMP macro. ABORT,,S macro. ABORT,NODUMP,S macro. Control statement error.	Special abort; aborts job step and skips all control statements until an EXIT,S. Terminates job if no EXIT,S found before end of control statement record.	Continues skipping.	Continues skipping.	Continues skipping.	Resumes processing after EXIT,S.
Job statement error. ACCOUNT statement error. Operator KILL. Operator RERUN. Time limit exceeded (second time). Checksum error during job input. Two consecutive EXIT statements.	Terminal abort; aborts job step and terminates job.	Not applicable.	Not applicable.	Not applicable.	Not applicable.

GENLDPF (RELOAD PERMANENT FILE CATALOG)

GENLDPF reads a log tape created by the PFLOG utility and generates LOADPF jobs, which will load the files that had a permanent file catalog (PFC) entry at the time PFLOG was run. This allows the installation to do a full reload of the permanent file base without reloading files purged since the last full dump.

Before GENLDPF is called, a REQUEST control statement must define a log file as an existing labeled SI tape whose logical file name is LOGTAPE.

For each entry read from the log tape, GENLDPF makes an output listing entry that contains the permanent file name, owner id, and cycle number.

The format of GENLDPF is:

GENLDPF,PW=pw,SN=setname,VSN=vsn,LF=lfn.

PW is required; all other parameters are optional. However, SN is specified if VSN is specified, and vice versa. All parameters are order independent.

PW=pw	EX password required for generated LOADPF jobs.
SN=setname	Name of device set onto which permanent files are to be reloaded, 1-7 letters or digits beginning with a letter. The master device for this set must have been mounted before GENLDPF can execute. Default is the permanent file default set.
VSN=vsn	Volume serial number of the master device of the device set specified by SN=setname.
LF=lfn	Name of file on which the listing is to appear, 1-7 letters or digits beginning with a letter. Default is OUTPUT. If lfn=0, no listing is generated.

GENLDPF EXAMPLES

1.
```
JOBX(NT01)
VSN(LOGTAPE=123456)
REQUEST(LOGTAPE,NT,PE,E,NORING)
GENLDPF(PW=HELLO)
6/7/8/9
```

   This job reloads files onto the permanent file default set and writes the output listing on OUTPUT.

2.
```
JOBY(NT01)
VSN(LOGTAPE=246801)
MOUNT(SN=SETNAME,VSN=MASTER)
REQUEST(LOGTAPE,NT,PE,E,NORING)
GENLDPF(PW=LOAD,SN=SETNAME,VSN=MASTER,LF=0)
```

   This job reloads files onto set SETNAME whose master pack vsn is MASTER. No output listing is generated.

GETPF (ATTACH PERMANENT FILE FROM LINKED MAINFRAME)

GETPF enables users in a multimainframe environment to attach permanent files from a linked mainframe. It can attach 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 set of any of the linked mainframes. 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,EC=ec,$\left\{\begin{matrix}LC=n\\CY=cy\end{matrix}\right\}$,PW=pw,ST=mmf,SN=setname,VSN=vsn.

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. SN and VSN are optional, but if one is specified, they both must be specified. 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	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	The mainframe on which file lfn is cataloged; three characters. The values for mmf are established at installation time. Required.
SN=setname	Device set name identifying the private device set containing the permanent file to be attached. This parameter may be 1-7 letters or digits and must begin with a letter. If SN is specified, VSN must also be specified to allow access to the private set on the mainframe specified by ST.
VSN=vsn	Volume serial number identifying the master device of the private device set. This parameter may be 1-6 letters or digits. If SN is specified, VSN must also be specified as explained in the SN description.

Refer to the ATTACH control statement for the remaining parameters.

GETPF always sets MR=1.

When a file is referenced by GETPF, 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.

ITEMIZE (LIST CONTENTS OF BINARY FILE)

ITEMIZE lists pertinent information about each record of a binary file in a format suitable for printing. Table 4-5 describes the types of records processed by ITEMIZE.

ITEMIZE processes mass storage files or system-logical-record format tape files. A file can be processed from beginning-of-information through end-of-information.

Output from ITEMIZE is affected by the type of record and options selected. A header appears for each file terminated by an end-of-file marker within the file specified by the file name. The first line of the header identifies the file name, file position within that file, and the date and time of the run. The second line of the header has the following fields:

REC	Position of the record in the file starting with the first record of each file.
NAME	Record name obtained from the second word of the prefix table or from the first word of the record.
TYPE	Type of record as shown in table 4-5.
LENGTH	Number of words (octal) in the record, excluding the prefix table.
CKSUM	Cyclic logical checksum (octal), excluding the prefix table.
DATE	Date record was created as stored in the prefix table.
COMMENTS	Contents of the comments field in the prefix table.

If no prefix table is present, the associated fields are blank.

Additional information listed depends on the type of record:

ABS	Entry point names are listed.
DATA	First line of the record is listed if the name of the record is OVERLAY.
OVL	Overlay level is listed in octal.
TEXT	Entire record is listed if the name of the record is CMRDC, IPRDECK, IPRDC, LIBDECK, LIBDC, or COMMENT.
UPL	Deck names are listed.
6PP	Information stored by EDITLIB is listed giving the octal equivalent of the load address, residence, and control statement call flag.
7PP	PP number is listed.

The E parameter can select further details about several types of records.

The last record in each file is the end-of-file marker, which appears on the listing as the characters *EOF*. The SUM= identification is the total length, in words, for all records in the file, including the prefix table lengths.

Any zero-length record in the file appears with the record name (00). When it is encountered, a sum of the lengths of the records encountered since the beginning of the file, or since the last sum was taken, is listed on the output. The length includes prefix tables. Record numbering is not restarted until a new file is encountered.

If a record of type UPL has more correction identifier names and/or deck names than can be accommodated within the ITEMIZE buffer, the following message appears on the listing in place of the excess names:

TRUNCATED--IDENT OR DECK LIST TOO LONG

Here, the Update utility must be used to obtain a complete list of identifiers and deck names.

NOS/BE deadstart tapes can be recognized by ITEMIZE. For deadstart tapes, ITEMIZE lists deadstart records or the library name tables according to their positions on the tape. The remaining records are listed as usual, with the library name becoming part of the header for each file.

A dayfile message is issued when ITEMIZE completes execution.

The format of ITEMIZE is:

ITEMIZE(lfn,L=listlfn,BL,PW=n,PD,NR,N=n,E)

The first parameter is positional; if lfn is omitted, its position must be indicated by a comma. All others are optional and order independent.

Parameter	Description
lfn	Name of file to be itemized; default name is LGO.
L=listlfn	List output on file listlfn; default is L=OUTPUT.
BL	Burstable listing; each file output starts at the top of a page. Default is a compact listing in which a page eject occurs only when the current page is nearly full.
PW=n	Print either 136-character lines or 72-character lines depending on the value of the decimal integer n. If n $\geq$ 136 print 136-character lines. If $<$ 136, print 72-character lines. If =n is omitted, print 72-character lines regardless of the listing file device. If PW=n is omitted, the default value is 72-character lines if the listing file is a terminal; otherwise, the default value is 136-character lines.

Parameter	Description
PD	Print density at eight lines per inch; default is six lines per inch. If this parameter is to produce the desired result, the programmer must ensure that output appears at a printer with eight lines per inch capability.
NR	No rewind of lfn before or after processing; default is rewind before and after processing.
N=n	Itemize n files, where n is a decimal integer; default is N=1. If =n is omitted, itemize until end of information. If n is zero, itemize until an empty file is processed.
E	Expand output to list further information; default is no expansion. For record types CAP and REL, list entry points. For record types UPL, list correction identifier names.

TABLE 4-5. TYPES OF RECORDS LISTED BY ITEMIZE†

Type of Record	Record Description	Type of Record	Record Description
ABS	Central processor overlay with one or more named entry points.	REL	Relocatable central processor program.
CAP	Capsule.	SDR	Special deadstart record.
DATA	Not any other described record type.	TEXT	Text record.
LIBNT	Library name table record.	UCF	Update compressed compile file.
OVL	Central processor overlay with one unnamed entry point (no ENTRY statement in program); system text.	UPLx	Update sequential program library with x master control character.
PPNT	Peripheral processor program name table.	6PP	6000 Series peripheral processor overlay.
PROC	CYBER Control Language procedure file.	7PP	7000 Series peripheral processor overlay.

†For additional information about how these types are determined, see appendix F.

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 multifile set. To use a LABEL statement the job statement must specify the tape track type and density (refer to MTk parameter in the JOB statement earlier in this section).

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 multifile volumes, a REQUEST establishing the multifile 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 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},{EEC|IEC},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 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 seven-track tape in SI format. Any other label or data format to be written, or a tape to be read, must be declared explicitly.

Nine-track tape can be selected only by giving either a nine-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 issued, the tape can be a candidate for automatic 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.

Hardware error correction:

EEC	Enable hardware GE write error correction. The system allows certain types of single-track errors to be written that can be corrected when the tape is read (on-the-fly correction). This is the recommended mode of operation, because it provides efficient throughput, error recovery, and tape usage when writing GE tapes on media that is suitable for use at 3200 fci or 6250 cpi.
IEC	Disable all error correction activity in GE write mode. The system invokes standard error recovery processing when an on-the-fly error occurs while writing a GE tape. The system erases the defective portion of tape, thereby reducing the amount of data that can be stored on the tape. Only tape that is suitable for recording at 6250 cpi should be used when this mode of operation is in effect.

NOTE

EEC and IEC apply only to GE (6250 cpi) operations. GE must also be specified in a REQUEST statement; otherwise, EEC and IEC are ignored.

EEC and IEC are applicable if the user requests default nine-track density and the installation nine-track default density is GE (6250 cpi).

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

†200 bpi can be read but not written by 667/677 tape drives.

For nine-track tapes, the d parameter determines density for writing only; data is always reading at the recording density.

HD	800 cpi
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 all nine-track tape labels and for conversion of data on coded nine-track tapes of types S or L. 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.

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 format is two digits for year and three 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.

---

†6250 cpi density is supported only on 679 GCR tape drives.

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 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=dt,mode,I=lfn.

All parameters are optional.

DT=dt — Device type. If DT is omitted, the operator can assign any device type. The value of dt is a device mnemonic; for example, AY for 844-21. (Refer to 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 or 885 disk pack. Default is defined at installation time.

HT	Half-tracking; read and write alternate sectors.
FT	Full-tracking; read and write sequential sectors.

NOTE

If FT is specified, 2xPP speed must be in effect, and there must be full-track controller access to the drive on which the pack resides.

I=lfn — 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 directive: Aas,Rpru,Nrbs.

Flaw directives: Ttn,Ccn,Ssn.
Ttn,Ccn,Sfsn–lsn.

Allocation Directive	Meaning
as	Allocation style number with limits of 0 to 77 (octal) that corresponds with a number of PRUs per record block and a number of record blocks in the RBR. By using the allocation style parameter on the REQUEST statement, 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.
pru	Number of PRUs per record block (RB) size with a maximum value of 7777 (octal). For an 844 device the specified RB size must be greater than or equal to 1/32 of the physical block (PB) size and less than or equal to 32 times the PR size. For an 885 device the specified RB size must be less than or equal to 12 times the PB size. For a user device set the specified RB size must be the same on all allocation directives.
rbs	Number of record blocks in 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 in table 4-6.

The number of record blocks (RB) on the first RBR must be sufficient to hold disk tables. For a master device, the minimum number of record blocks depends on record block size, whether or not this set has a permanent file device, and the number of permanent files allowed in this set. If the number of record blocks is insufficient, LABELMS will abort and the error message will specify the table that LABELMS tried to write when it ran out of space. A subsequent ADDSET may fail due to lack of space even though LABELMS was successful.

Determine the number of record blocks the disk table requires as follows:

Disk Table Space = LBL + PFT + LFT + PFD + PFC + PAM + SDT + DSR + DAM + SMT

Mnemonic	Meaning	RBs
LBL	Device label.	1
PFT	Physical flaw table.	1
LFT	Logical flaw table.	2
PFD	Permanent file directory.	NF/(4*RBSIZE)
	NF Maximum number of files allowed in set.	
	RBSIZE Physical record units (PRU) per record block (RB).	

Mnemonic	Meaning	RBs
PFC	Permanent file catalog (refer to PFD for meanings of NF and RBSIZE).	(NF*6)/(4*RBSIZE)
PAM	PFC allocation map.	1
SDT	Subdirectory table.	1
DSR	Deadstart recovery RB.	1
DAM	Device allocation map.	2
SMT	Set member table.	1

Example:

For an 844-21 master device with a maximum of 4000 files in a set (NF) and 57 PRUs per record block (RBSIZE), calculate the number of RBs needed for disk table space as follows:

```
PFD = 4000/(4*57)
    = 18 RBs
PFC = (4000*6)/(4*57)
    = 106 RBs
Disk Table Space = LBL + PFT + LFT + PFD + PFC + PAM + SDT + DSR + DAM + SMT
                 = 1   + 1   + 2   + 18  + 106 + 1   + 1   + 1   + 2   + 1
                 = 134 RBs
```

Flaw Directive	Meaning	
tn	Track number	Limits depend on device as shown in table 4-6.
cn	Cylinder number	
sn	Sector number	
fsn	First sector number	Indicates several contiguous flaw sectors.
lsn	Last sector number	

TABLE 4-6. DEVICE DEFAULTS

Device	PB Size (PRUs)	RB Size Default (PRUs)	rbs Default	tn Limits	cn Limits	sn Limits
844-21	114†	57	3232	0 to 18	0 to 403	0 to 23
844-41	114	57††	3232††	0 to 18	0 to 807	0 to 23
885	320	160†††	3356†††	0 to 39	0 to 838	0 to 31

†This value changed from 70 to 160 with the introduction of the 844-41 devices. Only devices with the following RB sizes are compatible on both pre- and post-844-41 supporting systems.

For devices with (RB size) $\leq 70_8$, RB sizes of 2, 4, 7, 10, 16, 34, 70 are compatible with both systems.

For devices with (RB size) $> 70_8$, RB sizes such that $(2n-1)*70+1 \leq$ RB size $\leq 2n*70$, where $n=1,2...20_8$ are compatible with both systems.

This value changed from 160 to 162 at NOS/BE 1.4 PSR level 508. Refer to the following NOTE for information on compatibility.

††To create an 844-41 (double-density) pack with an RB size of 71_8, two allocation directives must be input to LABELMS. The 844-41s require two RBRs when the RB size is 71_8.

†††To create an 885 disk with an RB size of 240_8, two allocation directives must be input to LABELMS.

NOTE

User packs cannot have the number of RBs greater than the installation-defined maximum number of record blocks to be used for private devices. All members of a user device set including the master must be labeled using the same set of allocation directives.

Elimination of gap sectors on 844 devices introduces a downward incompatibility at NOS/BE 1.4 PSR level 508. If a label is written on an 844 device in a system at level 508 or later, the user cannot read or write the device in a system release level prior to level 508.

For 885 (AJ), 844-21 (AY), and 844-41 (AZ) disk drives, 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	Maximum number of blocks that can be allocated to the job, 1-377777 (octal). The maximum value allowed may be reduced by the installation. 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 nonzero 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. For example, a disk might have record blocks of 3200 words. In this instance, a statement specifying LIMIT(2) would 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 multifile 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 multifile set is required before LISTMF is called.

The format of LISTMF is:

LISTMF,M=mfn,P=p.

M=mfn — Multifile 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 multifile 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 but 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. Refer to the CYBER Loader Reference Manual for further details.

The format of LOAD is:

LOAD,lfn_1/r,lfn_2/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_i — 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 FROM TAPE)

LOADPF loads permanent files that have been dumped to tape. All files 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.

LOADPF execution causes an implicit attach of a file whose permanent file name is DUM. The device set to which files are to be loaded must contain a copy of DUM cataloged with an ID of PUBLIC and defined passwords for RD, MD, CN, and EX. If a DUM permanent file with TK=DUMPF already exists (earlier systems required this), it must be purged and replaced as described above. The EX password to access DUM must be submitted as part of the LOADPF call.

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. However, running PFLOG after each change dump and then running GENLDPF with the last log tape restores the PFC without reloading the purged files. For multivolume LOADPF jobs, NORING must be specified on a REQUEST or LABEL control statement.

The format of LOADPF is:

LOADPF,LP=x,LF=lfn$_1$,CL,SN=setname,VSN=vsn,ID=name,PF=pfn,CY=cy,I=lfn$_2$,PW=pw,IN=ddd,JN=yyddd,
LA=mmddyy,DA=yyddd,CD=mmddyy, TI=hhmm,OR.

Only PW is required. All parameters are 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 Files to be loaded:

x	Significance
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_1	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 read from 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_2	Name of directive file, 1-7 letters or digits beginning with a letter. If I is specified but not equivalenced, file INPUT is used. If a directive file is used, the following parameters are not allowed on the LOADPF statement: VSN=vsn, ID=name, PF=pfn, CY=cy, IN=ddd, JN=yyddd, LA=mmddyy, DA=yyddd, CD=mmddyy, and TI=hhmm.
PW=pw	EX password for DUM.
IN=ddd	Load files not attached within this number of days; 1-3 digits. Can be qualified by a TI parameter.
JN=yyddd	Load files not attached on or after this date; five-digit ordinal date format. Can be qualified by TI parameter.
LA=mmddyy	Load files not attached on or after this date; six-digit month-day-year format. Can be qualified by TI parameter.
DA=yyddd	Load files created, modified, renamed, or extended after this date; five-digit year-and-day-of-year format. Can be qualified by TI parameter.
CD=mmddyy	Load files created, modified, renamed, or extended after this date; six-digit month-day-year format. Can be qualified by TI parameter.
TI=hhmm	Time qualifier for date parameters; four-digit 24-hour clock format. If date parameters are not specified, TI is ignored.
OR	Allows loading of files from the dump tape with a device type or an allocation style different from that defined on the equipment set to which the files are being loaded. If OR is not specified, a file with device type or allocation style conflict is not loaded.

A group of files to be loaded can be identified by name and owner in a directive record. Directive formats are as follows:

ID=name
ID=name, PF=pfn
ID=name, PF=pfn, CY=cy

Parameters are order independent. The PF and CY parameters are optional. The ID parameter should be specified. However, if the PF parameter is specified without an ID parameter, then ID=PUBLIC is assumed.

LOADPF EXAMPLES

1. JOB1.
   REQUEST(DUMTAPE,HY,S,E)
   LOADPF(PW=EXPW)
   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,PW=EXPW)
   6/7/8/9

   This job loads all nonexpired permanent files from tape.

3. JOB3.
   REQUEST(DUMTAPE,HY,S,E)
   LOADPF(PF=STARTREK,ID=SPOCK,PW=EXPW)
   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,PW=EXPW)
   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. Refer to the CYBER 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:

MAP,{OFF / FULL / ON / PART}.

OFF	No map is produced.
FULL	Full map is produced.
ON	Map has all items except entry point map.
PART	Map has all items except entry point map and cross-reference.

The effect of a MAP can be overridden for a particular load sequence by the MAP option of the loader statement LDSET (see the CDC 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 nonexistent 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.

---

†On a CYBER 176, address range errors always result in job termination, no matter what option is specified on the MODE statement.

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 except jump to location zero.
	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.

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.

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

The period is required. The comment can begin in any column after the period; ending punctuation is not required.

comment — String of 74 characters to be displayed for the operator. Any character 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.

PFLOG (DUMP PERMANENT FILE CATALOG TO TAPE)

PFLOG dumps the permanent file catalog (PFC) of a device set to a magnetic tape (log file).

Before PFLOG is called, a REQUEST control statement must define the log file as a new labeled SI tape whose file name is LOGTAPE.

PFLOG execution causes an implicit attach of a file whose permanent file name is DUM. The device set whose PFC is to be dumped must contain a copy of DUM cataloged with an ID of PUBLIC and a defined password for RD. The RD password must be submitted as the PW parameter on the PFLOG call.

For each PFC entry dumped, PFLOG makes an output listing entry that contains the permanent file name, owner id, and cycle number.

The format of PFLOG is:

PFLOG,PW=rd,SN=setname,LF=lfn.

Only PW is required. All parameters are order independent.

PW=rd — RD password for DUM.

SN=setname — Name of device set whose PFC is to be dumped; 1-7 letters or digits beginning with a letter. The master device for this set must have been mounted before PFLOG can execute. Default is the permanent file default set.

LF=lfn — Name of file on which listing is to appear; 1-7 letters or digits beginning with a letter. Default is OUTPUT. If lfn=0, no listing is generated.

PFLOG EXAMPLES

1. JOBCARD(NT01)
   VSN(LOGTAPE=123456)
   REQUEST(LOGTAPE,NT,PE,N,RING)
   PFLOG(PW=READ,LF=0)
   6/7/8/9

   This job dumps the permanent file default set to LOGTAPE. No output listing is generated.

2. JOBCARD(NT01)
   VSN(LOGTAPE=123456)
   MOUNT(SN=SETNAME,VSN=MASTER)
   REQUEST(LOGTAPE,NT,PE,N,RING)
   PFLOG(SN=SETNAME,PW=READ)
   6/7/8/9

   This job dumps the PFC of SETNAME to LOGTAPE and prints the output listing on OUTPUT.

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 a cycle number is not specified, the cycle with the highest cycle number is purged. If there is only one cycle, 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 file is attached to the job, the format of the PURGE statement is:

PURGE,lfn,RB=1.

lfn — Local file name by which the file is attached to the job.

RB=1 — Refer to the explanation in the next form of the PURGE statement.

All other parameters are ignored. The local file remains with all permissions that were granted when the file was attached, except in the following cases:

- The file resides on a mainframe other than the one on which the job is executing.
- The file is archived.
- The user has specified the RB=1 parameter and the system has set the record block conflict flag.

If the file is not attached to the job, the format of the PURGE statement is:

PURGE,lfn,pfn,ID=name,{LC=n / CY=cy},EC=ec,PW=pw,UV=uv,RB=1,RW=p,SN=setname,ST=mmf,VSN=vsn.

Parameter pfn is required. Parameters lfn (if present) and pfn are order dependent. All other parameters are optional depending on how the file was cataloged. They are also order independent. The PURGE statement can be continued from one line to the next. The first line must not be terminated by a period or right parenthesis. To be consistent with other control statements that require such a format, the last nonblank character on the line should be separator. The continuation begins in column 1 of the next line.

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, three characters. If the file is not cataloged on the mainframe at which the job is executing, a job is generated on the specific mainframe to purge the file.

SN=setname — Device set name identifying the private device set containing the file to be purged. This parameter may be 1-7 letters or digits and must begin with a letter. If SN is specified, VSN must also be specified to allow access to the private set on the mainframe specified by ST.

VSN=vsn — Volume serial number identifying the master device of the private device set. This parameter may be 1-6 letters or digits. If SN is specified, VSN must also be specified as explained in the SN description.

Refer to the ATTACH control statement for the meaning of the remaining parameters.

The system issues an ATTACH (or GETPF) using the parameters specified in the PURGE statement. It then purges the file. The local file remains as explained in the previous format of the PURGE statement.

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,V=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.

V=vsn — Volume serial number of device set master device, 1-6 letters or digits with leading zeros assumed.

In a multimainframe environment, permanent files on a shared device set 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 entry.

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,ECS. releases the ECS field length currently assigned to the job.

This control statement should be used whenever the job no longer requires special field length handling in CM or ECS.

The formats of REDUCE are:

REDUCE.	Decreases CM field length.
REDUCE,ECS.	Releases ECS field length.

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	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, and 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 first control statement that references the file or executes a 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. However, 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	Name by which file will be known throughout the job, 1-7 letters or digits beginning with a letter. lfn beginning with ZZ is reserved for the system. lfn cannot be OUTPUT. With private device sets, lfn also cannot be PUNCH, PUNCHB, P80C, FILMPR, FILMPL, PLOT, HARDPL, or HARDPR.
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 nine-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 (seven- and nine-track) including multifile 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 multifile 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. Refer to 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 and reports any discrepancy 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

To use a REQUEST statement for tape files the JOB statement must specify the tape track type and density (refer to the MTk parameter in the JOB statement earlier in this section). 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 seven-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 multifile sets, or any 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 nonscratch 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 seven- or nine-track tape is acceptable, an MN parameter can be used in place of MT or NT. The resulting tape has default density. However, to ensure that the job is not aborted because of maximum tape units exceeded, the job statement should specify both MT and NT. 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, which 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.

SEVEN-TRACK TAPE PARAMETERS:

$$\text{REQUEST,lfn,MT,}\left\{\begin{matrix}\text{LO}\\\text{HI}\\\text{HY}\end{matrix}\right\},\left\{\begin{matrix}\text{CK}\\\text{MF}\end{matrix}\right\},\left\{\begin{matrix}\text{S}\\\text{L}\end{matrix}\right\},\left\{\begin{matrix}\text{U}\\\text{Y}\\\text{Z}\end{matrix}\right\},\left\{\begin{matrix}\text{E}\\\text{N}\\\text{NS}\end{matrix}\right\},\left\{\begin{matrix}\text{IU}\\\text{SV}\end{matrix}\right\},\left\{\begin{matrix}\text{RING}\\\text{NORING}\end{matrix}\right\},\text{NR,VSN=vsn.}$$

File name:

If the MF parameter is not specified, lfn is the file name of 1-7 letters or digits beginning with a letter.

If the MF parameter is specified, this parameter is a multifile set name of 1-6 letters or digits beginning with a letter.

The multifile set name cannot be used in any input/output statement except as the M parameter in a LABEL statement or POSMF macro.

Seven-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. The MTk parameter must be specified in the JOB statement. Refer to the JOB statement earlier in this section.

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 used to read both the label and the data, except as indicated under Z in the Z parameter description later in this section. 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.

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.

†The 667/677 tape units can read but not write at 200 bpi.

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 multifile set.

absent — Neither of the above is assumed.

Data format:

S — S tape format.

L — L tape format.

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 nonstandard 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. Nonstandard labels are not supported on SI tapes.

NINE-TRACK TAPE PARAMETERS:

A declaration of NT or a nine-track density for a tape to be written is required to identify a nine-track tape and a declaration of NT, GE, PE, or HD is required on the JOB statement. Refer to the JOB statement earlier in this section. Definitions and conditions for all except the density and data format parameters are the same as those for seven-track tape.

$$\text{REQUEST,lfn,NT,}\left\{\begin{matrix}\text{PE}\\\text{HD}\\\text{GE}\end{matrix}\right\},\left\{\begin{matrix}\text{S}\\\text{L}\end{matrix}\right\},\left\{\begin{matrix}\text{CK}\\\text{MF}\end{matrix}\right\},\left\{\begin{matrix}\text{U}\\\text{Y}\\\text{Z}\end{matrix}\right\},\left\{\begin{matrix}\text{E}\\\text{N}\\\text{NS}\end{matrix}\right\},\left\{\begin{matrix}\text{US}\\\text{EB}\end{matrix}\right\},\left\{\begin{matrix}\text{IU}\\\text{SV}\end{matrix}\right\},\left\{\begin{matrix}\text{RING}\\\text{NORING}\end{matrix}\right\},\left\{\begin{matrix}\text{EEC}\\\text{IEC}\end{matrix}\right\},\text{NR,VSN=vsn.}$$

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.

HD	**800 cpi**
PE	**1600 cpi**
GE	**6250 cpi**
absent	Tape written at installation-declared density

Data format:

S	S tape format.

L† **L tape format.**

absent **Data format is SI format.**

Hardware error correction:

EEC Enable hardware GE write error correction. The system allows certain types of single-track errors to be written that can be corrected when the tape is read (on-the-fly correction). This is the recommended mode of operation, because it provides efficient throughput, error recovery, and tape usage when writing GE tapes on media that is suitable for use at 3200 fci or 6250 cpi.

IEC Disable all error correction activity in GE write mode. The system invokes standard error recovery processing when an on-the-fly error occurs while writing a GE tape. The system erases the defective portion of tape, thereby reducing the amount of data that can be stored on the tape. Only tape that is suitable for recording at 6250 cpi should be used when this mode of operation is in effect.

NOTE

EEC and IEC apply only to GE (6250 cpi) operations. GE must also be specified in a REQUEST statement; otherwise, EEC and IEC are ignored.

EEC and IEC are applicable if the user requests default nine-track density and the installation nine-track default density is GE (6250 cpi).

Code for conversion of all nine-track tape labels and of data on coded nine-track tapes of type S or L (refer to tape 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, nine-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.

†Currently L tapes are supported only on seven-track tape devices and 669/679 nine-track tape drives.

2. REQUEST(FILE1,*MT,RING)

   Depending on installation option, the system automatically assigns FILE1 to a scratch tape on a seven-track tape unit. The file is unlabeled and written in SI data format at an installation-declared density.

3. REQUEST(STANF27,HI,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 556 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 and 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 request device to the job.

REQUEST,lfn,dt.

lfn — 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
LR	580-12 line printer	HC	253-2 hardcopy recorder
LS	580-16 line printer	FM	254-2 microfilm recorder
LT	580-20 line printer	TR	Paper tape reader
CR	405 card reader	TP	Paper tape punch
CP	415 card punch	PL	Plotter

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 — 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 or 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=setname,VSN=vsn.

The first parameter must be lfn. Other parameters are optional and order independent.

lfn — 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:

A*	Any mass storage device
AH	819 disk drive (CDC CYBER 170 Model 176 only)
AJ	885 disk drive
AY	844-21 disk drive
AZ	844-41 disk drive

aa — Allocation style, 0 to 77 (octal), 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 and queue files are assigned only to permanent file devices and queue devices, respectively; otherwise, files 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 specified and requested device is unavailable 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.

EC	Default buffer size.
ECnnnn or ECnnnnK	Buffer size of nnnn 60-bit words multiplied by 1000 (octal).
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. (*PF can be used instead of PF and has the same meaning.)

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. (*Q can be used instead of Q and has the same meaning.)

SN=setname — Assign file to setname, 1-7 letters or digits beginning with a letter. If omitted, file is assigned to a public device set. If only SN is specified, setname is that specified by SETNAME control statement; if setname has not been specified previously, file is assigned to a public device.

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 and 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).

Refer to DEVICE SETS in section 3 for more examples and explanations of REQUEST.

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.

†All file types will be buffered for device type AH (CDC CYBER 170 Model 176 only).

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. The output file of the checkpointed job, unless redirected by a ROUTE command, will return to the source of the RESTART job.

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 the original job was allowed. An RFL control statement may be needed to give the RESTART routine sufficient memory to operate. 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,lfn_1,lfn_2, ...

More than one file or multi-file set can be specified; only one is required.

lfn_i Name of file to be returned, 1-7 letters or digits beginning with a letter or name of multi-file set tape to be returned, 1-6 letters or digits beginning with a letter. lfn_i cannot be INPUT.

For magnetic tape output files, RETURN causes trailer labels to be written and the file to be rewound and then unloaded. With the exception of members of a multifile 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 members of a multifile set, the tape units on which the files reside are not disassociated from the job. The multifile set is left without a member as it was immediately after the initial REQUEST was made for the master file.

For master multifile 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 multivolume tapes, a REWIND rewinds the current volume and a subsequent forward motion initiates a backward reel swap, positioning the file at its beginning.

For labeled multivolume, single-file tapes, a REWIND rewinds the current volume and sets the volume number in the system tables 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 multifile tapes, a REWIND rewinds the specified file to its beginning. If necessary, the operator is instructed to mount the previous volume.

The format of REWIND is:

REWIND,lfn_1,lfn_2, ...

More than one file can be specified; only one is required.

lfn_i 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.

A REWIND that references an lfn that is not local to the job creates an empty file of the same name.

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)

The RFL control statement requests a specific field length in central memory (CM) or extended core storage (ECS).

A job's field length requirement in central memory usually varies with each job step. For example, a FORTRAN compilation might require 45 000 words (octal) while a COPY routine might require only 5000 words (octal). Normally, the dynamic field length management of the system automatically varies the field length assigned in order to optimize use of system storage.

If a job has special requirements for which specific job steps require considerably more or less field length than the field length manager would assign, the user can override the system assignment by including an RFL control statement. When used with the CM parameter, this statement inhibits dynamic field length management by the system and assigns a user-specified field length to the job. This user-specified field length remains in effect until a REDUCE control statement again activates the system's dynamic field length management. Thus a REDUCE statement should immediately follow each RFL statement unless the user wants his field length specification to remain in effect for succeeding job steps.

The RFL control statement also specifies field length in ECS. This does not affect dynamic field length management by the system which applies only to central memory.

Once an ECS field length is assigned, it remains in effect until released by a REDUCE,ECS. control statement. Thus, the REDUCE,ECS. statement should immediately follow the last job step that uses the ECS assignment.

The formats of RFL are:

RFL,fl.

RFL,CM=fl.

RFL,EC=fle.

RFL,CM=fl,EC=fle.

fl	New CM field length (octal). Maximum value is established either by the CM parameter on the job statement or, if that parameter is omitted, by the installation default. The fl parameter must be specified; there is no default.
fle	New ECS field length in multiples of 1000 words (octal). Maximum value is established either by the EC parameter on the job statement or, if that parameter is omitted, by the installation default. If the installation is using ECS, the fle 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 nonallocatable 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, and so on.
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, binary, or transparent (INTERCOM 5) 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 file names, 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 as follows:

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:

ROUTE,lfn,DEF, DC=dc, EC=ec, FC=fc, FID=fid, IC=ic, PRI=pri, REP=n, SC=nn, ST=mmf, TID=tid.

Only lfn is required. All other parameters are optional and order independent.

lfn — 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.

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.

If DEF is used to dispose the file OUTPUT to a destination other than the job's origin, a copy of the dayfile is sent to the job's origin at end-of-job.

†Supporting software must be supplied by the installation.
††Depends on installation parameter.

DC=dc

File disposition:

SC	Evict the file	FR†	Print on microfilm recorder
PR	Print on any available printer	FL†	Plot on microfilm recorder
		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. If dc is not specified, and lfn is not a special file name such as OUTPUT, PUNCH, and so forth, DC=SC is assumed.

EC=ec

External characteristics of the print or punch file. If EC is not specified, default EC code is used.

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.
The print trains normally mounted for output from INTERCOM terminals are:

INTERCOM	
BCD 200UT	B6
ASCII 200UT	A6
730 series batch terminal	A6
711 and 714 terminal	A9
Others	A6

Punch Files:

O26 (or 026)	Punch format 026
O29 (or 029)	Punch format 029
ASCII	Punch format ASCII (INTERCOM files only)
SB	Punch format standard binary
80COL	Punch format 80 column free-format binary

†Supporting software must be supplied by the installation.

Default value for JANUS punch files is 026 or 029 depending on installation option. No standard binary punching is available with INTERCOM.

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. If FC is not specified, standard forms are used.

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.

If fid is not specified, 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.
TRANS	File format is transparent (INTERCOM).

IC=DIS is required by JANUS for all print files except where EC=A9, in which case, IC=ASCII is required. IC=BIN is required for binary punch files. IC=TRANS can be specified only at HASP or 2780/3780 terminals. If IC is not specified, IC=DIS is assumed.

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. If pri is not specified, file receives standard priority.

REP=n — Repeat count for output files, $n \leq 37_8$. If n is not specified, there are zero extra copies.

SC=nn — Spacing code for output sent to a 580 PFC printer. nn is an octal value, 0 to 77_8, 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. If nn is not specified, SC=0 is assumed.

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 any 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. If mmf is not specified, process the file on the system where it originated.

TID=tid — INTERCOM terminal identification. File is to be returned to terminal identified. If tid=C, file is to be output at central site. If tid is not specified, file is to be returned to the site or terminal where the job originated.

ROUTE EXAMPLES

1. job statement
   REQUEST(LOON,Q)
   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 the job aborts, the file is scratched. If the job terminates normally, file LOON is printed after the operator mounts the 95-character print train. 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
```
REQUEST (SWALLOW,Q)
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
```
REQUEST(FALCON,Q)
COPYBF(INPUT,FALCON)
ROUTE(FALCON,DC=IN,ST=ABC)
7/8/9
HAWK,T100.
REQUEST(OWL,Q)
COPYBF(INPUT,OWL)
REWIND(OWL)
REQUEST(EAGLE,Q)
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
```
REQUEST(SWIFT,Q)
COPY(INPUT,SWIFT)
ROUTE(SWIF,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,PW=pw,RD=rd,RP=rp,ST=mmf,
TK=tk,XR=xr,SN=setname,VSN=vsn.

The lfn and pfn 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. 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	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.

SN=setname — Device set name identifying the private device set containing the file to be made a permanent file. This parameter may be 1-7 letters or digits and must begin with a letter. If SN is specified, VSN must also be specified to allow access to the private set on the mainframe specified by ST.

VSN=vsn — Volume serial number identifying the master device of the private device set. This parameter may be 1-6 letters or digits. If SN is specified, VSN must also be specified as explained in the SN description.

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,ART command. The user then enters the following statement to make the file permanent under SCOPE 2: SAVEPF,ART,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,ART,RT=Z,BT=C,FL=80.
ATTACH,ART,ID=XX.
FTN5,I=ART
LGO.
```

With the SAVE and BATCH commands, the user makes the file local and then submits the job. The program on file ART is attached, compiled, and executed. The job aborts if the FILE statement is not included, since the FORTRAN compiler would expect W type records.

Refer to the CYBER Record Manager manuals and the SCOPE 2 Operator's Guide for additional details on file conversion requirements.

Refer to 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 should not be used with CYBER Record Manager file organizations unless RT=S. For S and L tapes SKIPB recognizes only levels 0 and 17_8 and treats any other level as a level 0.

The format of SKIPB is:

SKIPB,lfn,n,lev,mode.

Parameters are positional; only lfn is required.

lfn — 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. If n is set to zero, the system will treat it as n=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 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 should not be used with CDC 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	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. If n is set to zero, the system treats it as n=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.

†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_8 is specified, a skip to end-of-partition is performed. Any other level number is assumed to be zero, and one record is skipped.

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 request system bulletins to the OUTPUT file.

The format of SYSBULL is:

SYSBULL,p_1,p_2, . . . ,p_n.

Parameters are all optional.

p_i 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, all dependent jobs should be submitted before independent jobs. Jobs can be submitted from the central site, or from 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 (multi-programming).

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,job_1,job_2, ...

Multiple job names or multiple TRANSF control statements can be used.

job_i	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. The operator decides whether 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)
.                              .
.                              .
.                              .
6/7/8/9                        6/7/8/9

JOBC,DJS01.                    JOBD,DJS02.
execution call                 execution call
TRANSF(JOBD,JOBE)              TRANSF(JOBF)
6/7/8/9                        6/7/8/9

JOBE,DJS01.                    JOBF,DJS03.
execution call                 execution call
TRANSF(JOBF)                   6/7/8/9
6/7/8/9
```

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 a DUM permanent file with TK=DUMPF already exists (earlier systems required this), it must be purged and replaced as described above. 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=setname$_1$,TS=setname$_2$,FM=vsn$_1$,TM=vsn$_2$,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, modify, and extend passwords, separated by commas, if defined for permanent file DUM. If passwords have been defined for file DUM, all must be specified with this parameter or the utility aborts. No default exists.

FS=setname$_1$ — 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=setname$_2$ — 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=vsn$_1$ — 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=vsn$_2$ — 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 successfully 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,B,C,D)
   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. 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 passwords A,B,C,D on device set TEST.

2. SECOND.
   MOUNT(SN=TEST1,VSN=555) Mount master.
   TRANSPF(FS=TEST1,TS=TEST1,FM=888,TM=222,PW=Q,R,S,T)
   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. 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.

   The central site operator receives notification when a master-to-member table move is attempted and must authorize continuance of the transfer. The record block size of the first record block reservation table (RBR) of the FM and TM devices must be the same (refer to the rbs allocation directive of the LABELMS statement in this section).

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(lfn_2,SN=$setname_2$)
ATTACH(lfn_1,pfn,ID=owner,SN=$setname_1$)
COPY(lfn_1,lfn_2)
CATALOG(lfn_2,pfn,ID=owner)
RETURN(lfn_1,lfn_2)

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 and 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 insufficient 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=PW1,PW2,PW3,PW4)
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,lfn_1,lfn_2, ...

More than one file or multifile set can be specified; only one is required.

lfn_i — Name of file to be unloaded, 1-7 letters or digits beginning with a letter. Can be a member of a tape multifile set. If INPUT is specified, an error message is issued and INPUT is rewound but not unloaded.

Name of multifile 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 with 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 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,lfn_1=vsn_1,lfn_2=vsn_2, ...

One statement can be used for any number of files. Multiple VSN control statements can be used. The VSN statement can be continued from one line to the next. The last nonblank character on the line to be continued must be a separator. The continuation begins in column 1 of the next line.

lfn_i — For a single file, the file name of 1-7 letters or digits beginning with a letter.

For a multifile set, the multifile set name of 1-6 letters or digits beginning with a letter.

vsn_i — 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 multivolume 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 file name is to be reused 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.

VSN EXAMPLES

1. 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.

```
JOB5,MT1.
VSN(TAPE1=1234)
REWIND,TAPE1.
```

2. To have a specific magnetic tape assigned to the job, either of the following requests would suffice.

```
JOB6,MT1.
VSN(TAPE1=1234)
REQUEST(TAPE1,MT,E,NORING)
```

```
JOB7,MT1.
REQUEST(TAPE1,VSN=1234,MT,E,NORING)
```

3. A 679 GCR unit with any tape mounted that meets the installation criteria for a scratch tape is assigned.

```
JOB8,GE1.
VSN(TAPE1=0)
REQUEST(TAPE1,GE,N,IU)
```

4. A 669 or 679 unit with any tape mounted that meets the installation criteria for a scratch tape is assigned.

```
JOB9,PE1.
VSN(TAPE1=0)
REQUEST(TAPE1,PE,N,IU)
```

5. The magnetic tape with VSN of 1234 is assigned to the job and the subsequent reel has a VSN of 5000.

```
JOBA,MT1.
VSN(TAPE1=1234/5000)
REQUEST(TAPE1,MT,E,NORING)
```

6. A magnetic tape with a VSN of 1234 or 5000 is assigned to the job. If subsequent reels are needed, the first tape's EOV2 label or a VSN entered by the operator identifies the reel.

```
JOBB,MT1.
VSN(TAPE1=1234=5000)
REQUEST(TAPE1,MT,E,NORING)
```

CYBER CONTROL LANGUAGE 5

CYBER Control Language (CCL) is a set of job control statements, functions, and commands, that permit the user to:

- Control and alter the processing sequence of job control statements.
- Perform externally defined control statement sequences (procedures) during job execution.
- Determine and test the attributes and physical residence of files and use that information to determine the job processing sequence.

These capabilities are in the form of a procedure-oriented language that contains conditional and unconditional skipping, looping, procedure-calling, and displaying statements. CCL also provides a number of registers that the user can set, alter, and test.

The first subsection is an overview of the CCL statements, functions, and commands. The following three subsections describe the CCL statement syntax and the operators and operands that make up a CCL expression. The remainder of the subsections describe the CCL statements. The statement descriptions are organized according to shared attributes.

OVERVIEW

The following paragraphs briefly describe all CCL statements, functions, and commands. The descriptions are grouped according to functional use.

These CCL statements initiate or terminate the skipping of control statements.

Statement	Description
IFE	Evaluates a conditional expression. If its expression is true, the next statement is processed. If its expression is false, statements are skipped until a matching ELSE or ENDIF statement is found and the statements following the ELSE or ENDIF are processed.
SKIP	Skips statements until a matching ENDIF statement is found.
ELSE	Terminates or initiates skipping of statements following an IFE statement. If an IFE statement is true, the ELSE statement initiates skipping to the matching ENDIF statement; if it is false, the ELSE statement stops the skipping initiated by the false IFE expression.
ENDIF	Terminates skipping of statements initiated by a matching IFE, SKIP, or ELSE statement.

These CCL statements identify a sequence of control statements as a loop that can be repeatedly processed.

Statement	Description
WHILE	Establishes the beginning of the loop. If the associated expression is true, the loop is processed; if it is false, the loop is not processed.
ENDW	Establishes the end of the loop.

These CCL statements assign and display values associated with CCL symbolic names. A CCL symbolic name is an alphanumeric character string that is recognized by CCL and has an assigned value. The value is assigned by the system or the user.

Statement	Description
SET	Allows the user to assign values to CCL symbolic names.
DISPLAY	Evaluates an expression and displays the result in the job dayfile in both octal and decimal. In INTERCOM, the result is displayed at the terminal and in the job dayfile.

These CCL functions are used in expressions within the CCL statements. They can determine the conditions for transfer of control.

Function	Description
FILE	Determines the attributes of a file.
DT	Determines the type of device on which a file resides.
NUM	Determines whether or not a parameter has a numeric value.

These CCL statements initiate and terminate processing of a procedure. A procedure is a group of control statements (including CCL statements) that is separate from the job control record.

Statement	Description
BEGIN	Initiates processing of a procedure.
REVERT	Returns processing from a procedure to the control statement record or procedure that called it. Job processing continues with the control statement following the BEGIN statement.

This CCL statement identifies a procedure. A procedure is a sequence of control statements executed similarly to a subroutine. A procedure is called from the job control record, from a user at an interactive terminal, or from other procedures. It consists of any number of control statements, and is physically a single record on a file. The user can define any number of procedures to reside on a single file. Procedure files can be optionally put on user and system libraries.

Statement	Description
.PROC	Identifies the statements that follow .PROC as a procedure. There are two types of procedures; interactive and noninteractive. Both procedures use the .PROC statement.

These CCL statements allow descriptions of the procedure and its parameters to be displayed. They may only be used in interactive procedures.

Statement	Description
.HELP	Establishes the beginning of descriptive text for the procedure and its parameters.
.ENDHELP	Establishes the end of descriptive text for a procedure.

These CCL commands control processing of data within a procedure.

Command	Description
.DATA	Creates a data file containing information to be used by the noninteractive procedure in which it appears.
.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.

STATEMENT SYNTAX

The following CCL statement syntax rules are similar to the syntax rules of other control statements.

- A comma or left parenthesis must separate the statement name and the first parameter.
- Commas must separate consecutive parameters.
- A period or a right parenthesis must terminate the statement.
- Parentheses can nest expressions within expressions (parentheses do not imply multiplication).
- A right parenthesis ending an expression within a statement cannot also serve as the statement terminator. The user must include an additional right parenthesis or period to complete the statement.
- Comments can follow the statement terminator.

CCL statements may be longer than 80 characters. They may extend over more than one line if each line to be continued contains no more than 80 characters and ends with a comma. Lines in an interactive - procedure header statement do not have to end with a comma.

OPERATORS

Operators separate operands in a CCL expression. There are three types of CCL operators: arithmetic, relational, and logical. Operators are used in the expressions within the IFE, WHILE, DISPLAY, and SET statements, and within the FILE function.

ARITHMETIC OPERATORS

Integer arithmetic is used in each step of the evaluation of a CCL expression. Integer division truncates any remainders and no rounding occurs. Division, multiplication, and exponentiation produce a zero result if the absolute value exceeds $2^{48} - 1$. Computations are accurate to 10 decimal digits (20 octal digits) and overflow is ignored.

The following are the CCL arithmetic operators.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Leading -	Negation
Leading +	Ignored

RELATIONAL OPERATORS

A relational operator produces a value of 1 if the relationship is true, and 0 if it is false. The following are the CCL relational operators (either form may be used).

Operator	Operation
= or .EQ.	Equal to
.NE.	Not equal to
< or .LT.	Less than
> or .GT.	Greater than
.LE.	Less than or equal to
.GE.	Greater than or equal to

LOGICAL OPERATORS

When a CCL expression contains a logical operator, CCL evaluates each operand as true (nonzero) or false (zero). The following are the CCL logical operators.

Operator	Operation
.EQV.	Equivalence
.OR.	Inclusive OR
.AND.	AND
.XOR.	Exclusive OR
.NOT.	Complement

ORDER OF EVALUATION

Operators in an expression are evaluated in the following order:

1. Exponentiation
2. Multiplication, division
3. Addition, subtraction, negation
4. Relations
5. Complement
6. AND
7. Inclusive OR
8. Exclusive OR, equivalence

Operators of equal order are evaluated from left to right.

OPERANDS

One or more operands separated by operators make up a CCL expression. Expressions are used within the IFE, WHILE, DISPLAY, and SET statements and within the FILE function. An expression within an expression must begin with a left parenthesis and end with a right parenthesis. There is no limit on the length of an expression, except that a period or a right parenthesis (not acting as a statement terminator) must appear within the first 50 operands. Expressions can contain operands of one or more types. There are three types of operands: constants, symbolic names, and functions.

CONSTANTS

CCL uses two types of constants, numeric constants and literals.

A numeric constant is a string of 1 to 10 numerals that CCL processes as an interger. All characters within the string must be digits (0 through 9), except the final character, which may be a postradix B or D. A B postradix identifies an octal integer; a D postradix identifies a decimal integer. If no postradix is specified, decimal is assumed.

A literal is a string of alphanumeric characters delimited by dollar signs. When a literal is used as a numeric constant, the string can be from 1 to 10 alphanumeric characters, and the system treats the binary value of the display code string as a right-justified integer.

A dollar sign is represented within a literal by two dollar signs. A literal must have an even number of dollar signs. A literal may be used alone or in combination with other characters to form parameter values, as shown in the following list.

Parameter Specified	Value Produced
$$	null
$$$$	$
$A.B$	A.B
A$.$B	A.B
A$.B$	A.B
$A.$B	A.B

SYMBOLIC NAMES

A symbolic name is a string of characters that is recognized by CCL and has an assigned value. CCL uses symbolic names in tests for conditions. CCL can also display the value assigned to a symbolic name.

The value assigned to a symbolic name is specified by the installation or set either by the user or by CCL. All variable symbolic names have an initial value of 0 except OT (job origin type), SYS (host operating system), VER (operating system version number), EM (the current exit mode set by the user with the MODE statement), and TIME (the current time of day).

The symbolic names used with the FILE and DT functions are listed with the descriptions of the functions later in this section. The following symbolic names can be used in CCL expressions. They are grouped according to a shared attribute.

- Symbolic names whose values are passed to, but not from, a procedure (refer to Procedures, later in this section). When a procedure reverts, symbolic names are restored to the values they held when the procedure was called (refer to SET Statement, later in this section).

Name	Description
DSC	Flag determining whether skipped control statements are entered in the dayfile.
EF	Previous error flag.

Name	Description
R1	Control register 1 contents.
R2	Control register 2 contents.
R3	Control register 3 contents.

- Symbolic names whose values can be set by the user. All except EM are set by the SET control statement or the SETJCI macro (refer to section 7).

Name	Description
DSC	Flag determining whether skipped control statements are entered in the dayfile.
EF	Previous error flag.
EFG	Global error flag.
EM	Current exit mode (refer to MODE control statement, section 4). In the CYBER 170 series, EM is a four-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.
R1	Control register 1 contents.
R1G	Global control register 1 contents.
R2	Control register 2 contents.
R3	Control register 3 contents.

- Symbolic names whose values are set by the operating system.

Name	Description
DSC	Flag indicating that skipped control statements are to be entered in the dayfile.
EF	Previous error flag.
FL	Current central memory (CM) field length.
MFL	Maximum CM field length.
MFLL	Maximum extended core storage (ECS) field length.
OT	Job origin type.
SYS	Host operating system.
TIME	Current time of day (hhmm).
VER	CCL version number.

- Symbolic name whose value is set by the calling or termination of a procedure.

Name	Description
PNL	Procedure nesting level (0 when processing the original job control statement record, 1 when processing a first level procedure, and so forth). Its maximum value is 50.

- Symbolic name whose value can be set by the termination of a procedure (refer to SET Statement, later in this section).

Name	Description
EFG	Global error flag.

- Symbolic names that correspond to error code values. In an expression, a user typically checks the error flag (EF) for a nonzero value; a nonzero value indicates an error, and a zero value indicates no error. For detailed error examination, the user can compare EF with a particular symbolic name or its error code value. Users are encouraged to use the symbolic name, because the numeric values could change in future releases of NOS/BE. The following list contains the errors that allow exit processing.

Name	Value	Description
ARE	1	Arithmetic error.
CPE	4	Central processor unit (CPU) abort.
ESE	9	ABORT macro with S option initiated a search for an EXIT,S. statement.
MNE	5	Monitor call error.
MSE	8	Mass storage error.
ODE	11	Operator drop.
PPE	3	Peripheral processor (PP) abort.
TLE	6	Time limit.

- Symbolic names with fixed values. Usually these symbolic names are compared with the OT value within an expression.

Name	Description
BCO	Local batch origin.
EIO	Remote batch origin.
SYO	System origin.
TXO	Time-sharing origin.

- Symbolic names with fixed values. Usually these symbolic names are compared with the SYS value within an expression.

Name	Description
NOSB	NOS/BE Operating System.
SC2	SCOPE 2 Operating System.

- Symbolic names with true or false values. True is 1; false is 0.

Name	Description
F	Fixed value of 0.
FALSE	Fixed value of 0.
SWn	One of six sense switches (n is a number from 1 to 6). Their values are set by the SWITCH statement (refer to section 4).
T	Fixed value of 1.
TRUE	Fixed value of 1.

FUNCTIONS

Functions are used in CCL statements as expressions or operands within expressions. Functions are not control statements. The CCL functions are FILE, DT, and NUM.

File Function

The FILE function determines whether a file has a specified attribute. The system returns a value of true (1) or false (0) depending upon whether or not the file has or does not have the specified attribute(s). Only the equipment number (EQ) and file ID attributes can return values other than 1 or 0. The list of file attributes follows the description of the FILE function format.

The FILE function must be used as an expression or as a part of an expression in a CCL statement. A left parenthesis must appear before the file name, a comma must appear between the file name and the expression, and a right parenthesis must appear after the expression.

The format of FILE is:

FILE(lfn,exp)

lfn	File name for which attributes are being determined.
exp	Either a special FILE function attribute or an expression consisting of logical operators and special FILE function attributes.

The expression within a FILE function cannot include the NUM function or another FILE function; the DT function or the following symbolic names can be used within the expression. Any other symbolic name within the expression is treated either as an implicit DT function (refer to DT Function, which follows) or an unidentified variable.

Name	Description
AS	File is attached to the user's job (that is, NOS/BE recognizes the lfn of the file; the file exists).
BOI	File is positioned at the beginning-of-information. This is useful only if the sequential file is on mass storage.
EN	File has extend permission.
EOF	Last operation was a forward operation that encountered an EOP, and the file is now positioned at that EOP. This is useful only if the sequential file is on mass storage.
EOI	Last operation was a forward operation that encountered an EOI, and the file is now positioned at that EOI. This is useful only if the sequential file is on mass storage.
IN	File INPUT.
LB	File is on a labeled tape.
LO	File type is local. The file is a temporary (scratch) file; attached permanent files are not considered local in this context.
MD	File has modify permission.
MS	File is on mass storage.
OP	File is opened.
PF	File is an attached permanent file.
PH	File type is punch.
PR	File type is print.
RD	File has read permission.
TP	File is on magnetic tape.
TT	File is connected to a terminal.
WR	File has extend permission.

Example:

The following job segment shows the FILE function being used inside an IFE control statement to determine if file DATA is attached to the user's job. If DATA is not attached, the IFE statement is true and the system attaches file DATA. If DATA is attached, the IFE statement is false and the system skips to the ENDIF control statement. In both cases DATA is copied to ITEM.

```
IFE,FILE(DATA,.NOT.AS),ATTACH.
ATTACH(DATA,ID=FRAN2)
ENDIF,ATTACH.
COPY(DATA,ITEM)
```

DT Function

The DT function determines the device type on which a file resides. DT can be used only within a FILE function expression. The value of the DT function is true if the two-character mnemonic included in the function (dt) is equal to the two-character device type mnemonic of the file (lfn). The operating system defines the mnemonics. (Refer to Device Type, section 6, for a list of valid device type mnemonics.)

The format of DT as used in FILE is:

FILE(lfn,DT(dt))

lfn Name of the file for which device residence is being determined.

dt A two-character mnemonic identifying the device.

CCL assumes that any two-character symbol within a FILE function that is not a FILE function symbolic name is an implicit DT function. For example, both

DISPLAY,FILE(TAX,NT).

and

DISPLAY,FILE(TAX,DT(NT)).

test if file TAX is on a nine-track magnetic tape drive. If TAX is on a nine-track tape, a value of 1 (true) is displayed. If it is not on a nine-track tape, a value of 0 (false) is displayed.

Example:

If PATCH resides on a 9-track tape, the following job segment copies PATCH to a file on a mass storage device and catalogs it.

```
IFE,FILE(PATCH,DT(NT)),CATALOG.
REWIND,PATCH.
REQUEST,TEMP,PF.
COPY,PATCH,TEMP.
RETURN,PATCH.
CATALOG,TEMP,PATCH,ID=FIXIT.
ENDIF,CATALOG.
```

NUM Function

The NUM function determines whether or not a character string is numeric. It evaluates the character string as true (1) if it is numeric, or false (0) if it is not. NUM must be used as an expression, or as part of an expression, in a CCL statement.

The format of NUM is:

NUM(c)

c A string of 1 to 40 characters. If the string contains any nonalphanumeric characters, it must be delimited by dollar signs (for example $6:15 p.m.$) and is evaluated as nonnumeric.

Example:

The following procedure uses the NUM function to ensure that the passed parameter, NUMBER, is numeric. If a nonnumeric value is passed, the procedure reverts and aborts and processing searches for an EXIT,S. statement in the job control statement record.

```
.PROC,PROC1,NUMBER.
IFE,NUM(NUMBER),QUIT.
WHILE,R1.LE.NUMBER,LOOP.
    .
    .
    .
SET,R1=R1+1.
ENDW,LOOP.
REVERT.  PROC1 PROCESSING COMPLETE
ENDIF,QUIT.
REVERT,ABORT.  NONNUMBERIC PASSED TO PROC1
```

CONDITIONAL STATEMENTS

The following conditional control statements bracket groups of other control statements to be conditionally processed or skipped.

IFE
SKIP
ELSE
ENDIF

All conditional statements require a label string parameter. The label string consists of 1 to 10 alphanumeric characters, beginning with an alphabetic character. An IFE, SKIP, or ELSE statement (with a label string) initiates skipping, and skipping continues until CCL encounters an ELSE or ENDIF statement (ELSE only used in conjunction with IFE) with a label string matching the label string of the statement initiating the skipping. 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 procedure (covered later in this section), CCL skips all remaining statements in the 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 can 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 STATEMENT

The IFE statement conditionally initiates the skipping of succeeding statements. If the expression in the IFE statement is true, the next statement is processed. If the expression is false, CCL skips statements until it encounters a matching ELSE or ENDIF statement.

An IFE statement must have an ELSE or ENDIF statement with a matching label string. If the IFE statement is in a procedure, the ELSE or ENDIF statement must also be in that procedure.

The format of IFE is:

IFE,exp,ls.

exp — A CCL expression. The separator following exp must be a comma.

ls — Label string; 1 to 10 alphanumeric characters beginning with an alphabetic character.

Example 1:

In the following job segment the IFE control statement is used to check if it is after 6 p.m. If it is, a comment giving the time is entered in the job dayfile. Whether it is after 6 p.m. or not, job processing continues with the statement following the ENDIF statement.

```
IFE,TIME.GT.1800,HOME.
COMMENT.  GO HOME.  IT IS TIME.
ENDIF,HOME.
COMMENT.  JOB PROCESSING CONTINUES HERE
```

Example 2:

The following procedure file is a permanent file called COLORPR. It uses the IFE statement to determine if the color the BEGIN statement substituted for COLOR is red or blue. Different processing is done for the colors red and blue. Any other color is ignored.

```
.PROC,A,COLOR.
IFE,$COLOR$.EQ.$RED$,L1.
COMMENT.  PROCESSING DONE FOR RED
REVERT.
ENDIF,L1.
IFE,$COLOR$.EQ.$BLUE$,L2.
COMMENT.  PROCESSING DONE FOR BLUE
REVERT.
ENDIF,L2.
COMMENT.  NO PROCESSING DONE IF COLOR
COMMENT.  IS NOT RED OR BLUE
```

The following control statements call procedure A.

```
ATTACH,COLORPR,ID=PAINT.
BEGIN,A,COLORPR,BLUE.
BEGIN,A,COLORPR,RED.
BEGIN,A,COLORPR,PINK.
```

The following dayfile segment results when the preceding control statements are processed.

```
11.36.47.ATTACH,COLORPR,ID=PAINT.
11.36.47.PFN IS
11.36.47.COLORPR
11.36.48.AT CY= 002 SN=SPFSET
11.36.48.BEGIN,A,COLORPR,BLUE.
11.36.48.IFE,$BLUE$.EQ.$RED$,L1.
11.36.48.ENDIF,L1.
11.36.48.IFE,$BLUE$.EQ.$BLUE$,L2.
11.36.48.  PROCESSING DONE FOR BLUE
11.36.48.REVERT.
11.36.48.BEGIN,A,COLORPR,RED.
11.36.49.IFE,$RED$.EQ.$RED$,L1.
11.36.49.  PROCESSING DONE FOR RED
11.36.49.REVERT.
11.36.49.BEGIN,A,COLORPR,PINK.
11.36.49.IFE,$PINK$.EQ.$RED$,L1.
11.36.49.ENDIF,L1.
11.36.49.IFE,$PINK$.EQ.$BLUE$,L2.
11.36.49.ENDIF,L2.
11.36.49.  NO PROCESSING DONE IF COLOR
11.36.49.  IS NOT RED OR BLUE
11.36.50.REVERT.CCL
```

SKIP STATEMENT

The SKIP statement initiates unconditional skipping of succeeding control statements. Skipping is terminated by an ENDIF statement that has a label string matching the label string specified on the SKIP statement. Only an ENDIF statement, not an ELSE statement, terminates skipping initiated by a SKIP statement.

The format of SKIP is:

SKIP,ls.

ls Label string; 1 to 10 alphanumeric characters beginning with an alphabetic character.

Example:

The SKIP,HALT. statement initiates skipping, and the statements following SKIP are ignored until ENDIF,HALT. is encountered.

```
SKIP,HALT.
COMMENT.  THIS IS SKIPPED.
ELSE,HALT.
COMMENT.  ELSE DOES NOT TERMINATE A SKIP.
ENDIF,STOP.
COMMENT.  ENDIF WITH A MATCHING LABEL STRING IS NEEDED.
ENDIF,HALT.
COMMENT.  SKIPPING STOPPED & EXECUTION CONTINUES HERE.
```

ELSE STATEMENT

The ELSE statement is used only with a matching IFE statement. It performs one of the following functions.

- A false IFE statement initiates skipping of control statements and the ELSE statement with a matching label string terminates that skipping. Execution continues with the next control statement.
- A true IFE statement executes statements until the ELSE statement with a matching label string is encountered. The ELSE statement initiates skipping to the ENDIF statement with a matching label string.

If the IFE and ELSE label strings do not match, the ELSE statement is ignored.

Neither a SKIP nor an ELSE statement terminates skipping initiated by another SKIP or ELSE statement.

The format of ELSE is:

ELSE,ls.

ls Label string; 1 to 10 alphanumeric characters beginning with an alphabetic character.

Example:

In the following procedure different processing occurs for different sexes. The IFE statement determines the processing that occurs for the value of the parameter SEX that BEGIN sent to the procedure.

```
.PROC,SP1,SEX.
IFE,$SEX$.EQ.$FEMALE$,LABEL1.
COMMENT.  DATA COLLECTED HERE
COMMENT.  ON WOMEN EMPLOYEES.
ELSE,LABEL1.
COMMENT.  DATA COLLECTED HERE
COMMENT.  ON MEN EMPLOYEES.
ENDIF,LABEL1.
```

The following dayfile segment results when BEGIN passes the value FEMALE to the procedure.

```
15.18.23.BEGIN,SP1,,SEX=$FEMALE$.
15.18.24.IFE,$FEMALE$.EQ.$FEMALE$,LABEL1.
15.18.24.  DATA COLLECTED HERE
15.18.24.  ON WOMEN EMPLOYEES.
15.18.24.ELSE,LABEL1.
15.18.24.ENDIF,LABEL1.
15.18.24.REVERT.CCL
```

The following dayfile segment results when BEGIN passes the value MALE to the procedure.

```
15.18.21.BEGIN,SP1,,SEX=$MALE$.
15.18.22.IFE,$MALE$.EQ.$FEMALE$,LABEL1.
15.18.22.ELSE,LABEL1.
15.18.22.  DATA COLLECTED HERE
15.18.22.  ON MEN EMPLOYEES.
15.18.22.ENDIF,LABEL1.
15.18.22.REVERT.CCL
```

ENDIF STATEMENT

The ENDIF statement terminates skipping initiated by a SKIP, IFE, or ELSE statement. In all cases, the label string on the ENDIF statement must match the label string on the statement that initiates the skipping. If CCL encounters an ENDIF statement with a nonmatching label string, it ignores that statement.

The format of ENDIF is:

ENDIF,ls.

ls Label string; 1 to 10 alphanumeric characters beginning with an alphabetic character.

Examples of the use of ENDIF are given with the descriptions of the IFE, SKIP, and ELSE statements.

ITERATIVE STATEMENTS

The CCL iterative statements WHILE and ENDW bracket a group of control statements into a loop that can be repeatedly processed. The WHILE statement identifies the beginning of the loop and the ENDW statement identifies its end. The ENDW statement must have a label string that matches the label string specified on the WHILE statement. The loop is repeated as long as the expression in the WHILE statement is true. If the expression is initially false, control immediately skips to the ENDW statement; if no matching ENDW statement is found, all the remaining statements in the control statement record or procedure are skipped.

Label strings of all WHILE statements within the control statement record of a job and within each procedure should be unique. Duplication of a label string within a control statement record or within a procedure can produce unpredictable results. The same label string can be used in a called procedure and in the calling control statement record or procedure.

WHILE STATEMENT

The format of WHILE is:

WHILE,exp,ls.

exp A CCL expression. The separator following exp must be a comma.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

ENDW STATEMENT

The format of ENDW is:

ENDW,ls.

ls Label string; 1 to 10 alphanumeric characters, beginning with an alphabetic character.

Example:

In the following job segment the value of control register 1 (R1) is set to 1 and control register 2 (R2) is set to 5. The FTN5 compiler continues to take input from file FROG and executes the FORTRAN 5 programs as long as the value of R1 is less than R2 (four times). Each pass through the loop increases the value of R1 by 1.

```
SET,R1=1.
SET,R2=5.
WHILE,R1<R2,LEAP.
FTN5,I=FROG.
LGO.
RETURN,LGO.
SET,R1=R1+1.
ENDW,LEAP.
```

The user can vary the number of repetitions by setting different values in R1 and R2.

ADDITIONAL CCL STATEMENTS

The following control statements display or set symbolic name values. The DISPLAY statement can also evaluate and display an expression that does not contain a symbolic name.

DISPLAY STATEMENT

The DISPLAY statement evaluates an expression and sends the result to the user dayfile in both decimal and octal integer forms. The largest decimal value that can 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. For an expression larger than $2^{48}-1$, zeros are displayed.

The DISPLAY control statement can also evaluate an expression as true or false, and send a 1 for true and a 0 for false to the user dayfile in both decimal and octal format. Under INTERCOM, the result is also displayed at the terminal.

The format of DISPLAY is:

DISPLAY(exp)

exp A CCL expression. Character strings within the expression must be constants, symbolic names, or CCL functions.

Example:

The following sample dayfile shows several display operations.

```
11.36.55.DISPLAY(TIME)
11.36.55.      1136     2160B
11.36.55.DISPLAY($ABC$)
11.36.55.      4227     10203B
11.36.55.SET,R1=99.
11.36.55.SET,R2=901.
11.36.55.DISPLAY(R1)
11.36.55.      99     143B
11.36.55.DISPLAY(R1+R2)
11.36.55.      1000     1750B
11.36.55.DISPLAY(R1.GT.R2)
11.36.55.      0     0B
11.36.55.DISPLAY(3/2)
11.36.56.      1     1B
11.36.56.DISPLAY(2**47)
11.36.56. GT   9999999999     4000000000000000B
11.36.56.DISPLAY(-2**47)
11.36.56. LT -9999999999    -4000000000000000B
11.36.56.DISPLAY(2**48)
11.36.56.      0     0B
11.36.56.DISPLAY(99999999999)
11.36.56.  CCL156- STRING TOO LONG -  99999999999
```

The first DISPLAY statement displays the value of the TIME symbolic name. The current time given is in the form hhmm. The next DISPLAY statement displays the display code value of the $-delimited characters. The next eight lines demonstrate the use of the R1 and R2 symbolic names. The other DISPLAY statements specify numeric expressions. The numeric constant in the final DISPLAY statement has more than 10 digits, resulting in an error message.

SET STATEMENT

The SET statement allows the user to set the value of a control register, an error flag, or the dayfile skipped control statement flag that determines whether or not skipped control statements are entered in the dayfile.

The format of SET is:

SET(sym=exp)

sym One of the following symbolic names (initially these names are set to 0).

Name	Description
R1, R2, or R3	Local control registers. When a procedure is called, the current values of R1, R2, and R3 do not change. The values of these registers may change within the procedure. However, when processing reverts, these registers are restored to the values they had when the procedure was called.

Name	Description
R1G	Global control register. When a procedure is called or reverts, R1G keeps its current value.
EF	Local error flag. When a procedure is called, the current value of the error flag does not change. The value of the error flag may change within the procedure. However, when processing reverts, the error flag is restored to the value it had when the procedure was called.
EFG	Global error flag. When a procedure is called or reverts, EFG keeps its current value.
DSC	Dayfile-skipped-control-statement flag. Initially, it is set to 0, so that control statements that are skipped (not processed) are not entered in the dayfile. If DSC is not zero, skipped statements are entered in the dayfile.

exp A CCL expression. The value derived through evaluation of the expression is assigned to the symbolic name. Acceptable values for each symbolic name follow. If the value is outside the specified range, CCL does not issue a message and selects a value within the range.

sym	Suggested Value
R1, R2, R3, or R1G	Any integer between -131 071 and 131 071 (-377777_8 and 377777_8).
EF or EFG	Any integer between 0 and 63. To assign the value defined by the system for an error condition, the user should set the error flag to one of the error condition symbolic names (refer to Symbolic Names, earlier in this section). CCL sets the EF flag to the appropriate error code when an error occurs. If EFG is 0 when a REVERT statement is processed in a procedure, CCL sets EFG to the value of EF (refer to REVERT Statement, later in this section).
DSC	1 or 0. If the value of the expression is nonzero, DSC is set to 1. When DSC is 1, skipped control statements are entered in the dayfile preceded by two periods. Some CCL error processing routines set DSC to 1 so that skipped control statements are written in the dayfile. When DSC is 0, skipped control statements are not entered in the dayfile.

Example 1 - DSC Flag Use:

The following control statements (on the left side) demonstrate the effect of DSC=0 and DSC=1. On the right is the dayfile segment resulting from processing of the control statements.

```
SET,DSC=0.
SKIP,LABL1.
COMMENT.  SINCE THE DAYFILE SKIP
COMMENT.  CONTROL IS SET TO ZERO,
COMMENT.  THESE STATEMENTS WILL NOT
COMMENT.  APPEAR IN THE DAYFILE.
ENDIF,LABL1.
SET,DSC=1.
SKIP,LABL2.
COMMENT.  SINCE THE DAYFILE SKIP
COMMENT.  CONTROL IS NOW SET TO ONE,
COMMENT.  THESE STATEMENTS WILL
COMMENT.  APPEAR IN THE DAYFILE
COMMENT.  PREFIXED WITH
COMMENT.  TWO PERIODS.
ENDIF,LABL2.
```

```
11.36.58.SET,DSC=0.
11.36.58.SKIP,LABL1.
11.36.58.ENDIF,LABL1.
11.36.58.SET,DSC=1.
11.36.58.SKIP,LABL2.
11.36.58...COMMENT.  SINCE THE DAYFILE SKIP
11.36.58...COMMENT.  CONTROL IS NOW SET TO
11.36.58...COMMENT.  ONE, THESE STATEMENTS
11.36.58...COMMENT.  WILL APPEAR IN THE
11.36.58...COMMENT.  DAYFILE PREFIXED WITH
11.36.58...COMMENT.  TWO PERIODS.
11.36.58.ENDIF,LABL2.
```

Example 2 - Error Flag (EF) Use:

The following job segment determines if the error that occurred in the job control record was a CPU abort error (CPE). The IFE statement compares the value of EF to the value of CPE. If they are equal, the value of R1 is set to 1 and the ENDIF statement is printed in the dayfile. If the values of EF and CPE are not equal, CCL skips to the ENDIF statement. In either case, the value of R1 and EF are displayed.

```
EXIT,S.
SET,R1=0.
IFE,EF=CPE,SKIP.
SET,R1=1.
ENDIF,SKIP.
DISPLAY,R1.
DISPLAY,EF.
```

Following are the two possible dayfiles.

```
11.42.55.EXIT,S.
11.42.55.SET,R1=0.
11.42.55.IFE,EF=CPE,SKIP.
11.42.55.ENDIF,SKIP.
11.42.55.DISPLAY,R1.
11.42.56.      0     0B
11.42.56.DISPLAY,EF.
11.42.56.      3     3B
```

```
10.39.33.EXIT,S.
10.39.33.SET,R1=0.
10.39.33.IFE,EF=CPE,SKIP.
10.39.33.SET,R1=1.
10.39.33.ENDIF,SKIP.
10.39.33.DISPLAY,R1.
10.39.33.      1     1B
10.39.33.DISPLAY,EF.
10.39.33.      4     4B
```

Example 3 - Control Register Use:

Procedure P1 is on procedure file SETFILE.

```
.PROC,P1.
DISPLAY(R1)
DISPLAY(R1G)
SET(R1=9)
SET(R1G=888)
```

The following control statements (on the left side) set and display registers R1 and R1G. BEGIN calls procedure P1, which displays these registers and resets them, then processing reverts to the control statement record where the registers are again displayed.

The R1 and R1G registers retain their setting when the procedure is called. However, after new values are set in the procedure and control reverts to the control statement record, R1 returns to its previous value and R1G retains the value set in the procedure.

On the right is the dayfle segment resulting from processing of the control statements.

```
SET,R1=1.
SET,R1G=10.
DISPLAY,R1.
DISPLAY,R1G.
ATTACH,SETFILE,ID=SPARK.
BEGIN,P1,SETFILE.
DISPLAY,R1.
DISPLAY,R1G.
```

```
11.36.56.SET,R1=1.
11.36.56.SET,R1G=10.
11.36.56.DISPLAY,R1.
11.36.56.      1     1B
11.36.56.DISPLAY,R1G.
11.36.56.      10    12B
11.36.56.ATTACH,SETFILE,ID=SPARK.
11.36.56.PFN IS
11.36.56.SETFILE
11.36.57.AT CY= 005 SN=SPFSET
11.36.57.BEGIN,P1,SETFILE.
11.36.57.DISPLAY(R1)
11.36.57.      1     1B
11.36.57.DISPLAY(R1G)
11.36.57.      10    12B
11.36.57.SET(R1=9)
11.36.57.SET(R1G=888)
11.36.57.REVERT.CCL
11.36.57.DISPLAY,R1.
11.36.57.      1     1B
11.36.57.DISPLAY,R1G.
11.36.58.      888   1570B
```

Example 4 - Error Flag Use (EFG Zero):

To return the error code generated in a procedure to the control statement record, error processing must occur within the procedure and EFG must be 0 before the procedure reverts. The following procedure sets the global error flag to 0 and attempts to attach a file that does not exist.

```
.PROC,PP.
SET,EFG=0.
ATTACH,BOXES,ID=RIBBON.
FTN5(I=BOXES,L=0)
EXIT,S.
DISPLAY(EF)
DISPLAY(EFG)
REVERT.
```

The dayfile segment (on the right) resulting from processing of the job control statements (on the left) shows how the error code is returned.

```
ATTACH,SETFILE,ID=RIBBON.
BEGIN,PP,SETFILE.
DISPLAY(EF)
DISPLAY(EFG)
```

```
15.18.24.ATTACH,EFGFILE,ID=RIBBON.
15.18.24.PFN IS
15.18.24.EFGFILE
15.18.25.AT CY= 001 SN=SPFSET
15.18.25.BEGIN,PP,EFGFILE.
15.18.25.SET,EFG=0.
15.18.25.ATTACH,BOXES,ID=RIBBON.
15.18.25.PFN IS
15.18.25.BOXES
15.18.25.FILE NOT CATALOGED,SN=SPFSET
15.18.25.PF ABORT
15.18.26.EXIT,S.
15.18.26.DISPLAY(EF)
15.18.26.      3     3B
15.18.26.DISPLAY(EFG)
15.18.26.      0     0B
15.18.26.REVERT.
15.18.26.DISPLAY(EF)
15.18.26.      0     0B
15.18.26.DISPLAY(EFG)
15.18.26.      3     3B
```

PROCEDURES

A procedure is a sequence of control statements that can be executed from either the control statement record or another procedure. A procedure is to the job control statement record as a subroutine is to a program in that both

- Contain statements that usually perform a single function within the job.
- Can be repeatedly executed.
- Are executed by the main job or by another procedure/subroutine.

There are two types of procedures; interactive and noninteractive. All following references to procedures in this section apply to both types, unless prefixed by the word interactive or noninteractive.

A procedure consists of a procedure header statement and a procedure body. The procedure header statement must be the first line in the procedure. It names the procedure and identifies the parameters in the procedure body that can be changed during execution.

The procedure body contains all statements between the header statement and the end-of-record or end-of-partition. The control statements within the procedure body usually perform a function that the user wants to execute often, such as routing a file to a line printer (the procedure would request a file on a queue device, copy the file to be printed to the queue device file, and route it to the printer). A procedure body must contain at least one control statement. All control statements, including CCL statements, are legal within a procedure. The body can also include special procedure commands (explained later in this section).

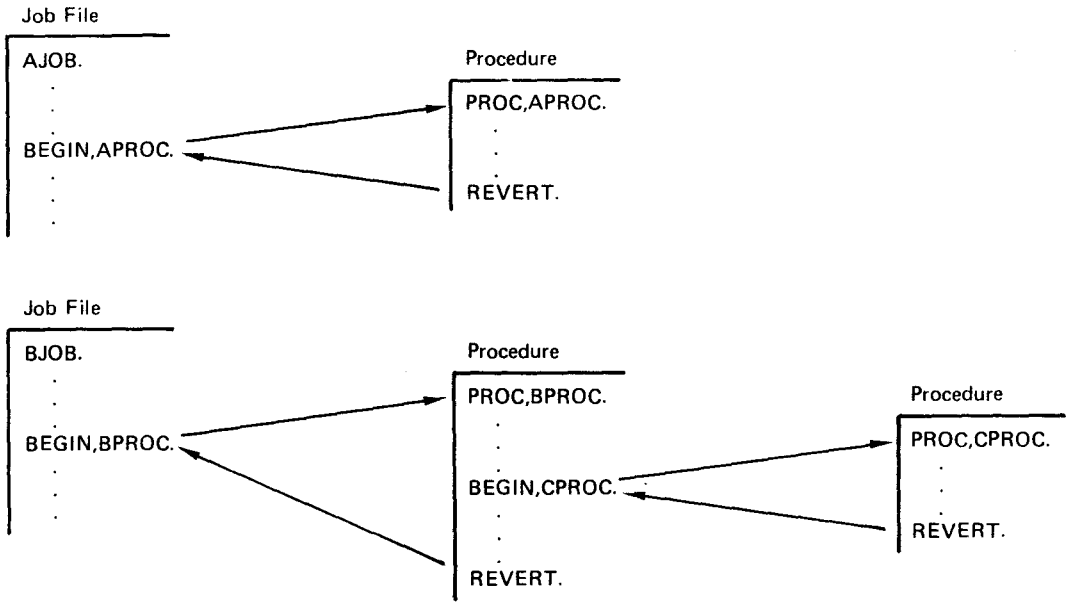
A procedure is stored as a record on a file. The user can put any number of procedures on a single file; each must be a separate record. The procedure file may be a local file or an attached permanent file. The procedure file can reside on magnetic tape as well as on mass storage and can optionally be in a user library (refer to EDITLIB statement in section 4 and to the CYBER Loader Reference Manual, listed in the preface) or a system library.

A BEGIN or name call statement initiates execution of a procedure. When a procedure is called, the BEGIN or name call parameters pass values to the procedure's parameters. If data records are defined within the procedure, CCL writes them to a separate local file. A REVERT statement within the procedure returns job control to the statement following the BEGIN or name call statement.

PROCEDURE CALL AND RETURN

The BEGIN or name call statement initiates processing of a procedure. A procedure can include BEGIN and name call statements to call other procedures. After the final control statement in the procedure is processed, a user- or CCL-supplied† REVERT statement continues processing with the control statement following the BEGIN or name call control statement. Use of the BEGIN and REVERT statements is illustrated in figure 5-1.

†CCL issues a REVERT statement if the user does not supply one within a procedure.



In AJOB, BEGIN initiates execution of procedure APROC and the REVERT statement returns job processing to the statement following BEGIN.

In BJOB, BEGIN initiates execution of procedure BPROC. Within BPROC a BEGIN statement initiates execution of procedure CPROC. The REVERT statemet within CPROC returns job processing to the statement following BEGIN,CPROC. When BPROC is done executing, REVERT returns processing to the statement following the BEGIN statement in the job control record.

Figure 5-1. BEGIN Statement Calling a Procedure

BEGIN Statement and Name Call Statement

The BEGIN control statement and name call statement call and initiate processing of a procedure. With optional parameters they can make substitutions for the keywords in the procedure body.

The format of BEGIN is:

BEGIN,pname,pfile,p_1,p_2,...,p_n.

The format of name call is:

pname,p_1,p_2,...,p_n.

pname — Procedure name from the procedure header.

In the BEGIN format, pname is the name of a procedure on file pfile. If pname is omitted, two consecutive commas must be specified. The default procedure is the record at the current position of file pfile. If pfile is at end-of-information, CCL rewinds pfile and uses the first record. If pfile is INPUT, the file is not rewound.

In the name call format pname is the name of the local file containing a procedure, the name of a procedure on NUCLEUS, the name of a procedure on a library (refer to EDITLIB control statement in section 4) in the global library set (refer to LIBRARY statement in the CYBER Loader Reference Manual), or the name of a procedure on the default system library. The procedure name may or may not be the same as the local file name. pname must be specified in the name call format.

pfile — Name of the file containing the procedure.

In the BEGIN format pfile must be the second parameter. Its omission is indicated by two consecutive commas. If pfile is omitted, the installation-defined default file name is used. The released default is PROCFIL.

When the BEGIN statement is processed, CCL looks for local file pfile. If no local pfile is found, CCL attempts to attach permanent file PROCFIL with the user id PUBLIC. Once pfile or PROCFIL is located, CCL searches for procedure pname.

p_i — Optional parameter specifying the substitution to be made for a keyword used in the procedure. If the user needs only the default values specified on the procedure header, omit the p_i parameters. If a required parameter is omitted on a call to an interactive procedure, the system prompts the user for a parameter value. (For a complete description of parameter use in procedures, refer to Parameter Substitution and Procedure Header Statement, or Interactive Procedures, later in this section.)

The following parameter formats are available.

keyword — The parameter is identical to a keyword on the procedure header. In noninteractive procedures, keyword, or the second default for the procedure's keyword, is used. In interactive procedures, the interactive procedure header determines what replaces the keyword in the procedure body.

keyword=

References to the keyword in the procedure are removed (null substitution).

value

CCL assigns this 1- to 40-character symbolic name or value to the keyword whose position in the procedure header parameter list matches the position of this parameter in the BEGIN statement parameter list. In interactive procedures, value must conform to parameter checklist specifications. A value containing nonalphanumeric characters, other than / or -, must be $-delimited.

keyword=value

The symbolic name or value is substituted for the keyword wherever it appears in the procedure. If value is followed by a +, value must be a symbolic name. (Refer to Symbolic Names earlier in this section.) This keyword in the BEGIN statement is the same keyword that is used in the procedure header statement. In interactive procedures, value must conform to parameter checklist specifications.

The following formats can be used.

Format	Description
keyword=value or keyword=symbol	Substitutes the value or symbolic name itself for the keyword in the procedure.
keyword=symbol+ or keyword=symbol+D	Substitutes the decimal value associated with the symbolic name for the keyword in the procedure.
keyword=symbol+B	Substitutes the octal value associated with the symbolic name for the keyword in the procedure.

When calling a procedure, a keyword can be named more than once if the keyword=value parameter format is used each time. CCL issues a message informing the user that a keyword is named more than once on the statement. It uses the value specified with the last occurrence of the keyword.

Example:

The following procedure is on file FKTEST and is accessed by a sequence of calling statements in the control statement record of the job.

```
.PROC,TEST1,FK.
COMMENT.        FK.
```

The resulting dayfile shows each calling statement and the substitutions made. The relevant segment of the dayfile is as follows:

```
11.53.43.BEGIN,TEST1,FKTEST,20.
11.53.43.      20.
11.53.43.REVERT.CCL
11.53.44.SET,R2=100.
11.53.44.BEGIN,TEST1,FKTEST,FK=R2+.
11.53.44.      100.
11.53.44.REVERT.CCL
11.53.44.BEGIN,TEST1,FKTEST,FK=R2+D.
11.53.44.      100.
11.53.44.REVERT.CCL
11.53.45.BEGIN,TEST1,FKTEST,FK=R2+B.
11.53.45.      144.
11.53.45.REVERT.CCL
11.53.45.BEGIN,TEST1,FKTEST.
11.53.45.      FK.
11.53.45.REVERT.CCL
11.53.45.BEGIN,TEST1,FKTEST,FK=.
11.53.46.      .
11.53.46.REVERT.CCL
11.53.46.BEGIN,TEST1,FKTEST,VALUE.
11.53.46.      VALUE.
11.53.46.REVERT.CCL
11.53.46.BEGIN,TEST1,FKTEST,VALUE-2.
11.53.46.  CCL212- SEPARATOR INVALID     VALUE-
11.53.46.EXIT.
11.53.46.BEGIN,TEST1,FKTEST,$VALUE-2$.
11.53.47.      VALUE-2.
11.53.47.REVERT.CCL
```

REVERT Statement

The REVERT statement terminates processing in the procedure and returns control to the statement following the BEGIN statement that called the procedure. The REVERT,ABORT statement sets the error flag EF=CPE (CPU abort) and processing continues with the next EXIT,S statement in the control statement record (refer to Exit Processing in section 4).

The formats of REVERT are:

REVERT.

or

REVERT,ABORT.

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 can place REVERT at the end of a procedure, but this is unnecessary because CCL provides an implicit REVERT sequence. CCL always appends the following three control statements to a procedure. The CCL following each statement identifies it as generated by CCL.

```
REVERT.CCL
EXIT,S.CCL
REVERT,ABORT.CCL
```

If the procedure did not produce a fatal error, CCL processes the REVERT. statement. If the procedure did produce a fatal error, CCL skips the first statement in this sequence. CCL executes the EXIT,S. statement to terminate skipping, and processes the REVERT,ABORT. statement.

The user may wish to use an 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.

NOTE

EXIT should be used with caution because it can terminate the job.

The user and/or system can set the value of the symbolic names R1, R2, R3, EF, EFG, and DCS. During a REVERT, CCL can change their values (refer to SET Statement, earlier in this section).

Example 1:

If a fatal error occurs during the processing of the LGO statement in the following procedure segment, the system skips to the EXIT statement, dumps CM, and processes the REVERT,ABORT. If no fatal error occurs during the processing of the LGO, CCL processes the REVERT statement.

```
LGO.
REVERT.
EXIT,S.
DMP.
REVERT,ABORT.
```

Example 2:

The following procedure (PFCLEAN) is on file PROCFIL. It purges all but the highest cycle of a permanent file on any device set. If the set name parameter (SN) is specified in the BEGIN statement, all but the highest cycle of the file are purged from the specified device set. If SN is omitted, the lower cycle files are purged from the default permanent file set.

```
.PROC,PFCLEAN,PFN,ID=LYNN,PW=MINE,SN=0.
IFE,NUM(SN)=0,OK.
ATTACH,PRF1,PFN,#ID=ID,MR=1,#SN=SN.
PURGE,PRF2,PFN,#ID=ID,#PW=PW,LC=1,#SN=SN.
PURGE,PRF3,PFN,#ID=ID,#PW=PW,LC=1,#SN=SN.
PURGE,PRF4,PFN,#ID=ID,#PW=PW,LC=1,#SN=SN.
PURGE,PRF5,PFN,#ID=ID,#PW=PW,LC=1,#SN=SN.
EXIT,U.
RETURN,PRF1,PRF2,PRF3,PRF4,PRF5.
REVERT.
ENDIF,OK.
ATTACH,PRF1,PFN,#ID=ID,MR=1.
PURGE,PRF2,PFN,#ID=ID,#PW=PW,LC=1.
PURGE,PRF3,PFN,#ID=ID,#PW=PW,LC=1.
PURGE,PRF4,PFN,#ID=ID,#PW=PW,LC=1.
PURGE,PRF5,PFN,#ID=ID,#PW=PW,LC=1.
EXIT,U.
RETURN,PRF1,PRF2,PRF3,PRF4,PRF5.
```

The following statements attach PROCFIL and call the procedure. The SN parameter equals zero on the BEGIN statement so SN is a numeric, the IFE statement is false, and procedure PFCLEAN purges all but the highest cycles of file TEMP from the default permanent file set.

```
ATTACH,PROCFIL,ID=LYNN.
BEGIN,PFCLEAN,,TEMP,ID=LYNN,PW=,SN=0.
```

Following is the resulting dayfile segment.

```
09.48.28.ATTACH,PROCFIL,ID=LYNN.
09.48.29.PFN IS
09.48.29.PROCFIL
09.48.29.AT CY= 007 SN=SPFSET
09.48.29.BEGIN,PFCLEAN,,TEMP,ID=LYNN,PW,SN=0.
09.48.29.IFE,NUM(0)=0,OK.
09.48.29.ENDIF,OK.
09.48.29.ATTACH,PRF1,TEMP,ID=LYNN,MR=1.
09.48.30.AT CY= 004 SN=SPFSET
09.48.30.PURGE,PRF2,TEMP,ID=LYNN,PW=*---*,LC=1.
09.48.30.PR ID=   LYNN PFN=TEMP
09.48.30.PR CY= 001 SN=SPFSET  00000064 WORDS.
09.48.30.PURGE,PRF3,TEMP,ID=LYNN,PW=*---*,LC=1.
09.48.31.PR ID=   LYNN PFN=TEMP
09.48.31.PR CY= 002 SN=SPFSET  00000064 WORDS.
09.48.31.PURGE,PRF4,TEMP,ID=LYNN,PW=*---*,LC=1.
09.48.31.PR ID=   LYNN PFN=TEMP
09.48.31.PR CY= 003 SN=SPFSET  00000064 WORDS.
09.48.31.PURGE,PRF5,TEMP,ID=LYNN,PW=*---*,LC=1.
09.48.31.FILE ALREADY ATTACHED
09.48.31.INCORRECT PERMISSION
09.48.31.PF ABORT
09.48.32.EXIT,U.
09.48.32.RETURN,PRF1,PRF2,PRF3,PRF4,PRF5.
09.48.32.REVERT.CCL
```

NONINTERACTIVE PROCEDURE HEADER STATEMENT

The noninteractive procedure header statement is the first line of the procedure. It identifies the procedure and specifies the keywords to which the BEGIN statement can pass values. The BEGIN statement substitutes the keywords into the procedure body. Unless the header statement contains an error, it does not appear in the dayfile.

Syntax rules for header statements are as follows.

- The header statement must begin with a period followed by the characters PROC.
- The separators between parameters must be commas.
- A period terminates the header statement.
- The header statement can extend over more than one line if each line to be continued ends with a comma.

The format of the noninteractive procedure header statement is:

.PROC,pname,$p_1,p_2,\ldots,p_n$.

pname Name of the procedure; one to seven alphanumeric characters. It can begin with or consist entirely of numeric characters, unless it is to be a name call statement. Then it must begin with an alphabetic character. pname cannot be BEGIN.

p_i Optional parameters whose keywords are used in the body of the procedure. Depending on the BEGIN statement parameters, keywords in the procedure body can be removed, left as they are, replaced by a value specified in the BEGIN statement, or replaced by first or second default values as specified in the procedure header parameter. (Refer to Parameter Substitution in Noninteractive Procedures, later in this section.)

The maximum number of procedure header keywords is defined by the installation. The released maximum default is 50.

The following are the legal formats for p_i.

Format	Example
keyword	FILE1
keyword=	FILE1=
keyword=default1	FILE1=LGO
keyword=default1/default2	FILE1=LGO/OLD
keyword=/default2	FILE1=/OLD
keyword=#DATA (Control Data graphics: keyword= DATA)	FILE1=#DATA
keyword=#FILE (Control Data graphics: keyword= FILE)	FILE1=#FILE

keyword A 1- to 10-character keyword.

default1 A 1- to 40-character first default value. If default1 contains nonalphanumeric characters, it must be $-delimited. This default value replaces the keyword in the procedure body if this parameter is omitted from the BEGIN statement.

default2 A 1- to 40-character second default value. If default2 contains nonalphanumeric characters, it must be $-delimited. This default value replaces the keyword in the procedure body if the BEGIN statement specifies a parameter identical to the noninteractive procedure's keyword.

default1 and default2 could be either of the special values #DATA and #FILE. The #DATA and #FILE options allow the procedure to access records within the procedure file. #DATA and #FILE are often used to access program data.

The #DATA and #FILE options can be overridden by BEGIN statement parameters.

#DATA	If this default value is used, keyword within the procedure body references the record(s) created by the .DATA command (refer to .DATA Command, later in this section). Keyword substitution occurs within the record(s) created by .DATA, since the procedure's keyword substitution occurs before the .DATA command creates the record(s).
#FILE	If this default value is used, keyword within the procedure body references the next record on the procedure file (refer to figure 5-4 and .DATA Command, later in this section). Keyword substitution does not occur within the referenced record, since it is not within the procedure.

PROCEDURE BODY

The procedure body consists of all statements between the procedure header statement and the end-of-record. These statements can be control statements, which include CCL statements, and CCL procedure commands (refer to Procedure Commands, later in this section).

The BEGIN (or name call) control statement initiates execution of the procedure and passes any specified parameters to the procedure. Within the procedure body the BEGIN parameters can change the control statement's parameters. When BEGIN calls a procedure, substitutions are made for the parameters in the procedure, and the procedure body becomes the control statement record until a REVERT statement is encountered.

PARAMETER SUBSTITUTION IN NONINTERACTIVE PROCEDURES

The .PROC control statement specifies keywords that are used in the procedure body. A user can change the value of these keywords each time the procedure is executed by using the appropriate parameters in the BEGIN statement. With the BEGIN parameter the user can remove a keyword (null substitution), leave it as is, or replace it with a .PROC default or another value.

After substitutions for the keywords are made in the procedure body, some control statements may be expanded beyond 80 characters. For most control statements, this is flagged as an error. Some exceptions are CCL statements and the LABEL and VSN statements, which can extend over more than one line if the statement is split at a separator. The user should ensure that the line containing the parameter is short enough so that possible expansion does not extend the line beyond the 80th character.

When a procedure is called, CCL must match each parameter on the call statement with a parameter on the noninteractive-procedure header statement. Order-dependent and order-independent are the two methods of parameter matching used by CCL.

Order-Dependent Parameter Matching Mode

Parameter matching always begins in order-dependent mode (refer to Order-Independent Parameter Matching Mode, later in this section, for information on changing parameter matching modes). CCL compares, in order, each parameter on the BEGIN statement with the parameter in that position on the noninteractive-procedure header statement. CCL then substitutes the selected parameters into the procedure body.

All possible parameter substitutions in order-dependent mode are summarized in table 5-1. The table shows each parameter format on the BEGIN statement, each parameter format on the noninteractive-procedure header statement, and the substitution resulting from each combination. The word value in the table indicates that the parameter in the BEGIN statement (called value) is different from the correspond- ing keyword and/or defaults on the header statement. Keyword in the BEGIN statement is identical to the keyword in the noninteractive procedure header statement parameter. For example, if the procedure keyword is SIZE, SMALL is a BEGIN value and SIZE is a BEGIN keyword.

TABLE 5-1. PARAMETER SUBSTITUTION IN ORDER-DEPENDENT MODE

	BEGIN Statement Parameter Format		
Noninteractive-Procedure Header Parameter Format	omitted	keyword or $keyword$	value or $value$
keyword	keyword	keyword	value
keyword=	null	keyword	value
keyword=default1	default1	keyword	value
keyword=default1/default2†	default1	default2	error

†Switches keyword substitution to order-independent mode for all subsequent parameters.

Assuming that all parameter matches between the BEGIN statement and the noninteractive-procedure header are valid for order-dependent mode (table 5-1), CCL completes parameter matching in order-dependent mode.

In order-dependent mode, CCL ignores excess parameters on the BEGIN statement.

The user should use table 5-1 with the following examples to clarify the table entry meanings (keyword, default1, default2, value, and null).

Examples - Parameter Matching In Order-Dependent Mode:

Noninteractive-Procedure on Attached File, MYFILE

```
.PROC,SAMPL1,L,M,N=XY.
REWIND,L,A,M,N.
```

Call, Substitution, and Explanation

```
BEGIN,SAMPL1,MYFILE.
```

yields

```
REWIND,L,A,M,XY.
```

When parameters are omitted on the BEGIN statement, the system uses the defaults from the noninteractive-procedure header (L, M, and XY).

Noninteractive-Procedure on Attached File, MYFILE	Call, Substitution, and Explanation

	`BEGIN,SAMPL1,MYFILE,,,N.`
	yields
	`REWIND,L,A,M,N.`
	Omitted parameters indicate use of the procedure header defaults (L and M). N overrides the procedure header default (XY).

`.PROC,SAMPL2,LFN1=,LFN2,SBF=/SBF.`
`COPY_SBF(LFN1,LFN2)`
(Refer to Parameter Alteration, later in this section,for a description of_).
	`BEGIN,SAMPL2,MYFILE.`
	yields
	`COPY(,LFN2)`
	Omitted parameters indicate use of procedure header defaults (LFN2 and null substitution for LFN1 and SBF).

	`BEGIN,SAMPL2,MYFILE,,,SBF.`
	yields
	`COPYSBF(,LFN2)`
	Omitted parameters indicate use of procedure header defaults (null and LFN2). The BEGIN statement parameter, SBF, indicates use of the second default of the SBF procedure header parameter (SBF). The linking character (_) connects COPY and SBF to make COPYSBF.

	`BEGIN,SAMPL2,MYFILE,FORMS,TANK.`
	yields
	`COPY(FORMS,TANK)`
	FORMS replaces LFN1= and TANK replaces LFN2. Since the third parameter is omitted, the system uses the procedure header default (null).

`.PROC,SAMPL3,PFN,CYCLE=$.$/$,CY=25.$.`
`ATTACH,PFN,ID=MARTY_CYCLE`
	`BEGIN,SAMPL3,MYFILE,TAXES,CYCLE.`
	yields
	`ATTACH,TAXES,ID=MARTY,CY=25.`
	`PFN IS`
	`TAXES`
	`AT CY= 025 SN=SPFSET`
	TAXES replaces PFN and the BEGIN statement parameter CYCLE indicates use of the second default of the CYCLE procedure header parameter (,CY=25.).

Example 1:

The following procedure is on file PROCFIL. It prepares a file for processing. If the file is local, it is rewound. If it is not local, the system searches for the file in the user's permanent file catalog. If the file is not found, the procedure reverts and aborts.

```
.PROC,PREPARE,FNAME=,ID=SOUL.
IFE,FILE(FNAME,AS),PREP1.
REWIND(FNAME)
REVERT.  FNAME PREPARED.
ENDIF,PREP1.
ATTACH(FNAME,#ID=ID)
REVERT.  FNAME PREPARED.
EXIT,S.
REVERT,ABORT.  FNAME NOT FOUND.
```

On the left is the call to procedure PREPARE. Since PROCFIL is the default file, it does not have to be specified and is noted by successive commas (PROCFIL is already local to the job). On the right is the resulting dayfile.

```
BEGIN,PREPARE,,TEST.
```

```
10.48.11.BEGIN,PREPARE,,TEST.
10.48.11.IFE,FILE(TEST,AS),PREP1.
10.48.11.ENDIF,PREP1.
10.48.11.ATTACH(TEST,ID=SOUL)
10.48.11.PFN IS
10.48.11.TEST
10.48.11.AT CY= 001 SN=SPFSET
10.48.12.REVERT.  TEST PREPARED.
```

Example 2 - Parameter Matching In Nested Procedures:

As shown in figure 5-2, procedure EXECUTE resides on file PFILE1 and procedure LISTING resides on file PFILE2. Procedure EXECUTE executes a FORTRAN 5 program and if no errors occur, calls procedure LISTING to print the FORTRAN 5 output.

Job Record Segment

```
ATTACH(PRGRAM1,ID=LYNN)
BEGIN,EXECUTE,PFILE1,PRGRAM1,PRINT.
```

PFILE1

```
.PROC,EXECUTE,NAME,OUT.
FTN5(I=NAME,L=OUT)
LGO.
EXIT(U)
IFE,EF=0,DROP.
ATTACH(PFILE2,ID=LYNN)
BEGIN,LISTING,PFILE2,OUT.
ENDIF,DROP.
```

PFILE2

```
.PROC,LISTING,OUTFILE=FILE
REWIND(OUTFILE)
COPYSBF(OUTFILE,OUTPUT)
```

Resulting Dayfile

```
10.48.12.ATTACH(PRGRAM1,ID=LYNN)
10.48.12.PFN IS
10.48.12.PRGRAM1
10.48.12.AT CY= 001 SN=SPFSET
10.48.12.BEGIN,EXECUTE,PFILE1,PRGRAM1,PRINT.
10.48.13.FTN5(I=PRGRAM1,L=PRINT)
10.48.14.      60700 SCM STORAGE USED.
10.48.14.      0.038 CP SECONDS COMPILATION TIME.
10.48.14.LGO.
10.48.17.     STOP
10.48.17.      15700 MAXIMUM EXECUTION FL.
10.48.17.       .024 CP SECONDS EXECUTION TIME.
10.48.17.EXIT(U)
10.48.17.IFE,EF=0,DROP.
10.48.17.ATTACH(PFILE2,ID=LYNN)
10.48.17.PFN IS
10.48.17.PFILE2
10.48.17.AT CY= 001 SN=SPFSET
10.48.17.BEGIN,LISTING,PFILE2,PRINT.
10.48.17.REWIND(PRINT)
10.48.17.COPYSBF(PRINT,OUTPUT)
10.48.18.REVERT.CCL
10.48.18.ENDIF,DROP.
10.48.18.REVERT.CCL
```

BEGIN calls procedure EXECUTE. BEGIN's PRINT parameter replaces the OUT parameters in procedure EXECUTE. In procedure EXECUTE the OUT parameter in the BEGIN statement becomes PRINT. The call to procedure LISTING replaces OUTFILE's default FILE with PRINT, and all occurrences of OUTFILE become PRINT.

Figure 5-2. Keyword Substitution in Two Procedures

Order-Independent Parameter Matching Mode

CCL switches to order-independent mode to match the remainder of the parameters if, in comparison of a BEGIN statement parameter and a noninteractive-procedure header parameter, one of the following occurs.

- A BEGIN statement parameter is in the format keyword= or keyword=value.
- A noninteractive-procedure header statement parameter is in the format keyword=default1/default2.

For each BEGIN statement, parameter matching always begins in order-dependent mode. Once in order-independent mode, CCL matches each successive keyword of the BEGIN statement to the identical keyword in the procedure header statement, regardless of the order of the procedure header parameters.

The following statements illustrate the parameter combinations that result in switching from order-dependent mode to order-independent mode.

Procedure on Local Default File PROCFIL

```
.PROC,SALES,TAX,TOTAL=,FLAG=A.
COPYL(TAX,TOTAL,HOLD,,FLAG)
CATALOG(HOLD,TAX,ID=QUIET,RP=31)
```

Call, Substitution, and Explanation

```
BEGIN,SALES,,TAX,TOTAL=SUM,FLAG.
```

yields

```
COPYL(TAX,SUM,HOLD,,A)
 COPYL COMPLETE.
CATALOG(HOLD,TAX,ID=QUIET,RP=31
```

Parameter matching starts in order-dependent mode. The BEGIN parameter TOTAL=SUM switches the mode to order-independent mode. FLAG is then matched in order-independent mode, which yields A.

```
.PROC,TAXES,TAX=FED/MN,DEDUCT,FLAG=A.
COPYL(TAX,DEDUCT,RESV,,FLAG)
CATALOG(RESV,TAX,ID=QUIET,RP=16)
```

```
BEGIN,TAXES,,TAX,DEDUCT.
```

yields

```
COPYL(MN,DEDUCT,HOLD,,A)
 COPYL COMPLETE.
CATALOG(HOLD,MN,ID=QUIET,RP=16
```

The TAX=FED/MN procedure parameter switches the mode to order-independent mode. All parameters will be matched in order-independent mode.

All possible parameter substitutions in order-independent mode are summarized in table 5-2. The table shows each parameter format on the BEGIN statement, each parameter format on the noninteractive-procedure header statement, and the substitution resulting from each combination.

The word value in the table indicates that the parameter in the BEGIN statement (called value) is different than the keyword and/or defaults on the procedure header statement. The user should use table 5-2 with the following examples to clarify the table entry meanings (keyword, default1, default2, value, and null).

TABLE 5-2. PARAMETER SUBSTITUTION IN ORDER-INDEPENDENT MODE

Noninteractive-Procedure Header Parameter Format	BEGIN Statement Parameter Format				
	omitted	keyword or $keyword$	keyword= or $keyword$=	keyword=value or $keyword$=value	value or $value$ †
keyword	keyword	keyword	null	value	error
keyword=	null	null	null	value	error
keyword=default1	default1	default1	null	value	error
keyword=default1/ default2	default1	default2	null	value	error

† Assumes the parameter is entered under order-independent mode.

Examples of Parameter Matching:

Procedure on Local Default File PROCFIL

```
.PROC,SAMPL1,L,M,N=XY.
REWIND,L,A,M,N.
```

Call, Substitution, and Explanation

```
BEGIN,SAMPL1,,L=SWITCH.
```

yields

```
REWIND,SWITCH,A,M,XY.
```

The BEGIN parameter L=SWITCH switches parameter matching mode to order-independent mode. Order-independent mode uses the procedure header defaults for omitted BEGIN parameters. Order-dependent and order-independent modes work identically for omitted BEGIN parameters.

```
BEGIN,SAMPL1,,L=CHANGE,M,N.
```

or

```
BEGIN,SAMPL1,,L=CHANGE,N,M.
```

yields

```
REWIND,CHANGE,A,M,XY.
```

The L=CHANGE parameter switches parameter matching to order-independent mode. In order-independent mode the order of the BEGIN parameters does not matter. M matches with M, and the BEGIN keyword N indicates substitution of the procedure header default (XY).

```
BEGIN,SAMPL1,,L=FLIP,M=B,N=Z.
```

yields

```
REWIND,FLIP,A,B,Z.
```

BEGIN parameters in the form keyword=value always override procedure header parameters. FLIP replaces L, B replaces M, and Z replaces XY.

Noninteractive-Procedure on Local Default File PROCFIL	Call, Substitution, and Explanation
.PROC,TRACE,MS,MR,MA. COPYL(MS,MR,MA)	BEGIN,TRACE,,MS=,MR=MD,MA=. yields COPYL(,MD,) The MS= parameter switches parameter matching to order-independent mode. All BEGIN statements in the form keyword= use null substitution.
.PROC,SAMPL4,FILE1,EC=B6/A6,DC=LR,REP=0. REQUEST,AAA,Q. COPY(FILE1,AAA) ROUTE(AAA,#DC=DC,#EC=EC,#REP=REP)	BEGIN,SAMPL4,,COIN. yields REQUEST,AAA,Q. COPY(COIN,AAA) ROUTE(AAA,DC=LR,EC=B6,REP=0) COIN is substituted in order-dependent mode. EC=B6/A6 switches the mode to order-independent mode. The omitted parameters indicate use of the procedure header defaults (order-dependent and order-independent mode work alike for omitted BEGIN parameters). BEGIN,SAMPL4,,COIN,EC,DC,REP. yields REQUEST,AAA,Q. COPY(COIN,AAA) ROUTE(AAA,DC=LR,EC=A6,REP=0) EC=B6/A6 switches the mode to order-independent mode. Specifying a keyword on the BEGIN statement produces the same result as omitting it (refer to previous example) except for the double default procedure parameter, EC=B6/A6. If EC is omitted, B6 is used. If EC is specified, A6 is used.

Example 1:

The following procedure resides on file PROCFIL. It routes a specified file (FNAME) to the specified equipment (default is any Control Data graphics line printer).

```
.PROC,PRINTR,FNAME,REP=0,DC=LR,EC=B6.
REQUEST,AAA,Q.
COPY(FNAME,AAA)
ROUTE(AAA,#DC=DC,#REP=REP,#EC=EC)
REVERT.  FNAME ROUTED.
EXIT,S.
REVERT,ABORT.  PRINTR PARAMETER ERRORS.
```

The following control statements attach file PROCFIL and call the procedure PRINTR. The system matches FORMS in order-dependent form. DC=PU switches the mode to order-independent mode. PU indicates the file is to be punched.

```
ATTACH,PROCFIL,ID=INK45.
BEGIN,PRINTR,,FORMS,DC=PU,EC=SB.
```

The following is a segment of the dayfile that results when the BEGIN statement is processed.

```
13.30.36.ATTACH,PROCFIL,ID=INK.
13.30.36.PFN IS
13.30.36.PROCFIL
13.30.36.AT CY= 006 SN=SPFSET
13.30.36.BEGIN,PRINTR,,FORMS,DC=PU,EC=SB.
13.30.36.REQUEST,AAA,Q.
13.30.37.COPY(FORMS,AAA)
13.30.37.ROUTE(AAA,DC=PU,REP=0,EC=SB)
13.30.37.OP  00000192 WORDS - FILE AAA     , DC 10
13.30.37.REVERT.  FORMS ROUTED.
```

Example 2 - Parameter Matching In Nested Procedures (Order-Dependent and Order-Independent Parameter Matching Modes):

As shown in figure 5-3, procedures ROUT and PREPARE reside on the default file PROCFIL. A BEGIN statement within ROUT calls PREPARE. In procedure ROUT the substitution for the FNAME parameter (TEST) is passed to procedure PREPARE by the BEGIN statement. The resulting dayfile is on the right side of figure 5-3.

Attach and Call

```
ATTACH,PROCFIL,ID=SOUL.
BEGIN,ROUT,,TEST,SBF.
```

PROCFIL

```
.PROC,ROUT,FNAME=L,SBF=/SBF,REP=O,EC=B6,DC=PR.
BEGIN,PREPARE,,#FNAME=FNAME.
REQUEST,HOLD,Q.
COPY_SBF(FNAME,HOLD)
ROUTE,HOLD,#DC=DC,#EC=EC,#REP=REP.
REVERT.  FNAME -> PRINTER
EXIT,S.
REVERT,ABORT.  ROUT ERRORS.
```

-EOR-

.
.
.

-EOR-

```
.PROC,PREPARE,FNAME=,ID=SOUL.
IFE,FILE(FNAME,AS),PREP1.
REWIND(FNAME)
REVERT.  FNAME PREPARED.
ENDIF,PREP1.
ATTACH(FNAME,#ID=ID)
REVERT.  FNAME PREPARED.
EXIT,S.
REVERT,ABORT.  FNAME NOT FOUND.
```

Resulting Dayfile

```
11.41.41.ATTACH,PROCFIL,ID=SOUL.
11.41.41.PFN IS
11.41.41.PROCFIL
11.41.42.AT CY= 006 SN=SPFSET
11.41.42.BEGIN,ROUT,,TEST,SBF.
11.41.42.BEGIN,PREPARE,,FNAME=TEST.
11.41.43.IFE,FILE(TEST,AS),PREP1.
11.41.43.ENDIF,PREP1.
11.41.43.ATTACH(TEST,ID=SOUL)
11.41.43.PFN IS
11.41.43.TEST
11.41.43.AT CY= 001 SN=SPFSET
11.41.43.REVERT.  TEST PREPARED.
11.41.44.REQUEST,HOLD,Q.
11.41.44.COPYSBF(TEST,HOLD)
11.41.44.ROUTE,HOLD,DC=PR,EC=B6,REP=O.
11.41.44.OP  00000192 WORDS - FILE HOLD    , DC 40
11.41.44.REVERT.  TEST -> PRINTER.
11.41.44.REVERT.CCL
```

BEGIN calls procedure ROUT. The SBF=/SBF parameter switches parameter matching to order-independent mode. The first control statement of ROUT is a BEGIN statement that calls procedure PREPARE. The parameters are matched in order-independent mode. PREPARE readies a file for processing. If the file is local, it is rewound. If it is not local, the system searches for the file in the user's permanent file catalog. If the file is not found, the procedure reverts and aborts. If the file is found, processing continues with the second control statement in procedure ROUT. The procedure prepares the file for printing, routes the file to the printer, and reverts to the statement following the BEGIN control statement.

Figure 5-3. Keyword Substitution in Nested Procedures

INTERACTIVE PROCEDURES

A specific procedure header statement format identifies a procedure as interactive. Interactive procedures allow the user to have a dialog with the system before the procedure is executed. Interactive sessions or batch jobs may use interactive procedures, but only users on time-sharing terminals can take full advantage of the interactive procedure's following capabilities.

- Getting a description of the procedure.
- Getting a description of each parameter and its acceptable values.
- Prompting by the system for required parameters if they had been omitted from the procedure call.
- Reprompting by the system for a parameter value when an unacceptable entry has been made.

The procedure writer may make these capabilities available to the end user by providing the following information.

- Designating procedure parameters as optional or required.
- Designating permissible values and correct syntax for each parameter through a checklist in the procedure header statement.
- Supplying prompt descriptions for each parameter.
- Supplying directions for each of the parameters and for the procedure with .HELP statements.

The procedure call statement replaces keywords in the procedure body as soon as all required parameters are supplied. Unless the header statement contains an error, it does not appear in the dayfile.

A user can execute an interactive procedure from a noninteractive source as long as all the required parameters are specified on the procedure call. If any required parameters are omitted, the system issues diagnostics to the dayfile specifying parameter values missing or in error, and the procedure is not executed.

Interactive-Procedure Header Statement

The interactive-procedure header statement is the first line of the procedure. It names the procedure, identifies it as interactive, and specifies the keywords to which the procedure call statement or the interactive user can pass values.

The following syntax rules apply to interactive header statements:

- The statement must begin with a period followed by the characters .PROC.
- An *I must be appended to the procedure name; *I identifies the procedure as interactive.
- A comma must separate .PROC and the procedure name.

- The separators between the procedure name and the procedure parameters may be commas (,), reverse slashes ( ), or slashes (/). Reverse slashes and slashes change parameter substitution from order-dependent to order-independent format. (Refer to Interactive Parameter Substitution, later in this section.)
- A period terminates the header statement.
- The header statement can extend over more than one line.

The format of the interactive procedure header is:

.PROC,pname*I,p_1"description$_1$"=(checklist$_1$),p_2"description$_2$"=(checklist$_2$),...,
p_n"description$_n$"=(checklist$_n$).

pname*I

pname is the procedure name (from one to seven alphanumeric characters). The procedure name should begin with an alphabetic character and must not be BEGIN. *I must be appended to the procedure name to indicate that the procedure is interactive.

p_i"description$_i$"=(checklist$_i$)

Optional parameters. The maximum number of parameters is 50.

p_i

A keyword which identifies the parameter. The occurrences of p_i in the procedure body are replaced by a value that must conform to the specifications made in the checklist. This value is specified by the procedure call parameters or by interactively entered parameters. Values entered interactively override procedure call parameters.

description$_i$

An optional 1- to 40- character text string that must be enclosed in quotation marks. The system displays this text string when prompting for a parameter. description$_i$ may be null ("") or omitted.

checklist$_i$

A list of the acceptable values and syntax for p_i. The checklist must be surrounded by parentheses. The value specified for a parameter in a procedure call is compared to each of the entries in the checklist in a left-to-right order. A match must occur for a value to be acceptable.

If a checklist is omitted, the system assumes a checklist containing *A.

A description of acceptable checklist entries follows.

*N=value

If p_i is not specified on the procedure call, value replaces each occurrence of p_i in the procedure body. If only *N= is specified in the checklist, a null value replaces each occurrence of p_i. If only *N is specified, no substitution occurs for p_i. If the *N entry is omitted from a checklist, p_i is a required parameter and interactive prompting occurs when p_i is omitted from the procedure call.

For example, procedure SUB on local file PROCFIL

```
.PROC,SUB*I,P1=(*N),P2=(*N=LGO),P3=(*N=).
COMMENT. #P1=P1, #P2=P2, #P3=P3
```

called by

```
BEGIN,SUB.
```

will produce the procedure body

```
COMMENT.  P1=P1, P2=LGO, P3=
```

Specifying the *N entry more than once in a single checklist is an error.

*K=value

If the keyword (p_i) is specified on a procedure call or interactive entry, value replaces each occurrence of p_i in the procedure body. If only *K= is specified in the checklist, a null value replaces p_i. If only *K is specified, no substitution occurs for p_i.

For example, procedure KEY on local file PROCFIL

```
.PROC,KEY*I,P1=(*K),P2=(*K=INPUT),P3=(*K=).
COMMENT. #P1=P1, #P2=P2, #P3=P3
```

called by

```
BEGIN,KEY,,P1,P3,P2.
```

will produce the procedure body

```
COMMENT.  P1=P1, P2=INPUT, P3=
```

Specifying the *K entry more than once in a single checklist is an error.

*F=value

If the procedure call or the interactive entry specified a file name that conforms to the operating system format for a file name, value replaces each occurrence of p_i in the procedure body. If only *F= is specified in the checklist, a null value replaces each occurrence of p_i in the procedure body. If only *F is specified, the file name specified on the procedure call or the interactive entry replaces all occurrences of p_i in the procedure body.

For example, procedure EXEC on local file EX

```
.PROC,EXEC*I,I=(*F),B=(*N=LGO,*F),L=(*F=OUTPUT).
FTN5(#I=I,#B=B,#L=L)
```

called by

```
BEGIN,EXEC,EX,I=CARDS,L=PRINT.
```

will produce the procedure body

```
FTN5(I=CARDS,B=LGO,L=OUTPUT)
```

*A=value

Anything can be specified for p_i on the procedure call or interactive entry, and value replaces each occurrence of p_i in the procedure body. If only *A= is specified in the checklist, a null value replaces each occurrence of p_i in the procedure body, no matter what is specified for p_i on the call. If only *A is specified, whatever is specified on the procedure call or interactive entry replaces each occurrence of p_i in the procedure body.

*Sn(set) =value

After the set selection criteria have been met, value replaces all occurrences of p_i in the procedure body. The set selection criteria require that the parameter entry for p_i on the procedure call or interactive entry must contain n or fewer characters selected from the specified set. set may contain from 1 to 40 alphanumeric characters, or a literal. n is the maximum number of characters allowed for the p_i parameter entry. If n is omitted on the procedure header statement, the maximum is assumed to be one. If the selection criteria are not met, the user is reprompted for the parameter.

If only *Sn(set)= is specified in the checklist, a null value is substituted for each occurrence of p_i in the procedure body if the set selection criteria are met.

If only *Sn(set) is specified, the parameter entry for p_i on the procedure call or interactive entry replaces all occurrences of p_i in the procedure body if it meets the set selection criteria.

For example, procedure COPIL on local file COPI

```
.PROC,COPIL*I,O"OLD FILE NAME"=(*F,*N=OLD)
,R"REPLACEMENT FILE NAME"=(*F,*N=LGO)
,N"NEW FILE NAME"=(*F,*N=NEW)
,L"LAST RECORD"=(*F,*N=)
,F"FLAG"=(*S4(ARTE),*N=).
COPYL(O,R,N,L,F)
REVERT.
```

called by

```
BEGIN,COPIL,COPI,O=OLD,R=MODIFIED,N=NEW,F=AE.
```

will produce a procedure body of

```
COPYL(OLD,MODIFIED,NEW,,AE)
```

Procedure COPIL will accept up to four letters for F parameter.

More than one set may be specified for a parameter. For example, procedure SET has two sets specified for the P parameter.

```
.PROC,SET*I,P=(*S3(ABC),*S3(XYZ)).
```

Parameter entries could include P=BB or P=XZY, but not P=AZ or P.

Null sets are not allowed; in the following procedure header all set entries in the P1 checklist are illegal.

```
.PROC,SET*I,P1=(*S,*S(),*S3).
```

string=value

If the parameter on the procedure call or interactive entry matches string, value is substituted for each occurrence of p_i in the procedure body. A string may contain from 1 to 40 alphanumeric characters, or a literal. If only string= is specified in the checklist and the parameter on the procedure call or interactive entry matches string, a null value replaces p_i. If only string is specified and the parameter on the procedure call or interactive entry matches string, string replaces p_i.

For example, procedure LABL on local file PROCFIL

```
.PROC,LABL*I,FN"FILE NAME"=(*F)
,LT"LABEL TYPE Y OR Z"=(Y,Z)
,WRITE"YES OR NO"=(YES=W,NO=R)
,RING"YES OR NO"=(YES=RING,NO=NORING).
LABEL(FN,LT,WRITE,RING)
REVERT.
```

called by

```
BEGIN,LABL,,FN=TAPE1,LT=Z,WRITE=YES,RING=YES.
```

will produce a procedure body of

```
LABEL(TAPE1,Z,W,RING)
```

Interactive Procedure Body

Procedure bodies are the same whether or not the procedure header statement is interactive, except that interactive procedures may use .HELP and .ENDHELP statements. (Refer to Procedure Body, earlier in this section.)

Interactive Processing

A procedure call statement initiates the execution of an interactive procedure. The procedure call may have the same format for an interactive procedure as it has for a noninteractive procedure. In addition, on an interactive procedure call, an interactive user may:

- Request a description of the procedure.
- Request a description of a procedure parameter.
- Omit required parameters.
- Specify an incorrect procedure parameter name.

If any of the preceding conditions occur, and if the interactive procedure had been called from an interactive source, the system initiates an interactive dialogue with the user.

If descriptions are requested, the system lists optionally provided descriptive text. Procedure or parameter descriptions can be requested by doing one of the following:

- Appending a question mark to the procedure file name.
- Appending a question mark to the name of a procedure parameter name.
- Entering a question mark as a parameter on the procedure call.
- Entering a question mark in response to an interactive prompt.

In the following example, the first two calls request the description of procedure LIST and initiate prompting for all procedure parameters. The third call requests the description of the parameter KEY and prompts for the KEY parameter entry and all other parameters on the header statement.

```
BEGIN,LIST,FILE?
BEGIN,LIST,FILE,?
BEGIN,LIST,FILE,KEY?
```

When the system encounters a question mark in a procedure call statement, it stops reading the call statement and starts help processing. Anything entered after the question mark, therefore, will not be read.

If required parameters are omitted, or if any parameter is in error on the procedure call, CCL prompts the user for those parameters. If CCL prompts the user for a parameter that has an *N in its checklist, and if the user wants to omit that parameter, %EOR or %EOF must be entered. If the format of a parameter entry is not correct according to the parameter checklist, or if the parameter entry is not specified on the procedure header statement, the user is reprompted for the parameter. Prompting for parameters terminates when any of the following situations occur.

- All parameter requirements have been satisfied.
- The user enters a parameter terminated by a period or right parenthesis, or enters a period or right parenthesis. If all required parameters have been entered, the system executes the procedure. Otherwise, the system continues prompting until all required parameters are satisfied.
- The user enters the terminal abort command (user break, %A). The call statement and interactive dialogue are terminated. The procedure is not executed.

The following example shows the interactive entering of parameters.

Procedure FTN5 resides on local file F5.

```
.PROC,FTN5*I,I"INPUT"=(*F,*N=INPUT)
,B"BINARIES"=(*F,*N=LGO)
,L"OUTPUT"=(*F,*N=OUTPUT)
,LO"LIST OPTIONS"=(*N=O,O,O,R,A,M,S).
FTN5(#I=I,#B=B,#L=L,#LO=LO)
REVERT.
```

To be prompted for the parameters on the procedure, the user enters

```
F5,?
```

at the terminal.

System responses (uppercase) and user entries (lowercase) appear as follows:

```
PARAMETERS FOR FTN5 ARE I, B, L, LO
ENTER I INPUT
i?
MAY BE A FILE NAME
PARAMETER MAY BE OMITTED

ENTER I INPUT
example
ENTER B BINARIES
b?
MAY BE A FILE NAME
PARAMETER MAY BE OMITTED

ENTER B BINARIES
%eor
ENTER L OUTPUT
l?
MAY BE A FILE NAME
PARAMETER MAY BE OMITTED

ENTER L OUTPUT
lfile
ENTER LO LIST OPTIONS
lo?
ALLOWABLE VALUE(S)
    O
    O
    R
    A
    M
    S
PARAMETER MAY BE OMITTED

ENTER LO LIST OPTIONS
O
      70000 SCM STORAGE USED.
      0.022 CP SECONDS COMPILATION TIME.
```

When an interactive procedure is called from a non-interactive source and descriptions of the procedure or parameters are requested, the descriptions are written to the dayfile. If required parameters are omitted from a call on a non-interactive source, error messages are written to the dayfile. In either case, the procedure is not executed.

Interactive Procedure Parameter Substitution

The interactive .PROC procedure header specifies keywords used in the procedure body. A user may change the value of these keywords each time the procedure is executed by specifying the appropriate parameters in the procedure call statement or during the interactive entry of a parameter. The values of these parameters must conform to the restrictions specified in the parameters' checklists.

Control statements sometimes expand beyond 80 characters after substitutions for keywords have been made in the procedure body. For most control statements, this will be flagged as an error. The user should ensure that a line containing keywords is short enough so that possible expansion does not extend the line beyond the 80th character. CCL, LABEL, and VSN statements are among those statements which may extend one line, as long as the statement splits at a separator.

When a procedure is called, CCL must match each parameter on the call statement with a parameter on the procedure header statement. CCL uses two methods of parameter matching; order-dependent and order-independent.

Parameter matching always begins in order-dependent mode. CCL compares, in order, each parameter on the procedure call statement with the parameter in the same position on the procedure header statement. If any of the entered parameters do not conform to the restrictions in the parameter checklist, or required parameters are omitted on the procedure call, the system prompts the user for those procedures. After all required parameters have been entered, CCL substitutes the selected keywords into the procedure body.

For example, assuming that the following procedure is on a local file named ITEMIZE

```
.PROC,ITEMIZE*I,F"LOCAL FILE NAME"=(*N=LGO,*F)
,L"NAME OF LIST OUTPUT FILE"=(*N=OUTPUT,*F)
,BL"EACH FILE START ON NEW PAGE? YES OR NO"=(YES=$,BL$,NO=,*N=)
,NR"REWIND BEFORE & AFTER? YES OR NO"=(YES=,NO=$,NR$,*N=).
ITEMIZE(F,#L=L  BL  NR)
REVERT.
```

The procedure call

```
ITEMIZE,LIST,,NO,NO.
```

matches all parameters in order-dependent mode and produces a procedure body of

```
ITEMIZE(LIST,L=OUTPUT  ,NR)
```

In order-dependent mode, CCL treats excess parameters on a procedure call statement as a non-fatal error.

CCL switches to order-independent mode if, in the comparison of a procedure call statement parameter with an interactive-procedure header statement parameter, one of the following conditions occurs.

- A call statement parameter is in the format keyword=value.
- A reverse slash (\) separates two parameters on the interactive procedure header or call statement.
- A slash (/) separates two parameters on the interactive procedure header statement.
- A slash (/) separates two parameters on the call statement and a slash is also used as a separator in the procedure header. If a slash is specified on the call statement and not on the header statement, the slash is not treated as a separator, but as part of the parameter value. This feature can be helpful in using EDITLIB directives in a procedure. (Refer to the ADD and REWIND directives in the EDITLIB subsection of section 4.)

The parameter-matching mode cannot switch back from order-independent to order-dependent mode.

Once in order-independent mode, CCL matches each successive keyword of a call statement or interactive entry to the identical keyword in the procedure header statement, regardless of the order of the procedure header parameters.

The preceding example, which showed the order-dependent parameter matching mode, has been slightly modified in the following example. An *K entry has been added to the BL parameter checklist to make BL a valid parameter entry. A reverse slash is being used as a separator before the L parameter to ensure order-independent parameter matching mode for all parameters after the local file name parameter, F.

```
.PROC,ITEMIZE*I,F"LOCAL FILE NAME"=(*N=LGO,*F)
\L"NAME OF LIST OUTPUT FILE"=(*N=OUTPUT,*F)
,BL"EACH FILE START ON NEW PAGE? YES OR NO"=(YES=$,BL$,NO=,*N=,*K=$,BL$)
,NR"REWIND BEFORE & AFTER? YES OR NO"=(YES=,NO=$,NR$ *N=).
ITEMIZE(F,#L=L BL NR)
REVERT.
```

The procedure call

```
ITEMIZE,LIST,,NR=NO,BL.
```

starts parameter matching in order-dependent mode. The reverse slash in the procedure header switches parameter matching to order-independent mode. In order-independent mode, the user must specify all parameters in the form keyword=value, unless there is an *K entry in the parameter checklist. If an *K entry is in the parameter checklist, the user may specify the keyword alone as the parameter entry. Since *K is in the BL parameter checklist, the system accepts BL as a parameter entry. The NR parameter must be specified as NR=value.

The procedure body appears as follows:

```
ITEMIZE(LIST,L=OUTPUT ,BL ,NR)
```

.HELP Statement

The .HELP statement allows the procedure writer to provide descriptions of the procedure and/or its parameters in the procedure. The procedure user can access these descriptions interactively by entering a question mark as a parameter in a procedure call, or by appending a question mark to a procedure or parameter name. (Refer to Interactive Processing, earlier in this section.)

When parameter descriptions are requested, the system displays the following information:

- Text information following the .HELP statement for the parameter.
- Description string specified with the parameter on the procedure header statement.
- Parameter values acceptable according to parameter checklist specifications.
- Current value, if any, of the parameter, and prompts for parameter values.

When the procedure description is requested, the system displays the following information.

- Text following the procedure's .HELP statement.
- All parameters on the procedure header and prompts for parameter values.

All .HELP statements, if specified, must immediately follow the procedure header statement. A terminator must not be specified on a .HELP statement.

The format of the .HELP statement is:

.HELP,param,NOLIST

param	Param must be a parameter in the procedure header statement. Text which follows this .HELP statement describes the parameter. Omitting param creates a .HELP statement for the procedure.
NOLIST	If param is specified, NOLIST prohibits the display of acceptable parameter values. If param is omitted from the .HELP statement, NOLIST prohibits the display of the procedure parameter list.

Text description must start on the line following the .HELP statement. Text description can span lines and is not subject to parameter substitution.

Examples of the .HELP statement are provided in the ENDHELP Statement subsection .

.ENDHELP Statement

The .ENDHELP statement specifies the end of the help descriptions. Only one .ENDHELP statement is allowed in a procedure. A terminator must not be specified on the .ENDHELP statement and nothing else can appear on the same line.

The format of the .ENDHELP statement is:

```
.ENDHELP
```

Example: Use of .HELP and .ENDHELP Statements

Procedure ROUT, on file PRINT, verifies that the selected file has been attached before it routes the file to a printer.

```
.PROC,ROUT*I,F"FILE NAME"=(*F)
,DC=(*N=PR,PR,LR,LS,LT).
.HELP
THIS PROCEDURE ROUTES A PERMANENT FILE TO THE SELECTED LINE PRINTER.
.HELP,F
THE NAME OF THE PERMANENT FILE TO BE ROUTED.
.HELP,DC
THE DISPOSITION CODE.  DC ONLY ACCEPTS LINE PRINTER OPTIONS.
.ENDHELP
IFE,FILE(F,.NOT.PF),PF.
ATTACH(F,ID=SHE)
ENDIF,PF.
REQUEST,Z,Q.
COPYSBF(F,Z)
ROUTE(Z,#DC=DC)
```

Directions on how to use the procedure are obtained by appending a question mark to the file name, PRINT.

```
ATTACH(PRINT,ID=SHE)
PRINT?
```

System responses and user input follows. System responses appear in uppercase letters, and user input appears in lowercase letters.

```
THIS PROCEDURE ROUTES A PERMANENT FILE TO THE SELECTED LINE PRINTER.
PARAMETERS FOR ROUT ARE  F,  DC
ENTER F FILE NAME
f?
MUST BE A FILE NAME
THE NAME OF THE PERMANENT FILE TO BE ROUTED.

ENTER F FILE NAME
tues
ENTER DC
dc?
ALLOWABLE VALUES (S)
              PR
              LR
              LS
              LT
PARAMETER MAY BE OMITTED
THE DISPOSITION CODE.  DC ONLY ACCEPTS LINE PRINTER OPTIONS.   NS.

ENTER DC
%eor
  PFN  IS
  TUES
  AT CY= 002 SN-SPFSET
..
```

PARAMETER ALTERATION

When specifying keywords in the procedure body, two special characters, ASCII graphics # and _ (or Control Data graphics ≡ and ↱), are used to inhibit keyword substitution and to combine parts of a parameter after keyword substitution. These characters may be used in interactive and noninteractive procedures.

A single # character placed immediately before a keyword in a procedure statement inhibits substitution for that keyword. Two such characters (##) placed immediately before a keyword allow substitution; one # is retained. The # does not affect a separator or nonkeywords.

The linking character, underline (_), is used in a procedure statement to temporarily separate two parameters (keyword or nonkeyword). After possible substitutions are made, the underline character is removed and the two parameters are merged into one. # before _ retains _ and allows substitution. _ before # does not affect the inhibiting action of #.

Examples of use of the # and _ characters in a procedure are shown in table 5-3. Because the call statement in table 5-3 has no keywords, the defaults from the procedure header are used unless the # character inhibits the substitution.

TABLE 5-3. ALTERATIONS OF PARAMETERS IN A PROCEDURE BODY BY USE OF # AND _

Call statement: BEGIN, DATE, APROCFL.		
Procedure header: .PROC,DATE,DAY=19,MONTH=02.		
Procedure Parameters in Procedure Body Before Substitution	Procedure Parameters in Procedure Body After Substitution	Comment
#DAY,DAY	DAY,19	# inhibits substitution in the keyword that immediately follows and does not effect nonkeywords.
#I,J	I,J	
DAY#MONTH	19MONTH	
##DAY,MONTH	#19,02	## allows substitution if a keyword immediately follows; one # is retained.
##I,J	#I,J	
DAY#,DAY	19,19	# does not affect a separator.
DAY_MONTH	1902	_ separates two parameters before substitutions are made; after all substitutions are made, they are joined into one parameter.
I_J	IJ	
DAY_J	19J	
J_MONTH	J02	
DAY#_DAY	19_19	# before _ retains _ and allows substitution.
DAY#_MONTH	19_02	
DAY_#DAY	19DAY	_ before an # does not affect the inhibiting action of the #.

Example 1 - # Character:

The following procedure resides on file PROCFIL.

```
.PROC,INHIBIT,I=TEST.
ATTACH(I,ID=HUSH)
FTN5(#I=I,L=0)
LGO.
COMMENT.  I, #I, I#I. #I#I.
```

On the left are the ATTACH and BEGIN statements that attach file PROCFIL and call procedure INHIBIT. On the right is the resulting dayfile.

```
ATTACH,PROCFIL,ID=HUSH.
BEGIN,INHIBIT.
```

```
11.45.54.ATTACH,PROCFIL,ID=HUSH.
11.45.54.PFN IS
11.45.54.PROCFIL
11.45.55.AT CY= 006 SN=SPFSET
11.45.55.BEGIN,INHIBIT.
11.45.55.ATTACH(TEST,ID=HUSH)
11.45.56.PFN IS
11.45.56.TEST
11.45.56.AT CY= 001 SN=SPFSET
11.45.56.FTN5(I=TEST,L=0)
11.45.57.      60000 SCM STORAGE USED.
11.45.57.      0.029 CP SECONDS COMPILATION TIME.
11.45.57.LGO.
11.45.59.     STOP
11.45.59.      15700 MAXIMUM EXECUTION FL.
11.45.59.       .027 CP SECONDS EXECUTION TIME.
11.45.59.  TEST, I, TESTI. II.
11.45.59.REVERT.CCL
```

Example 2 - # Character:

The following procedure file is a permanent file called COLORPR. It uses the IFE statement to determine if the color the BEGIN statement substituted for COLOR is red or blue. Different processing is done for the colors red and blue. Any other color is ignored. The # character in the comment line inhibits substitution for the word (COLOR) it precedes.

```
.PROC,A*I,COLOR=(*A).
IFE,$COLOR$.EQ.$RED$,L1.
COMMENT.  PROCESSING DONE FOR #COLOR OF COLOR
REVERT.
ENDIF,L1.
IFE,$COLOR$.EQ.$BLUE$,L2.
COMMENT.  PROCESSING DONE FOR #COLOR OF COLOR
REVERT.
ENDIF,L2.
COMMENT.  NO PROCESSING DONE FOR #COLOR OF COLOR
```

The following control statements call procedure A.

```
ATTACH,COLORPR,ID=PIGMENT.
BEGIN,A,COLORPR,BLUE.
BEGIN,A,COLORPR,RED.
BEGIN,A,COLORPR,PINK.
```

The following dayfile segment results when the preceding control statements are processed. It shows the effect of the # character.

```
11.36.47.ATTACH,COLORPR,ID=PIGMENT.
11.36.47.PFN IS
11.36.47.COLORPR
11.36.48.AT CY= 002 SN=SPFSET
11.36.48.BEGIN,A,COLORPR,BLUE.
11.36.48.IFE,$BLUE$.EQ.$RED$,L1.
11.36.48.ENDIF,L1.
11.36.48.IFE,$BLUE$.EQ.$BLUE$,L2.
11.36.48.  PROCESSING DONE FOR COLOR OF BLUE
11.36.48.REVERT.
11.36.48.BEGIN,A,COLORPR,RED.
11.36.49.IFE,$RED$.EQ.$RED$,L1.
11.36.49.  PROCESSING DONE FOR COLOR OF RED
11.36.49.REVERT.
11.36.49.BEGIN,A,COLORPR,PINK.
11.36.49.IFE,$PINK$.EQ.$RED$,L1.
11.36.49.ENDIF,L1.
11.36.49.IFE,$PINK$.EQ.$BLUE$,L2.
11.36.49.ENDIF,L2.
11.36.49.  NO PROCESSING DONE FOR COLOR OF PINK
11.36.50.REVERT.CCL
```

Example 3 - Character:

Procedure LINK resides on file LFILE.

```
.PROC,LINK,TYPE=SBF,LFN1,LFN2.
REWIND(LFN1)
COPY_TYPE(LFN1,LFN2)
```

The first BEGIN statement does a COPYSBF of file PLAN to file SCHEME. The next BEGIN statement does a COPY of file MAZE to file TAXES. The resulting dayfile segment follows the BEGIN statements. LFILE is already attached.

```
BEGIN,LINK,LFILE,TYPE=SBF,LFN1=PLAN,LFN2=SCHEME.
BEGIN,LINK,LFILE,TYPE=,LFN1=MAZE,LFN2=TAXES.
```

```
11.45.59.BEGIN,LINK,LFILE,TYPE=SBF,LFN1=PLAN,LFN2
11.45.59.=SCHEME.
11.46.00.REWIND(PLAN)
11.46.00.COPYSBF(PLAN,SCHEME)
11.46.00.REVERT.CCL
11.46.01.BEGIN,LINK,LFILE,TYPE=,LFN1=MAZE,LFN2=TA
11.46.01.XES.
11.46.01.REWIND(MAZE)
11.46.01.COPY(MAZE,TAXES)
11.46.02.REVERT.CCL
```

PROCEDURE COMMANDS

Procedure commands enable the user to format a data file and to insert documentary comments within a procedure. The commands are in fixed format with a period in column 1 and the command name beginning in column 2.

.DATA Command

A .DATA command in a procedure marks the beginning of a sequence of data lines to be written to a separate file when the procedure is called. File marks generated by .EOR and .EOF commands can subdivide the lines written to the data file. The sequence of data lines is terminated by one of the following:

- Another .DATA command.
- A system end-of-record (not an .EOR command).
- A system end-of-partition (not an .EOF command).
- A system end-of-information.

The data file created does not include the .DATA command. Keyword substitution occurs within the data statements.

The format of .DATA is:

.DATA,lfn

lfn Optional name of the file on which the data lines are to be written. If a file named lfn is already attached to the job, it is released, and new local file lfn is created. After the data file is written, it is automatically rewound. If lfn is specified, lfn references the data file and not the special default #DATA.

If lfn is specified, the separator between DATA and lfn can be a comma, a left parenthesis, or a blank space. The terminator after the lfn can be a period, a right parenthesis, or a blank space. If the user wants to include comments, a period or a right parenthesis must terminate the command.

If lfn is omitted, the user references the data file with the special default #DATA. At the first procedure level, the system calls this file ZZCCLAA; at the second procedure level it is called ZZCCLAB; and so forth.

The following examples show three different ways of inserting a FORTRAN program into a noninteractive procedure.

Example 1 - Procedure Accesses Program Data With .DATA Command:

The following procedure file is a permanent file named DATAFIL.

```
.PROC,ALPHA,P1=#DATA,X=OUTFILE.
FTN5(I=P1,L=X)
LGO.
CATALOG(X,LISTFIL,ID=FRAN,RP=15)
REVERT.
.DATA.    FORTRAN PROGRAM.
      PROGRAM X(OUTPUT)
        .
        .
        .
        .
      END
```

The following call statement in the control statement record of the job accesses procedure ALPHA on file DATAFIL. A previous ATTACH statement made DATAFIL local to the job.

```
BEGIN,APLHA,DATAFIL,X=FTNOUT.
```

A sample of a resulting dayfile is:

```
16.04.09.BEGIN,ALPHA,DATAFIL,X=FTNOUT.
16.04.16.FTN5(I=ZZCCLAA,L=FTNOUT)
16.04.17.      60700 SCM STORAGE USED.
16.04.17.      0.035 CP SECONDS COMPILATION TIME.
16.04.17.LGO.
16.04.19.      END  FTNOUT
16.04.19.      16100 MAXIMUM EXECUTION FL.
16.04.19.       .026 CP SECONDS EXECUTION TIME.
16.04.20.CATALOG(FTNOUT,LISTFIL,ID=FRAN,RP=15)
16.04.20.NEWCYCLE CATALOG
16.04.21.CT ID=    FRAN PFN=LISTFIL
16.04.21.CT CY= 007 SN=SPFSET  00000128 WORDS.
16.04.21.REVERT.CCL
```

All input after the .DATA command (the FORTRAN source program) is written onto the default temporary file ZZCCLAA. Parameter substitution occurs in the data statements, as shown by FTNOUT replacing the program name X in the END dayfile line.

Example 2 - Procedure Accesses Program Data With Special Default #FILE:

The following noninteractive procedure is a permanent file named PFILE. The record immediately following procedure BETA contains the program data. The #FILE default tells the FTN5 compiler to search for input from the next record on file BETA.

```
.PROC,BETA,P1=#FILE,X=OUTFILE.
FTN5(I=P1,L=X)
LGO.
CATALOG(X,LISTFIL,ID=WWW,RP=15)
*EOR
      PROGRAM X(OUTPUT)
        .
        .
        .
        .
      END
```

After file PFILE is attached, the following call accesses procedure BETA.

```
BEGIN,BETA,PFILE,X=FTNOUT.
```

The following is a segment of the resulting dayfile. Parameter substitution occurs within the procedure but not within the FORTRAN program.

```
16.04.37.BEGIN,BETA,PFILE,X=FTNOUT.
16.04.37.FTN5(I=PFILE,L=FTNOUT)
16.04.39.      60700 SCM STORAGE USED.
16.04.39.      0.037 CP SECONDS COMPILATION TIME.
16.04.39.LGO.
16.04.41.      END  X
16.04.41.      16100 MAXIMUM EXECUTION FL.
16.04.41.       .023 CP SECONDS EXECUTION TIME.
16.04.41.CATALOG(FTNOUT,LISTFIL,ID=WWW,RP=15)
16.04.41.NEWCYCLE CATALOG
16.04.42.CT ID=   WWW PFN=LISTFIL
16.04.42.CT CY= 009 SN=SPFSET  00000128 WORDS.
16.04.42.REVERT.CCL
```

Figure 5-4 shows diagrammatically how #FILE is used to access program data.

Example 3 - Procedure Accesses Program Data From Another File:

To access program data outside of the procedure file, the procedure must include an ATTACH control statement. The following procedure is on file PROCFIL.

```
.PROC,GAMMA,P1=PRGRAM1,X=OUTFILE.
ATTACH,P1,ID=WIND.
FTN5(I=P1,L=X)
LGO.
CATALOG(X,LISTFIL,ID=WIND,RP=20)
```

The following statements attach PROCFIL and call procedure file GAMMA.

```
ATTACH,PROCFIL,ID=WIND.
BEGIN,GAMMA,,X=FTNOUT.
```

The following is the resulting dayfile segment. Parameter substitution occurs within the procedure but not within the FORTRAN program.

```
16.04.22.ATTACH,PROCFIL,ID=WIND.
16.04.23.PFN IS
16.04.23.PROCFIL
16.04.23.AT CY= 008 SN=SPFSET
16.04.23.BEGIN,GAMMA,,X=FTNOUT.
16.04.24.ATTACH,PRGRAM1,ID=WIND.
16.04.24.PFN IS
16.04.24.PRGRAM1
16.04.24.AT CY= 002 SN=SPFSET
16.04.24.FTN5(I=PRGRAM1,L=FTNOUT)
16.04.26.      60700 SCM STORAGE USED.
16.04.26.      0.030 CP SECONDS COMPILATION TIME.
16.04.26.LGO.
16.04.33.      END  X
16.04.33.      16100 MAXIMUM EXECUTION FL.
16.04.33.       .025 CP SECONDS EXECUTION TIME.
16.04.33.CATALOG(FTNOUT,LISTFIL,ID=WIND,RP=20)
16.04.33.NEWCYCLE CATALOG
16.04.34.CT ID=   WIND PFN=LISTFIL
16.04.34.CT CY= 008 SN=SPFSET  00000128 WORDS.
16.04.34.REVERT.CCL
```

Figure 5-4 shows diagrammatically how ATTACH is used to access program data. An example of a data file written from a procedure to a named file is shown in figure 5-5.

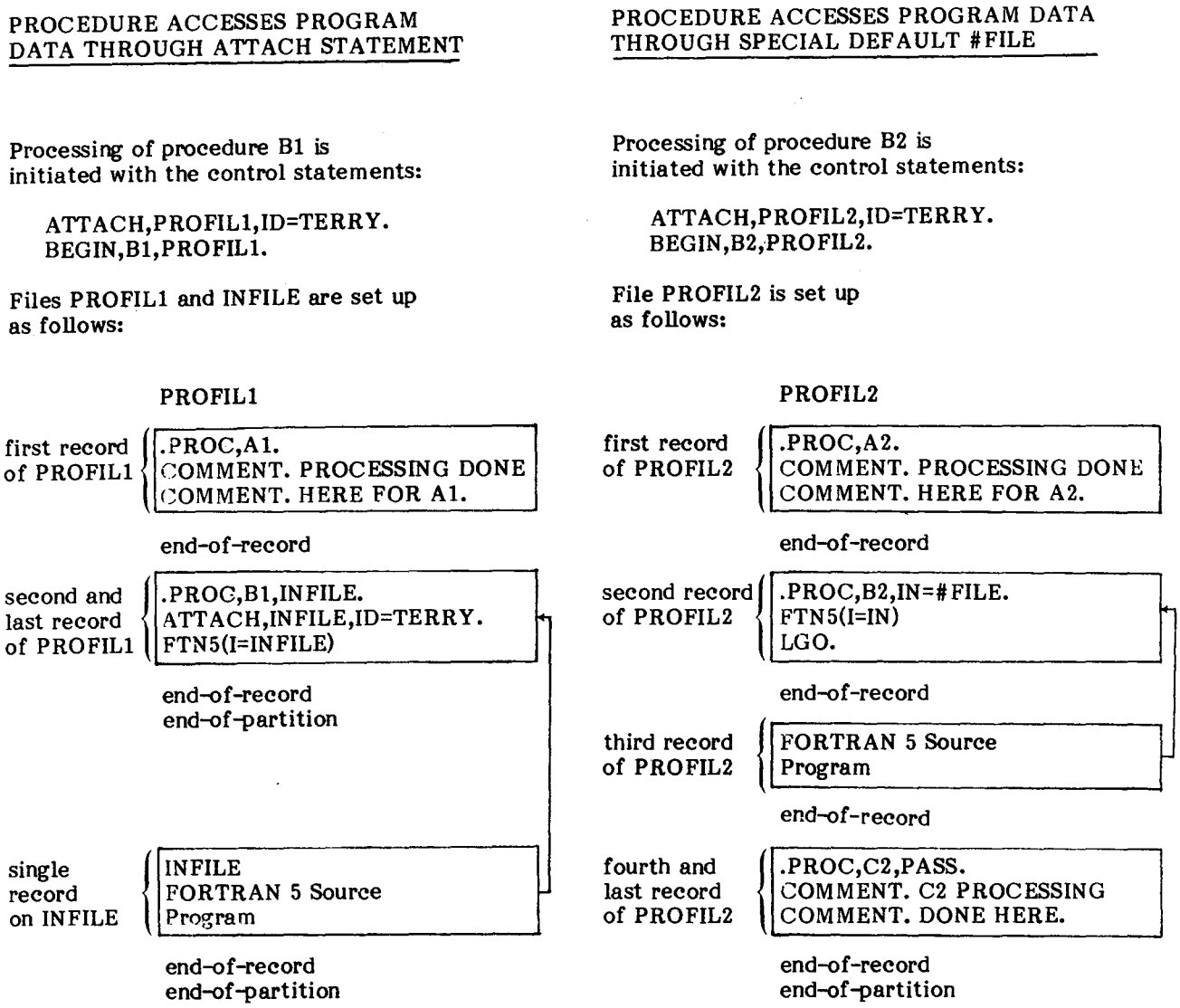


Figure 5-4. Procedure Access to Program Data

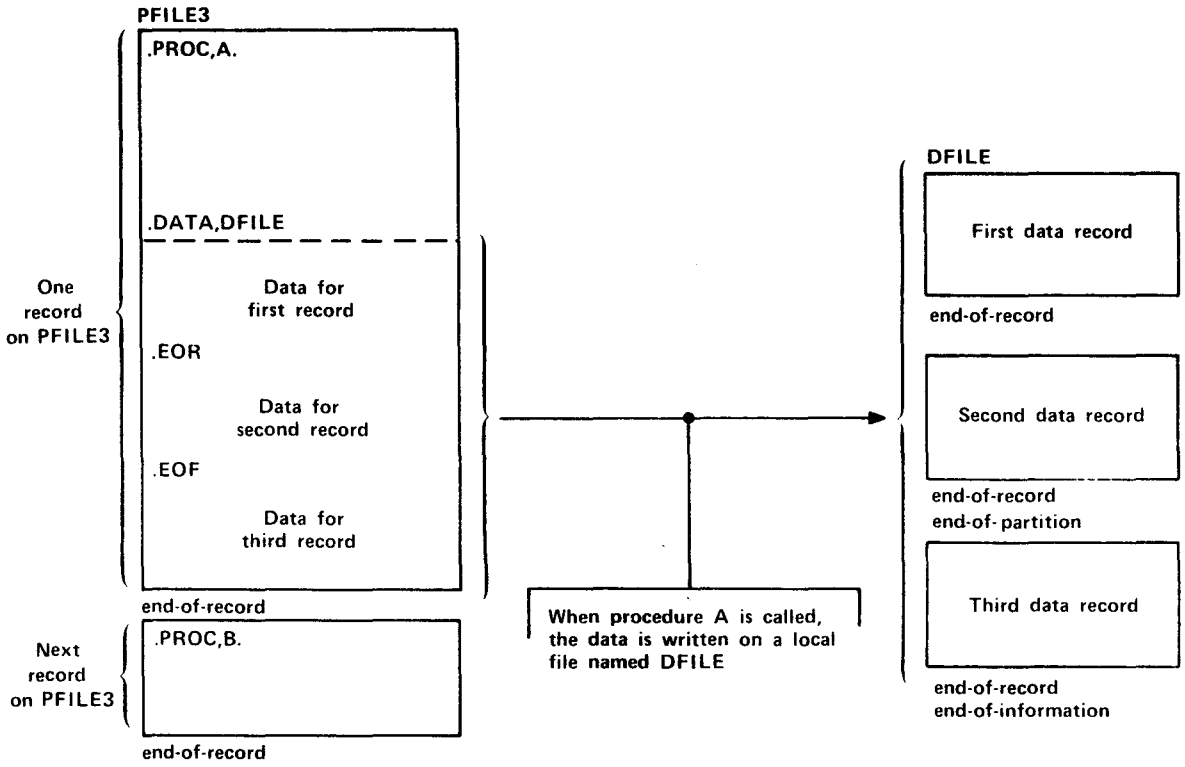


Figure 5-5. Data File Written from a Procedure to a Named File

.EOR Command

The .EOR command is used to separate records in a data file originating in a procedure. Whenever an .EOR is placed, an actual end-of-record is recorded when the data file is written on #DATA or lfn. Since the data statements are written on an external file, the .EOR command has no effect on the system end-of-record on the procedure file that terminates the procedure. The .EOR command is valid only after a .DATA command (figure 5-5). A terminator must not be used and nothing else can appear on the same line.

.EOF Command

The .EOF command generates an end-of-partition on a data file originating in a procedure. An actual end-of-partition is recorded when the data statements are written on #DATA or lfn. This command has no effect on the end-of-record that terminates the procedure. If the end of the data file format is also the end of the procedure, no .EOF command is needed. In this case, an end-of-record mark is added. If the user wants an end-of-partition mark, an .EOF command must be included. The .EOF command is valid only after a .DATA command (figure 5-5). A terminator must not be used, and nothing else can appear on the same line.

.* Command

The .* command enables the user to document a procedure with internal comments. These comments appear when the file is copied to output or displayed at a terminal; they do not appear in the dayfile when the procedure is processed. The comment, which follows the *, can contain any combination of characters.

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 CDC CYBER Record Manager is used for input/output, the user need supply only the file information table (FIT) data. CDC CYBER Record Manager will construct and manipulate the FET from information in its FIT. The FIT is fully described in the CDC CYBER Record Manager Manuals.

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 CDC CYBER Record Manager manipulation. A description exists in the reference manuals for CDC 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.

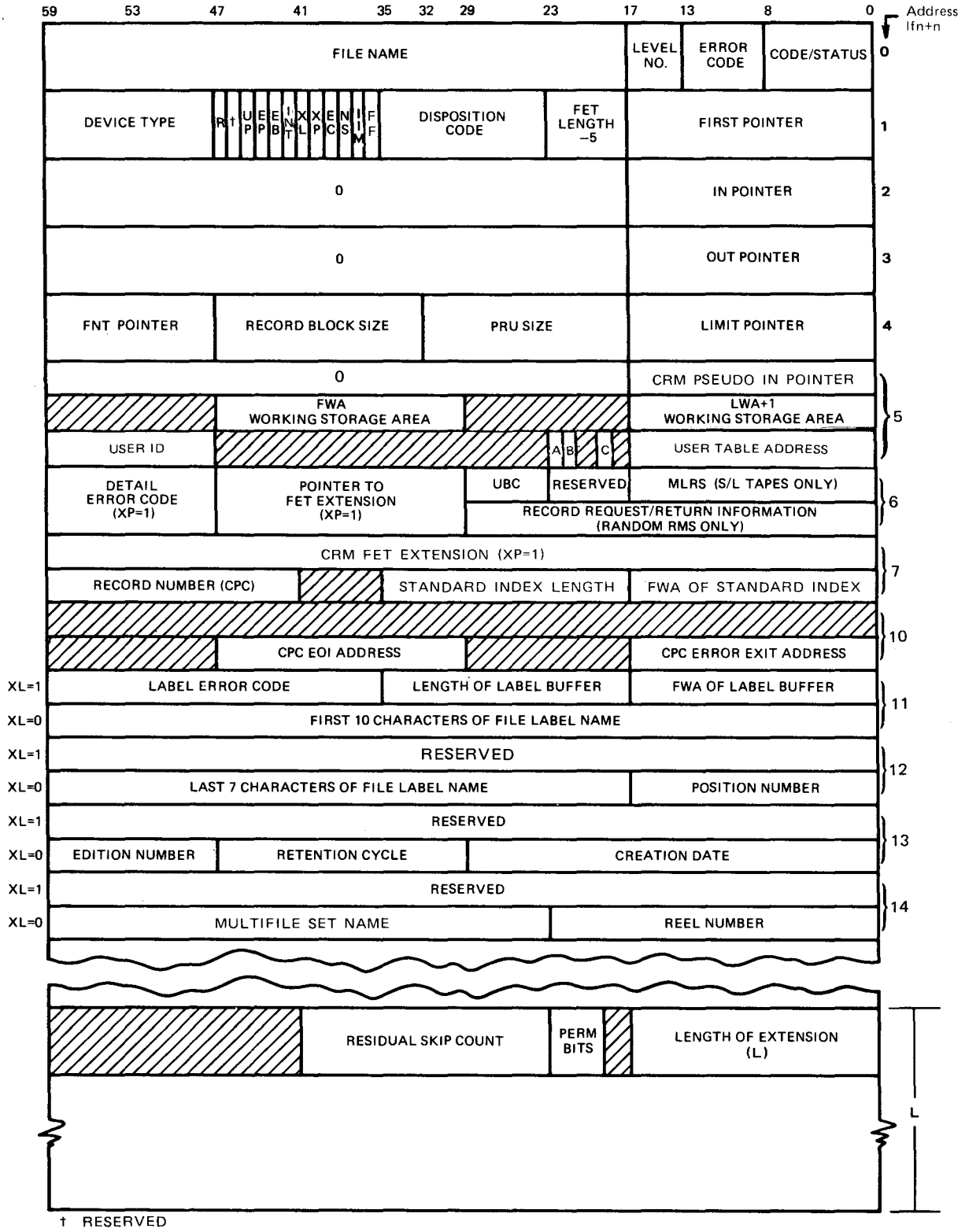


Figure 6-1. File Environment Table

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,1w),(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	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 (lfn+1)
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 (lfn+1); 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+14 octal. All parameter values are octal unless otherwise noted. Bits are numbered 0-59 right to left in decimal.

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 (lfn+0) 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. Non-zero 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 not listed are illegal. All codes are shown for coded mode operations; add 2 for binary mode (for example, 010 is coded READ, 012 is binary READ). Upon completion of operation, code/status in FET is changed to an odd number, ususally by adding 1 to the code. In some cases, code is further modified to indicate manner in which operation concluded [for 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_8 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.

Device Type Mnemonic	Device Type Value	Device
–	01-05	Reserved for CDC
–	06	Reserved for installations
–	07-12	Reserved for CDC
AY	13	844-21 disk drive
AZ	14	844-41 disk drive
AH	15	819 disk drive
–	16	Reserved for CDC
AJ	17	885 disk drive
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.

Device Type Mnemonic	Device Type Value (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

Device Type Mnemonic	Device Type Value (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.

Device Type Mnemonic	Device Type Value (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
–	52	Reserved for installations
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 CDC 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 non-fatal 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 or the Record Request/Return Information field in FET word 7 (lfn+6) = 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 (lfn+6), or written at the end-of-information; and the logical disk address is returned to FET word 7 (lfn+6).

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.

INTERCOM (INT) (bit 42 at lfn + 1)

Set to allow use of the INTERCOM word (lfn + 5). The INTERCOM word makes ASCII 256 mode, ASCII 128 mode, or multi-line reads available for terminal input/output. The user ID and user table address also 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 (lfn + 6) 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 (lfn + 6) must be set.

EC (bit 39 at lfn + 1)

This bit is reserved for the 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.

INHIBIT IMPLICIT MOUNT (IIM) (bit 37 at lfn + 1)

This bit can be set to prevent implicit mounting of disk devices which would otherwise occur in situations such as disk overflow or new file assignment.

FILE FLUSH (FF) (bit 36 at lfn + 1)

Setting this bit to 1 indicates that a sequential scratch file will be flushed (unwritten data sent to mass storage) if the job ends abnormally.

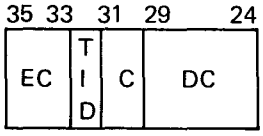
DISPOSITION CODE (bits 35-24 at lfn + 1)

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=O26	100	BCD 64-character print train/punch O26	--/--
EC=A6/EC=O29	101	ASCII 64-character print train/punch O29	--/--
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
IC=TRANS	11	File format is transparent (INTERCOM 5)	----/----

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
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 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, the pointer is not updated.

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.

INTERCOM (lfn + 5)

If bit 42 (INT) of lfn + 1 is set, five fields are defined in lfn + 5. Bits 59-48 contain the user ID. Bit 23, if set by the user, specifies ASCII 256 mode. Bit 22, if set by the user, specifies ASCII 128 mode. Bit 19, if set by the user, specifies multi-line reads. Bits 17-0 contain the user table address.

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:

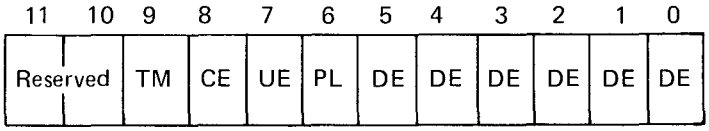
Error Code (Octal)	Data Path Error
400	Hardware – 6681 or 6683 malfunction
402	Hardware – 6681 failed, no data on IAN
403	Hardware – transmission parity error
404	System error

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

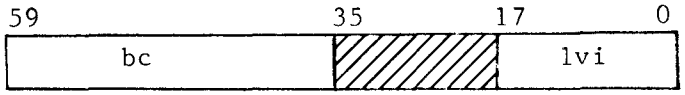
The references to system noise record and last good record refer to procedures the system follows in recovery attempts.

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.

The following errors will cause a parity error notification word to be inserted in the circular buffer at the position pointed to by IN.

Error Code (Octal)	Description
24	Read error in opposite mode (binary/coded)
27	Record fragment possible
1000	Tape parity error

The format of the word is:



bc Block count.
lvi Last valid IN pointer before the error was detected.

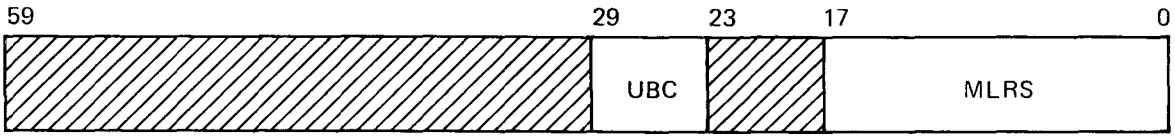
FET EXTENSION POINTER (bits 30-47 at lfn + 6)

When the XP 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 READSKIP 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 FET 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. If the field is zero at the start of a request, then the file is treated as sequential regardless of the setting of the random access bit.

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 + 10)

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 + 10)

The 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 returned.
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 file name assigned to the non-existent 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.

†This error will cause a parity error notification word to be inserted in the circular buffer at the position pointed to by IN. For further information, refer to Detail Error Code earlier in this section.

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 (lfn + 0) of the FET as it appeared before those bits were zeroed.

LABEL PARAMETERS (lfn + 11 through lfn + 14)

Words at lfn + 11 through lfn + 14 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 (lfn + 1). 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 (lfn + 10) 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. Upon entry of the OWNCODE routine, 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 (lfn + 1) of the FET. Extended label processing is possible only when this bit is set. An explicit open is required.

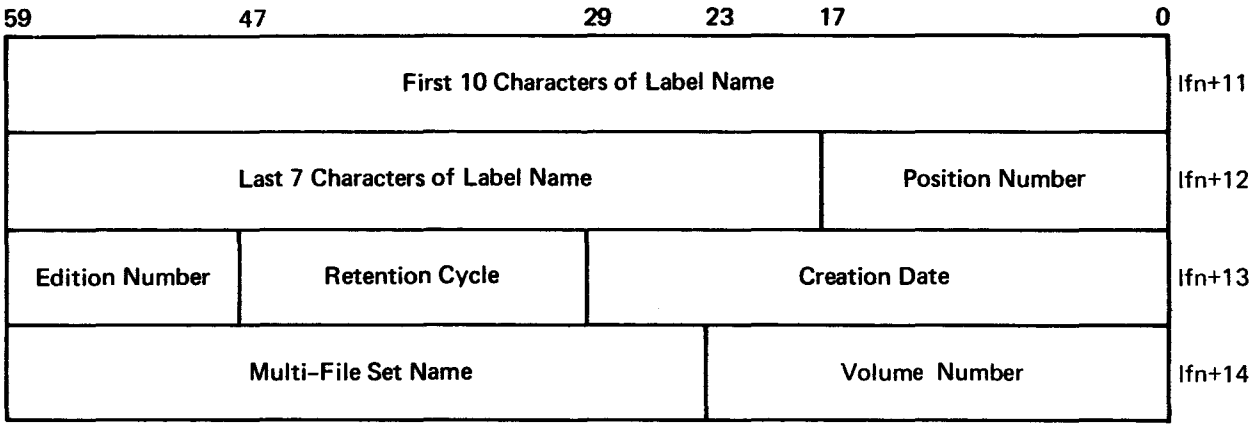
When the bit is off, the system generates output data and checks input data only for required ANSI, Z format, and Y (3000 series) format labels. Labels that are processed by standard processing (excluding Y labels) are label types VOL1, HDR1, EOF1, and EOV1. Default values are written if the user does not specity 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 (lfn + 11 through lfn + 14) 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 of less than 13 words, but any labeled tape will be accepted for processing.

When output tapes are opened, any information in words 10-13 (lfn + 11 through lfn + 14) 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 (lfn + 11 through lfn + 14) 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 — 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 data in format of two digits for year and three digits for day (yyddd); default is current date.

vol — 1-4 decimal digits indicating volume within a multivolume set; default is 0001.

mfn — Multifile 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 multifile 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.

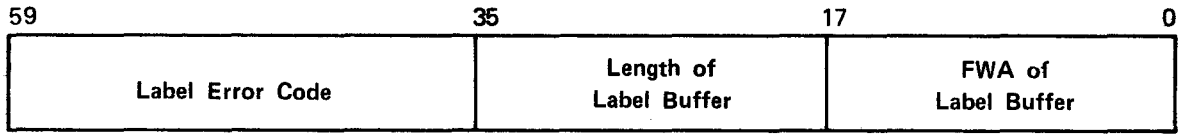
When the information in the FET is used to check existing labels, binary-zero fill characters will be converted to the display code blank appropriate for character fields or display code zero for numeric fields before comparison is made. Fields in the FET containing all binary zeros are not compared. 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 nonmatching label fields. If the mounted tape is accepted, the values returned to the FET will reflect the header label on that tape.

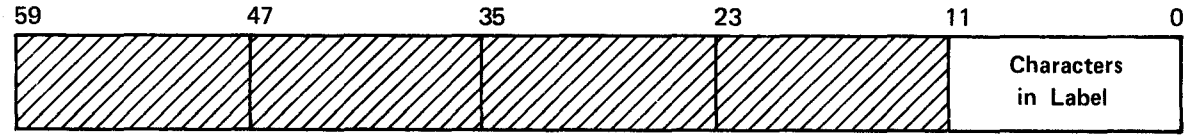
EXTENDED LABEL PROCESSING

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 (lfn + 11) 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 multifile set processing with the XL bit set and calls to the COMPASS macro POSMF, word 10 (lfn + 11) of the FET must point a label buffer. One of the first entries in the buffer must be a formatted HDR1 label with the multifile name in the set identifier field. The position number field in the label has four 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 multifile 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 central program control (CPC) 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 (CPMTR and MTR).

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 nonzero 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 completed. 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 Monitor to suspend the program execution for a while. The length of time that Monitor suspends the program execution depends on whether periodic or automatic recall was requested, as well as other system activity.

Periodic recall puts the program in recall status for the amount of time specified in the request. If no period (that is, zero) is specified in the request, CPMTR supplies a default period of approximately 25 milliseconds. If a program calls a PP program without recall and issues a periodic recall request before that PP program completes, then when the first PP program does complete, the periodic recall is terminated regardless of the time remaining in the specified recall period. In addition, when a program is performing an RMS Input/Output operation without automatic recall, the program's periodic recall period is terminated whenever there is a change in the FET IN or OUT field that the system is updating. For RMS files, the system updates this field after each transfer of 64 words.

Automatic recall (auto recall) causes Monitor to suspend execution of the program until the specified request is completed. The request must contain a status word address greater than RA+1 and less than RA+FL. Recall status is terminated when Monitor detects that the complete bit (bit 0) of the status word is set. For any non-RCL request, the program is aborted with an AUTO-RECALL ERROR message if the complete bit is already set when the request is issued.

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 nonblank 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 setting of bit 40. If bit 40 is zero, then Monitor treats the request as a periodic recall request and uses the value in bits 0-10 as the recall period. This is the only way a program can issue a periodic recall request with a non-default recall period. The period is specified in quarter-milliseconds (4096 quarter-milliseconds = 1 sec.), so the largest recall period that can be specified is 500 milliseconds; the shortest period is approximately 7.5 milliseconds on a CDC CYBER 170 Series machine and approximately 15 milliseconds on other machines. If a shorter period is specified in the request, the actual time suspended varies. If bit 40 is set to one, then it is an automatic recall request with bits 0-17 containing the address of a status word. It is expected that the complete bit in this status word will be set by some previously issued request (without recall) to terminate the recall status. If the complete bit is already set when the request is issued, then the request is ignored and the program continued.
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 nonblank 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.

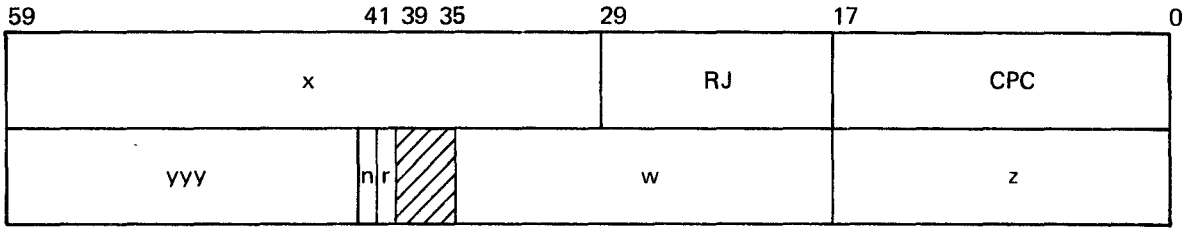
The user communicates with CPC through macro requests and the FET. Communication with the operating system is handled by CPC through setting and checking RA+1. CPC may also cause the execution of one or more user OWNCODE subroutines for which addresses are specified in word 9 (lfn + 10) of the FET.

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 is one of the following.

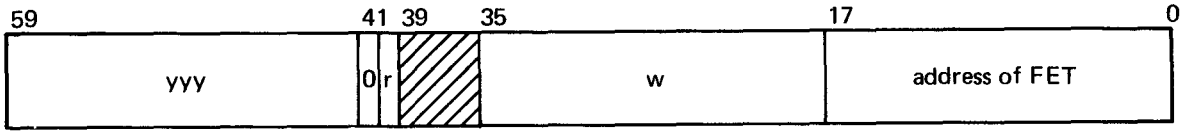
CIO	CPC generates a call to CIO.
000001	Only file recall is desired. Display code characters RCL are generated in RA+1. OWNCODE routines are executed if appropriate.
000002	Used for most read or write functions. A function in progress is not reissued by CPC. When the file becomes inactive, CPC issues a call to CIO.

000003	Used for all other functions. When the file becomes inactive, CPC issues a call to CIO.
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.

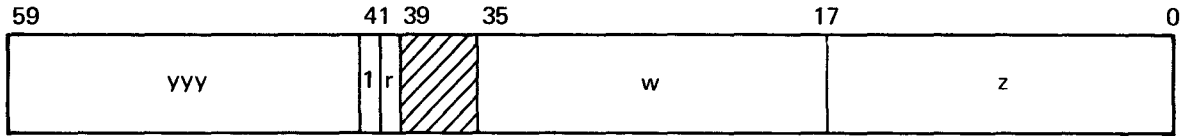
If n=1 indicating other than a file action request:

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 (lfn+0) 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 reissue 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, preceding, 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 at lfn + 0; 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 the CPC waits for a zero quantity to reappear. If the auto-recall bit was set, CPC executes step 2, preceding, as a subroutine. CPC then exits with X1 containing zero if no abnormal conditions (error code in FET equals zero) were encountered; otherwise, X1 contains the value from FET word 1 (lfn + 0).

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 Comment from operator (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 statements 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 (refer to NOS/BE 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 to illustrate 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.

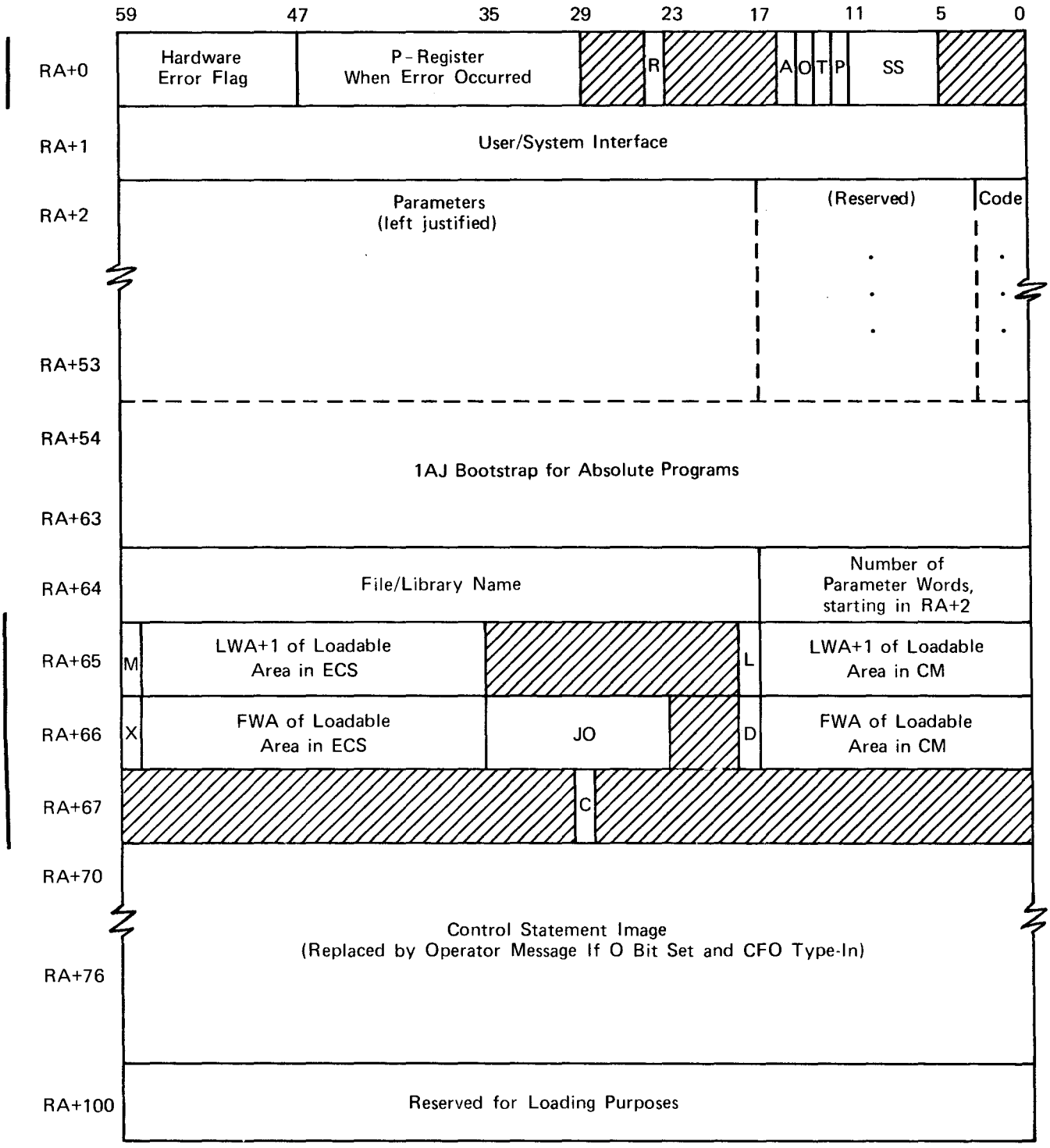


Figure 7-1. Communication Area RA through RA+100

CDC CYBER RECORD MANAGER MACROS

CDC 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.

CDC 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 CDC 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 CDC 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 can 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 CDC 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 manuals along with other product capabilities.

Macro	Function
File Creation and Maintenance Macros:	
FETCH	Retrieves value of any field in FIT
FILE	Creates file information table (FIT)
STORE	Sets value in field of FIT
File Initialization and Termination Macros:	
CLOSEM	Terminates file processing; initiates label processing
FLUSHM	Flushes buffers as if the files had been closed
OPENM	Prepares a file for processing; initiates label processing
Data Transfer Macros:	
CHECK	Determines completion status of I/O operations
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

Macro	Function
PUTP	Transfers a portion of a record to a file
File Positioning Macros:	
REWINDM	Rewinds volume to beginning-of-information
SEEK	Provides overlap between I/O and processing by positioning while processing
SKIP	Repositions file backward or forward
START	Positions a file to a record that satisfies a specific condition
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 CDC 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 manuals for Record Manager contain details of its use. CDC 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 CPUTEXT.

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 symbols listed below are made available for use by the user program.

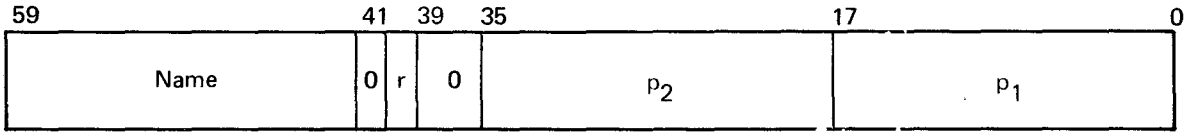
RA.SSW	=	0	Sense switches in bits 11-6.
RA.MTR	=	1	System monitor request register.
RA.ARG	=	2	Start of control statement argument list.
RA.PGN	=	64_8	Bits 59-18 = program name.
RA.ACT	=	64_8	Bits 17-00 = argument count.
RA.LWP	=	65_8	Last word pointers for overlay load.
RA.CMU	=	65_8	Compare move unit flag (bit 59).
RA.FWP	=	66_8	First word pointers for overlay load.
RA.CEJ	=	66_8	Bit 59 = central exchange jump enable flag.
RA.LDR	=	67_8	Loader communication word.
RA.CCD	=	70_8	First word of control card image.
RA.ORG	=	100_8	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, X2, X6, A1, and A6 are destroyed during macro execution and should not be used as parameters.

SYSTEM name,recall,p_1,p_2

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
p_1	First parameter to PP program
p_2	Second parameter to PP program

The value of p_1 or p_2 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,,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 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

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, CPUTEXT, 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 macro 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,p_1,p_2

lfn — Position allows for SCOPE 2 compatibility. Any nonblank value in this field causes an assembly error under NOS/BE.

p_1 — 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 nonblank value in this field is ignored.

p_2 — 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 nonblank value in this field is ignored.

The DMPX user dump produced when p_1 is blank (or any nonblank 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 macro is usally 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/1000_8.
Bits 35-24	Number of PPs.
Bits 23-20	Reserved for hardware characteristics.
Bits 19-18	CYBER 170 Model 176 mainframe flag: 0 Not a 176 1-2 Reserved 3 Model 176 mainframe
Bit 17	PPs 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/200_8.
Bit 0	Completion bit; must be set to 0 before execution. It is automatically set to 1 by the function processor when execution of the macro has finished.

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, a PP-call 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 nonblank parameter is acceptable. Recall is required on all requests for memory increases.

length — Optional parameter giving number of words of field length requested. For ECS the maximum field length is 377_8 K.

nabort — Optional parameter which averts a job abort if nonblank 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 nonblank 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 (7777777777_8) 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 (7777777776_8) 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 nonblank	Display on B display but do not record elsewhere.
recall	Optional recall parameter; if nonblank, 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 57_8 or 0 is replaced with a blank (display code value of 55_8). 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. The RECALL macro calls the system communication subroutine RCL=, or CPC if CPCTEXT is used.

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 addr is zero and CPC is not used, control returns immediately 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=.

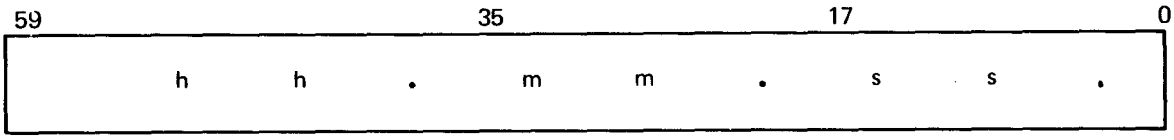
CLOCK	Current system clock in hours, minutes, and seconds
DATE	Current date established at deadstart time when the system was loaded
JDATE	Current date in format yyddd for year and date
RTIME	Real time clock maintained by Monitor, in fractional seconds
TIME	Central processor time allowed and used by 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.

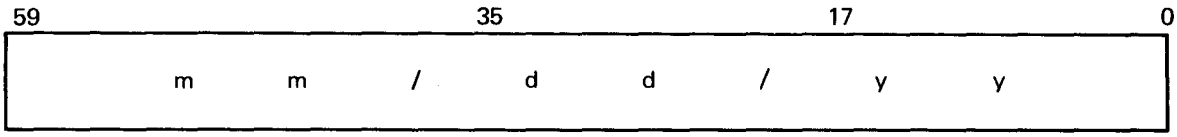
The system clock is that established when the operator loads the system. Display code hours, minutes, and seconds appear with periods and a leading blank as follows:

CLOCK status



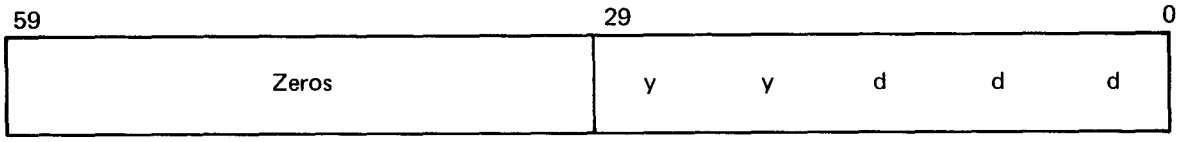
The date returned is that typed by the operator when the system was loaded. Its format is display code, and generally is mm/dd/yy for month, day, and year; this order may be changed at installation option. A leading and trailing blank appear.

DATE status



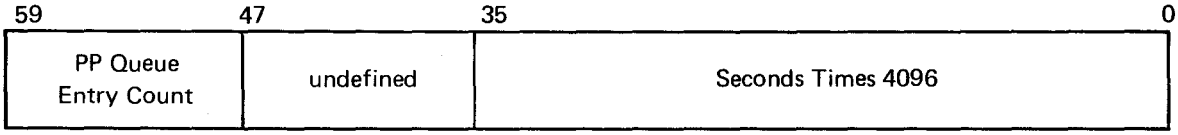
Date in a format suitable for calculating elapsed days is returned with JDATE. Five display code characters appear in the low order position; the first two digits are the year, the last three the number of the day in the year.

JDATE status



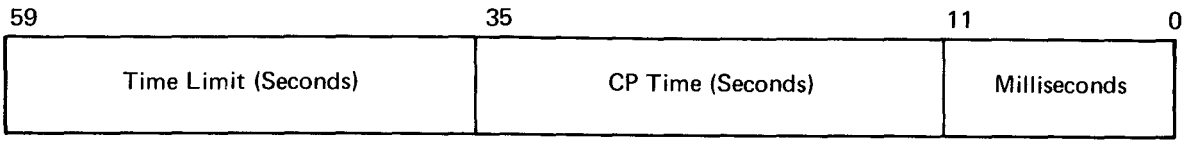
The real time clock is that maintained by Monitor for purposes such as determining peripheral processor time used. The status word will show seconds in bits 12-35 and units of 4096ths of a second (244 9/64 microseconds) in bits 0-11.

RTIME status



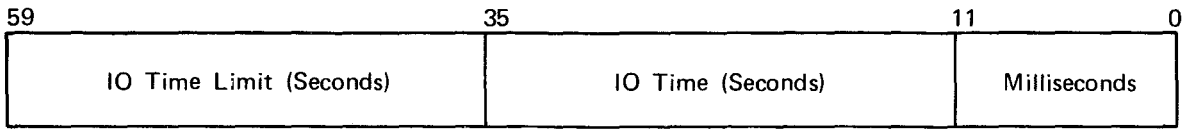
The job time limit is that requested on the job statement or assigned by installation default. Central processor time used is shown in seconds and milliseconds.

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

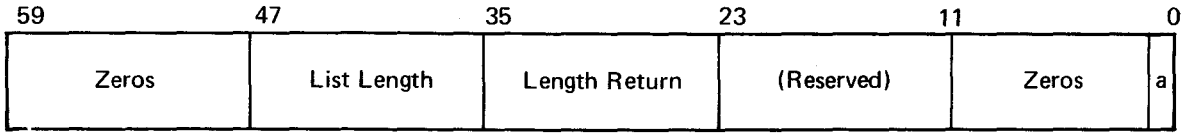
The STATUS function provides a user program with information about system resources. Two types of information are available depending on the value of the x parameter as described below.

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 returns control point activity information to the header word. x = 5-777 reserved; 1000 to 7777 reserved for installation use.
recall	Optional recall parameter; any nonblank character.

Format of the header word for x = 1, 2, or 3 must be:



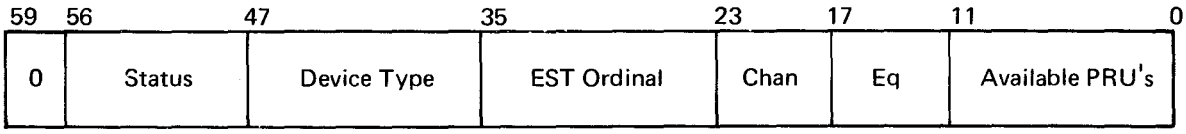
List Length — Number of words, excluding this header word, to be used for return information; must be set by user to other than 0.

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
AJ	885 disk drive
AX	ECS resident files
AY	844-21 disk drive
AZ	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.

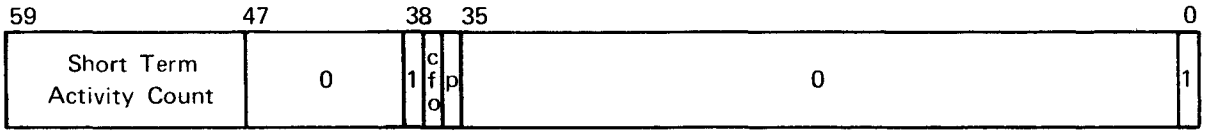
Eq — Equipment (controller) number to which device is connected.

PRUs — Number of PRUs, divided by 100 octal, of space remaining on the device; value of 7777 indicates at least 262,100 PRUs available. This value is not returned for ECS resident files.

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.

When x=4, bit 0 (complete flag) in the header word must be zero initially, and all other bits are ignored. The PP program STS processes the request. Upon completion, the following information is returned to the header word.



Short Term Activity Count	Sum of active PPUs (excluding the STS program), stack results, and subsystem control point wait responses.
Bit 38	Bit 38 is set equal to 1 to indicate a subsystem control point long-term connect.
cfo	Comment-from-operator flag; set equal to bit 14 of RA+0.
p	Pause flag; set equal to bit 12 of RA+0.

Although this STS request can be used by any program, it is required only by system programs involved with interactive debugging.

FILESTAT MACRO

The FILESTAT macro is an alternate for the STATUS macro:

```
FILESTAT list,recall
```

This macro is equivalent to:

```
STATUS  list,2,recall
```

GETACT MACRO

The GETACT macro is an alternate for the STATUS macro:

```
GETACT  list,recall
```

This macro is equivalent to:

```
STATUS  list,4,recall
```

FILINFO MACRO

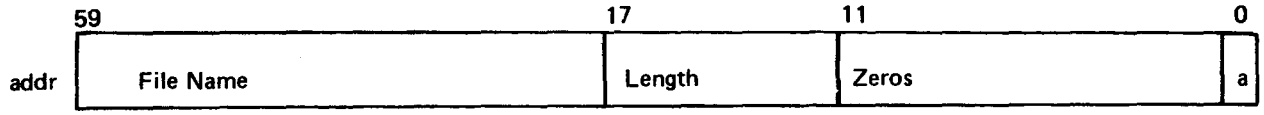
The FILINFO macro provides a user program with information about a file assigned to the user's control point. The general information, common to most files, is returned to a table (return block) whose standard size is five words. However, the size of this table is variable and additional information for tape files, if requested, is returned in extra key words in the return block after the standard five words.

The call to this macro is:

FILINFO addr

addr — Address of a table (FILINFO return block) to receive file information.

Format of the header word must be:

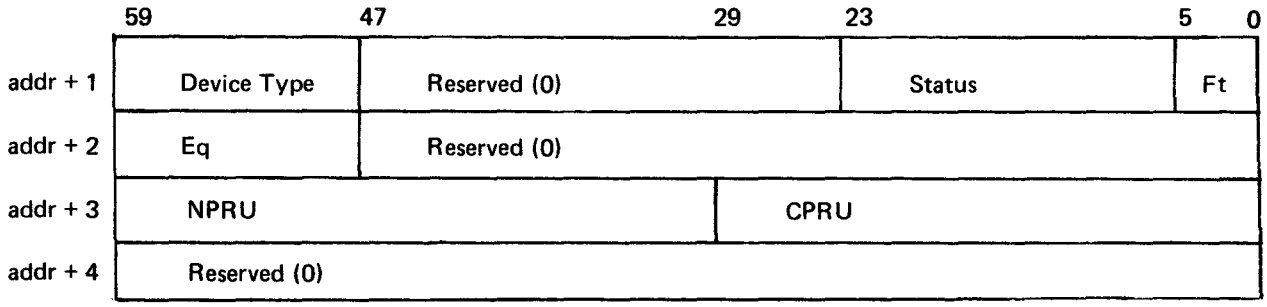


File Name — A valid display-coded file name.

Length — Table length including the first word. Must be at least 4.

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 standard table will have the following format. If the file is not assigned to the user's job, the table entries will be zero.



Device Type — Hardware mnemonic in display code:

AH	819 disk drive
AJ	885 disk drive
AX	ECS resident files
AY	7054/844-21 disk drive
AZ	7054/844-41 disk drive
CP	415 card punch
CR	405 card reader
DS	Console display
FM	254-2 microfilm recorder
GC	252-2 graphics console
HC	253-2 hardcopy recorder
KB	Remote terminal keyboard
LM	Link medium file
LP	Line printer (any)
LR	580-12 line printer
LS	580-16 line printer
LT	580-20 line printer
MT	667 or 677 magnetic tape drive
NT	669 or 679 magnetic tape drive
PL	Plotter
TP	3691 paper tape punch
TR	3691 paper tape reader

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. This bit is set when the device type field is zero (the file has not yet been assigned to a device with a REQUEST or the file has not yet been accessed with a GETPF).

Bit 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 mass storage file):

Bits 59-36 PRU count
Bits 35-30 0

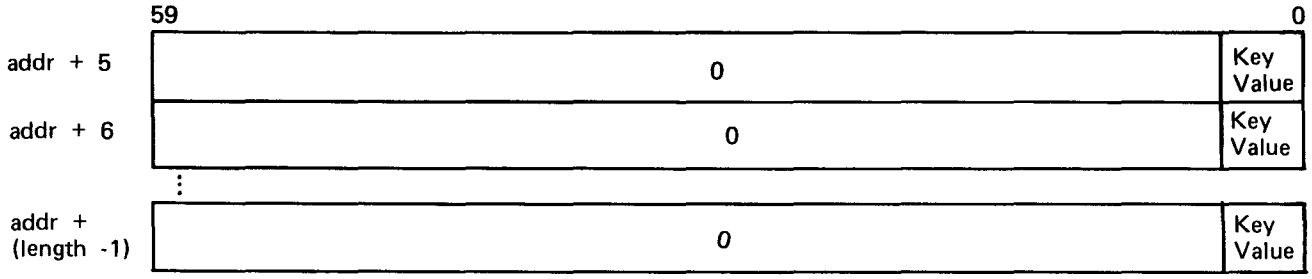
CPRU Current file position (if mass storage file) given as the number of PRUs+1 from beginning-of-information (beginning-of-information as indicated by PRU count=1):

To request additional tape file information, the user must set up additional key words, each with a key value specified in bits 4-0 which corresponds to information desired. The key words begin at addr+5 in the FILINFO return block and can be defined in any order. The table length contained in bits 17 through 12 of the header word must include the additional key words. The key values and corresponding information are as follows:

Key Value	Information
0	Ignored
1	Tape data format
2	Error processing flag, label type, density, and conversion mode
$3\text{-}27_8$	Reserved for CDC
$30_8\text{-}37_8$	Reserved for installations

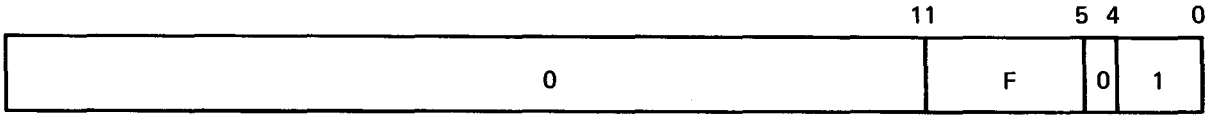
†Read, extend, and modify reflect permanent file permissions for mass storage files. Write permission is set if either modify or extend permission is set. Modify, extend, and write permissions are set for magnetic tape files if the write-enable ring is present and cleared if the ring is absent. Read permission is set unless the file is a multi-file tape.

Format of the additional key words must be:



When the operation is completed, the FILINFO return block includes the additional key words which contain the additional information requested. Any key word with a key value of zero is ignored. Bit 5 in a key word is set if the specified key value is unknown to the operating system, not applicable to the file type, or if the file is not found.

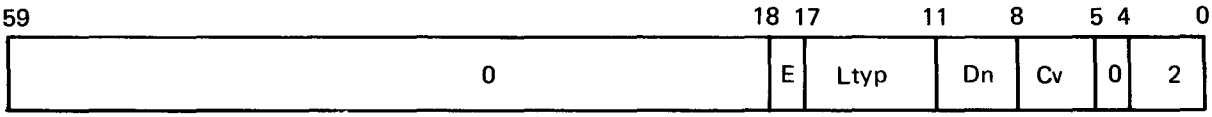
If the key value equals one, the returned key word will have the following format.



F Bits 11-6 **Tape data format:**

00	Reserved for CDC
01	System internal (SI)
02	Reserved for CDC
03	Stranger (S)
04	Long block stranger (L)
$05\text{-}77_8$	Reserved for CDC

If the key value equals two, the returned key word will have the following format.



E	Bit 18	Error processing inhibited (if set)
Ltyp	Bits 17-12	**Label type:**

00	Unlabeled
01	Standard (ANSI 1969 Std.)
$02\text{-}10_8$	Reserved for CDC
11_8	Y label (3000 label)
12_8	Z label
13_8	Nonstandard (no tape positioning)
$14_8\text{-}67_8$	Reserved for CDC
$70_8\text{-}77_8$	Reserved for installations

Dn	Bits 11-9	**Tape density:**

0	Reserved for CDC
1	200 bpi (7-track)
2	556 bpi (7-track)
3	800 bpi/cpi (7- or 9-track)
4	1600 cpi (9-track)
5	6250 cpi (9-track)

Cv	Bits 8-6	**Conversion mode:†**

0	External BCD conversion
1	ASCII conversion
2	EBCDIC conversion
3-7	Reserved for CDC

GETJCI MACRO

The GETJCI macro allows a user program to transfer the job control information used by 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, modifying the appropriate fields, and then executing the SETJCI macro to save the new fields in the system area.

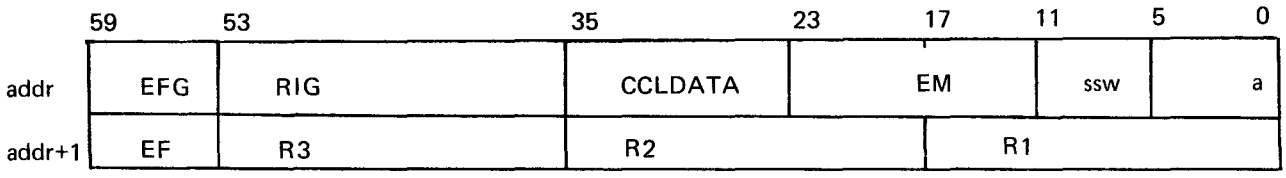
The call to this macro is:

```
GETJCI    addr
```

addr Address of a 2-word table.

†Conversion mode of labels (if any) or of coded data (if any). This does not indicate whether the data on the tape is coded or binary.

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	Completion flag; must be set to 0 before execution. 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.

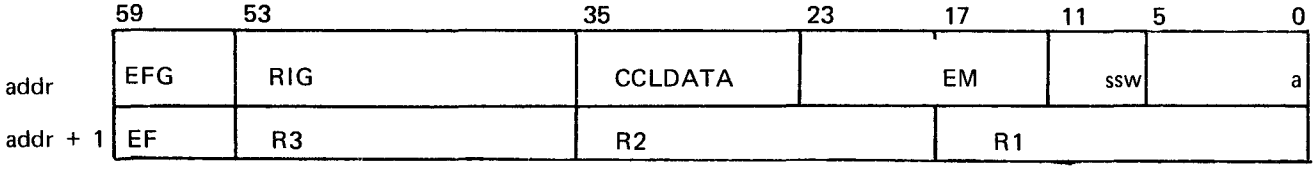
The call to this macro is:

SETJCI addr

addr Address of a 2-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	Completion flag; must be set to 0 before execution. 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 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 STATEMENTS

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. A job can be recovered from only one operator KILL.

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 5 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.

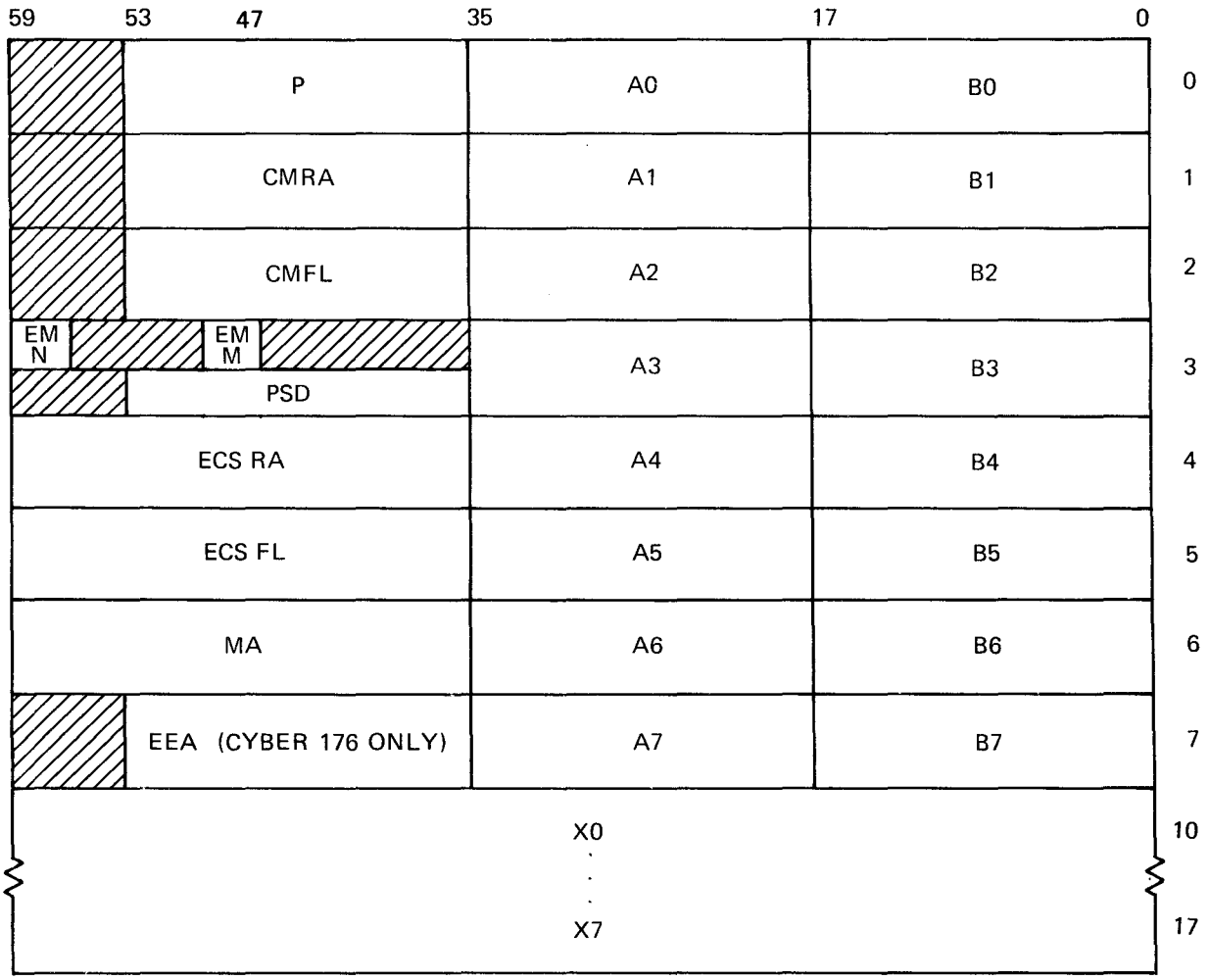
checksum — Last word address of recovery code to be checksummed; 0 if no checksum.

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.

A 17-word (decimal) array showing the program situation when RPV was called. The first 16 words are an image of the exchange package with the error condition in the B0 field (refer to figure 7-2). The seventeenth word is the contents of RA+1.

A flag that determines the type of program termination. If the user sets the flag nonzero, ENDRUN termination occurs upon completion of the last postprocessing 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.

An array, starting at RA+1, that allows a FORTRAN subroutine to access all of the user's field length.



Field Name	Description
EM	Exit mode bits (all models except CYBER 176).
N	Hardware exit mode flag for the CYBER 170 only.
M	Hardware exit mode flag for all models.
PSD	Program status designator for the CYBER 176 only.

Figure 7-2. Format of the Exchange Package Image

he subroutines called by RECOVR should return; if they do not, additional subroutine calls, if any, and he 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 error conditions and associated error codes and masks are:

Condition	Error Code (octal)	RECOVR Mask (octal)
Normal termination	0	100
CP time limit	1	004
Mode error	2	001
PP program requested abort	3	020
CP program requested abort	4	040
PP call error	5	002
Operator DROP	6	010
Operator KILL	7	010
Operator RERUN	10	010
Control statement error	11	040
ECS parity error	12	020
Auto-recall error	15	002
Hung in auto-recall	16	002
Mass storage limit	17	004
PP program not in library	20	002
I/O time limit	21	004

The FORTRAN language contains RECOVR subroutines as detailed in that reference manual.

CALLING RPV DIRECTLY

RPV should be called directly only by programs with complete control over all RPV requests that the routine issues. The RECOVR utility should be used in all other situations, because system-supplied routines such as a CDC CYBER Record Manager routine define (through RECOVR) their own reprieve routines.

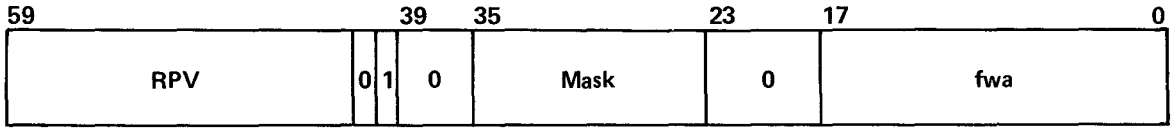
Two modes of reprieve processing are available with RPV, normal and extended. Normal mode provides the reprieve processing capabilities as described for the RECOVR macro. Extended mode provides all the capabilities of normal mode and the means of disabling external interrupts while the reprieve routine is active and resuming the interrupted program after the error has been processed.

NORMAL RPV

SETUP and RESET calls can be made with normal mode reprieve processing. Before an error is encountered, a SETUP call establishes the address of the reprieve routine and the classes of errors which cause control to be transferred to the reprieve routine. Following the processing of an error, a RESET call issued by the reprieve routine reinstates the error condition and allows system processing of the error as if the job had not been reprieved.

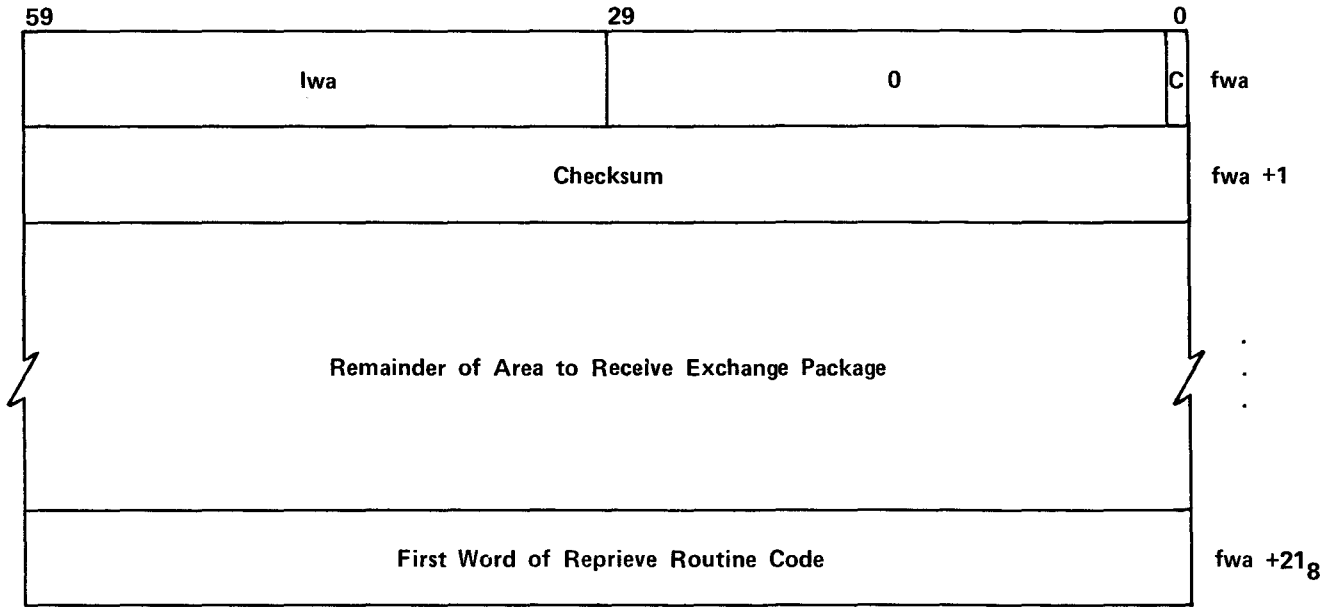
SETUP Call

The RA+1 RPV SETUP call has the following form.



Mask — Mask specifying the classes of errors for which the system initiates reprieve processing.

fwa — First word address of a parameter list formatted as follows:

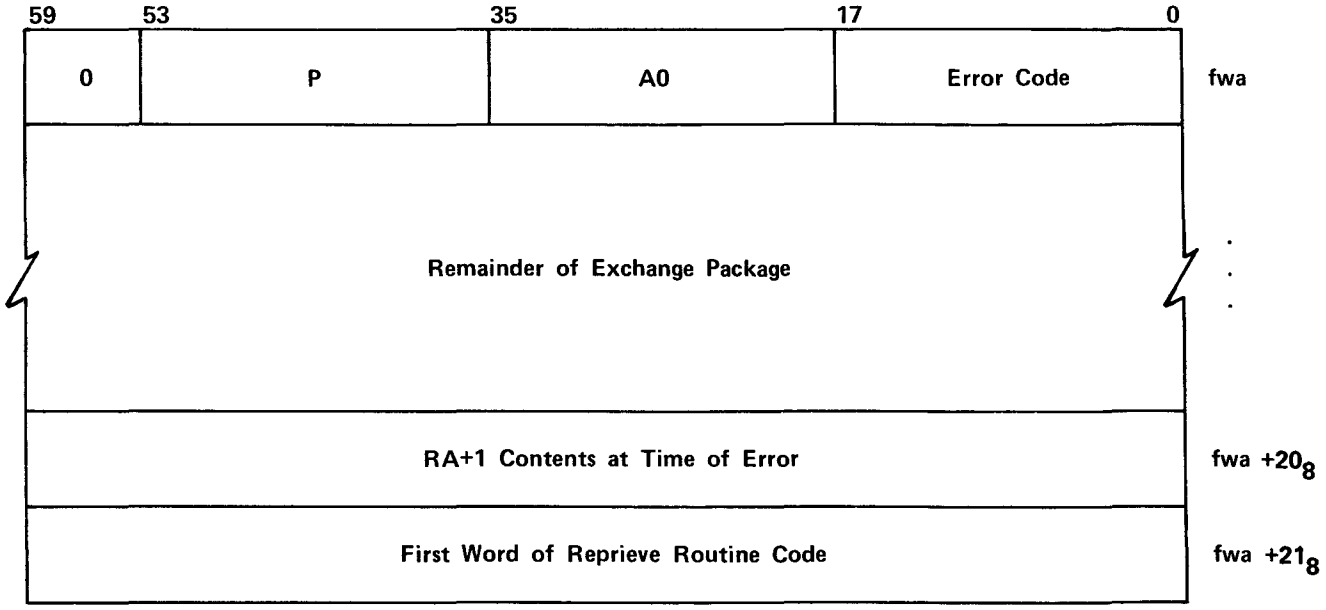


lwa — Last word address of user routine to checksum (0 if no checksum).

C — Completion bit; 0 before the call; 1 upon completion.

Checksum — Set by RPV. Checksum of words fwa+21_8 (first word of user routine) through lwa.

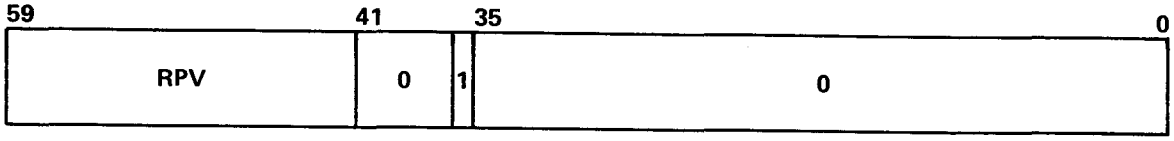
Upon detection of an error condition, the system initiates RPV to reprieve the job. If the error condition is in one of the error classes specified by the mask, RPV transfers control to fwa+21_8. RPV provides information about the error and the state of the interrupted program in the parameter block as follows:



P	Contents of the program register.
A0	Contents of the A0 register.
Error Code	Code identifying the error condition which initiated reprieve processing.

RESET Call

The RA+1 RPV RESET call has the following form.

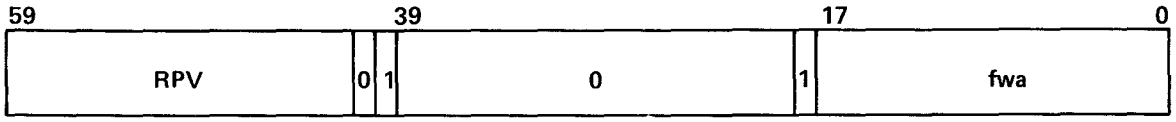


RPV resets the error and exchange package from the fields, starting at fwa specified in the SETUP call.

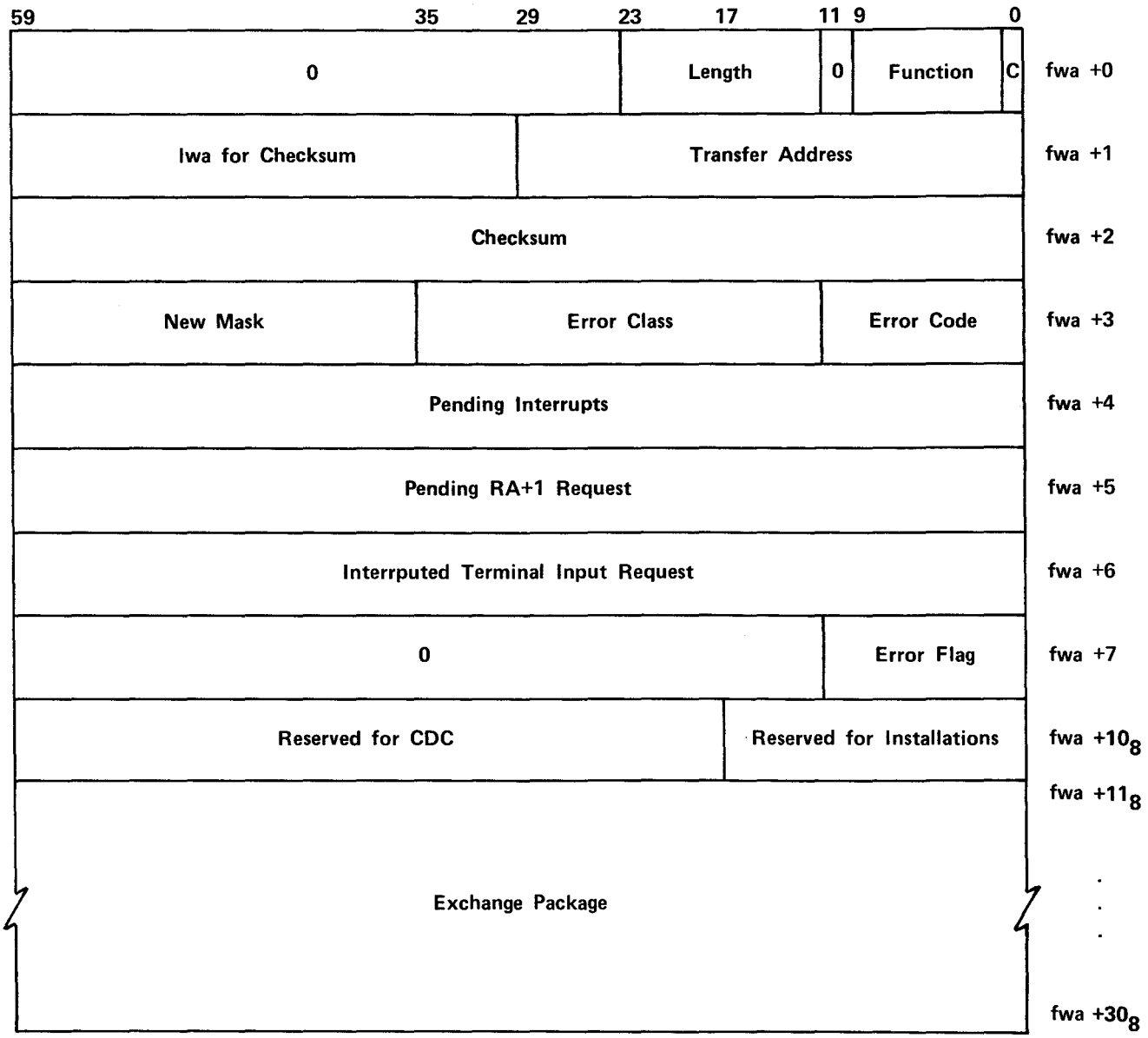
EXTENDED RPV

SETUP, RESET, and RESUME calls can be made with extended mode reprieve processing. The SETUP and RESET calls perform functions similar to the SETUP and RESET calls for normal mode RPV. The RESUME call restores the exchange package and causes execution of the program to resume at the point at which the error was detected.

The RA+1 RPV call has the following form.



fwa First word address of a parameter list formatted as follows:



Length	Length of parameter block. Set by user.
Function	Function code. Set by user. One of the following. 1 SETUP call 2 RESUME call 3 RESET call
C	Completion bit. Cleared by user; set to 1 by RPV upon completion of function.
lwa for Checksum	Last word address of user routine to checksum (0 if no checksum). Set by user.
Transfer Address	Address to which control is transferred when an interruption occurs (first word of user routine). Set by user.
Checksum	Checksum from transfer address through lwa of area to checksum (first through last word of user routine). Set by RPV.
New Mask	Mask specifying the error class for which reprieve is desired. Set by user.
Error Class	Mask bit which specified the class of the reprieved error. Set by RPV.
Error Code	Code of reprieved error. Set by RPV.
Pending Interrupts	Bit i set indicates system error flag i is pending. Set by RPV.
Pending RA+1 Request	RA+1 contents at the time of interruption. Set by RPV.
Interrupted Terminal Input Request	If a terminal input request with auto-recall was in progress at the time of the interrupt, this field contains the reconstructed CIO RA+1 request (0 if no interrupted terminal input). Set by RPV.
Error Flag	Value of the system error flag at the time of interrupt. Set by RPV.
Exchange Package	Copy of the exchange package at the time of interrupt. Set by RPV.

The error conditions and associated error codes and masks are as follows:

Condition	Error Code (octal)	Mask (octal)
Normal termination	0	0100
CP time limit	1	0004
Mode error	2	0001
PP program requested abort	3	0020
CP program requested abort	4	0040
PP call error	5	0002
Operator DROP	6	0010
Operator KILL	7	0010
Operator RERUN	10	0010
Control statement error	11	0040
ECS parity error	12	0020
Auto-recall error	15	0002
Hung in auto-recall	16	0002
Mass storage limit	17	0004
PP program not in library	20	0002
I/O time limit	21	0004
Reserved for Control Data	22-33	0400-1000
Reserved for installations	34-37	2000-4000
Terminal interrupt	40	0200

In a SETUP call, the user sets the new mask, the transfer address, and the lwa for checksum, and the user clears the pending interrupts, pending RA+1 request, and the interrupted terminal input request. If a SETUP call has the pending interruptions, pending RA+1 request, or interrupted terminal input request set, these fields are processed in the same way that a RESUME call processes them (see the description of RESUME).

Upon detection of an error, the system initiates RPV to reprieve the job. If the error condition is in one of the error classes specified by the mask, three possibilities exist.

If the reprieve routine is active and did not generate the error (error codes 1, 6, 7, 10, 12, 17, 21, or 40), the job is reprieved. RPV marks the error in the pending interrupts word, and the reprieve routine continues.

If the reprieve routine is active and did generate the error, the job is not reprieved.

If the error condition is not in one of the classes specified by the mask, the job is not reprieved. Also, the second occurrence of KILL, CP time limit, mass storage limit, or I/O time limit is not reprieved.

A RESUME call directs RPV to restore the P, A, B, and X registers of the exchange package, pending RA+1 request, and interrupted terminal input request and to transfer control back to the point in the program where the error was detected. However, if the pending interrupts word is non-zero, the reprieve routine is reinitiated to process the highest priority of the pending interrupts. The priority ordering of the interrupts is as follows:

ERROR FLAG VALUE	ERROR CONDITION
7 (Highest)	Operator KILL
10	Operator RERUN
6	Operator DROP
12	ECS parity error
17	Mass storage limit
21	I/O time limit
1	CP time limit
11	Control statement error
20	PP program not in library
15	Auto-recall error
16	Hung in auto-recall
5	PP call error
3	PP program requested abort
4	CP program requested abort
2	Mode error
0	Normal termination
40 (Lowest)	Terminal interrupt

When RESUME is used following the processing of an error, the error condition should be corrected. For example, after a reprieve for a CP abort, the pending RA+1 request should be cleared before RESUME is called so that the RA+1 ABT request is not reissued.

Caution is advised when using the RESUME call or depending on the disabling of interrupts. An interrupt can often cause premature termination of activities related to the execution of the CP program.

Therefore, if a program requires a RESUME call after interruption and disabling of interrupts, it must avoid using any of the following.

Magnetic tapes
Device set operations (PP programs: ADS, DUM, MNT, and DSM)
Permanent file operations (ALTER, ATTACH, CATALOG, EXTEND, PURGE, and RENAME macros)
Explicit file requests (REQUEST macro, and REQ PP program)
DIS PP program
Checkpoint/restart
Job dependency processing (TRANSPF macro, and JDP PP program)
Multi-mainframe file functions on files not yet transmitted
Multiply-connected files
PP programs: LBL, LDC, LDV, LDW, MDI, NSV, and XDQ

PP programs MNT and DSM can be executed implicitly as the result of other I/O operations. Before execution, the user should explicitly request possible mounts of all member devices used to avoid implicit mounts. After execution, the user may be required to explicitly request dismounts for dismounts that did not complete.

Proper resumption of the program after an interruption and disabling of interrupts for activities related to the execution of the CP program are not guaranteed for an operator KILL.

Normal mass storage operations always complete. The interrupted terminal input FET is left incomplete, and the FET address is available in the interrupted terminal input request word.

Caution is advised when designing reprieve routines that issue RESUME calls to avoid alteration of the interrupted program, including the subroutine return address.

REPRIEVE MACRO

The REPRIEVE macro can call RPV directly. It issues an RA+1 request for extended reprieve processing. Programs that cannot call RPV directly should use the RECOVR macro. The request takes the following form.

REPRIEVE addr,type,mask

addr	Address of the extended RPV parameter block.
type	Type of call (SETUP, RESET, or RESUME).
mask	Reprieve mask specifying the classes of errors for which the system initiates reprieve processing.

The REPRIEVE macro inserts the parameter values into the corresponding fields in the parameter block. The values in the other fields of the parameter block are the responsibility of the user. Use of the macro destroys the contents of registers A1, A6, X1, and X6.

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:

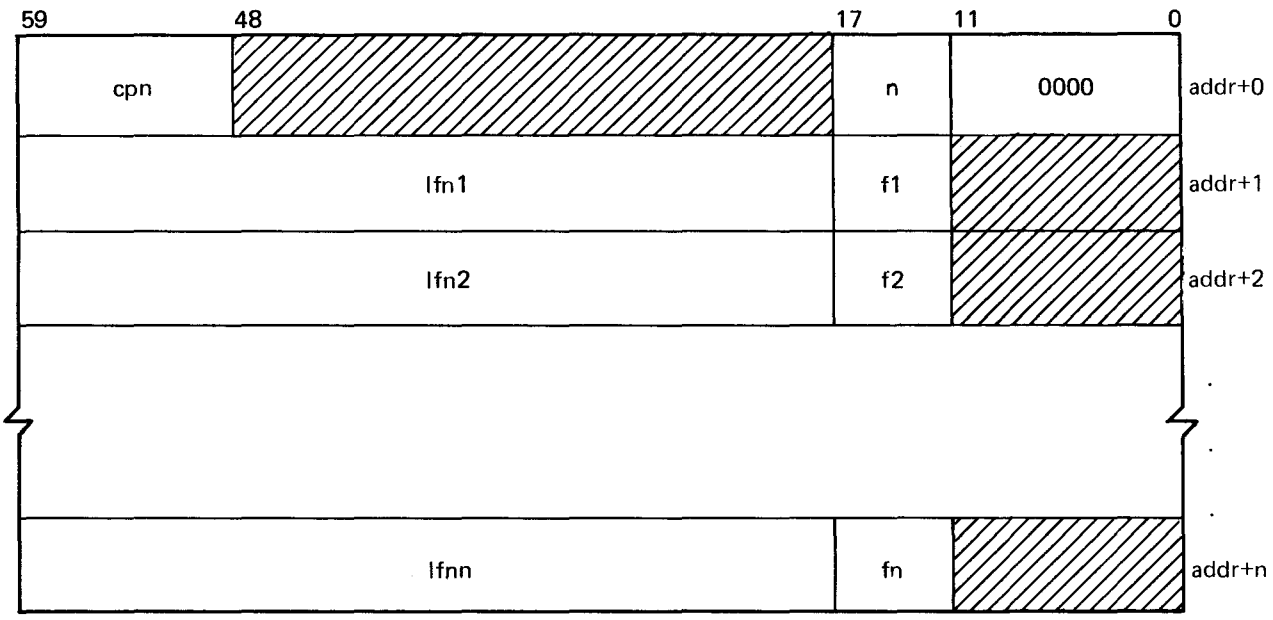
CHECKPT param,sp

sp Mass storage files to be processed.

0 All files.

Nonzero 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.

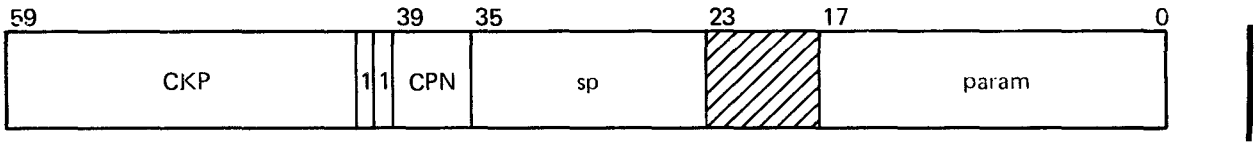
The following standard files, if they exist, are always copied to the checkpoint dump tape.

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 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 file name (lfn) of the file. If the file is a member of a multifile 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.

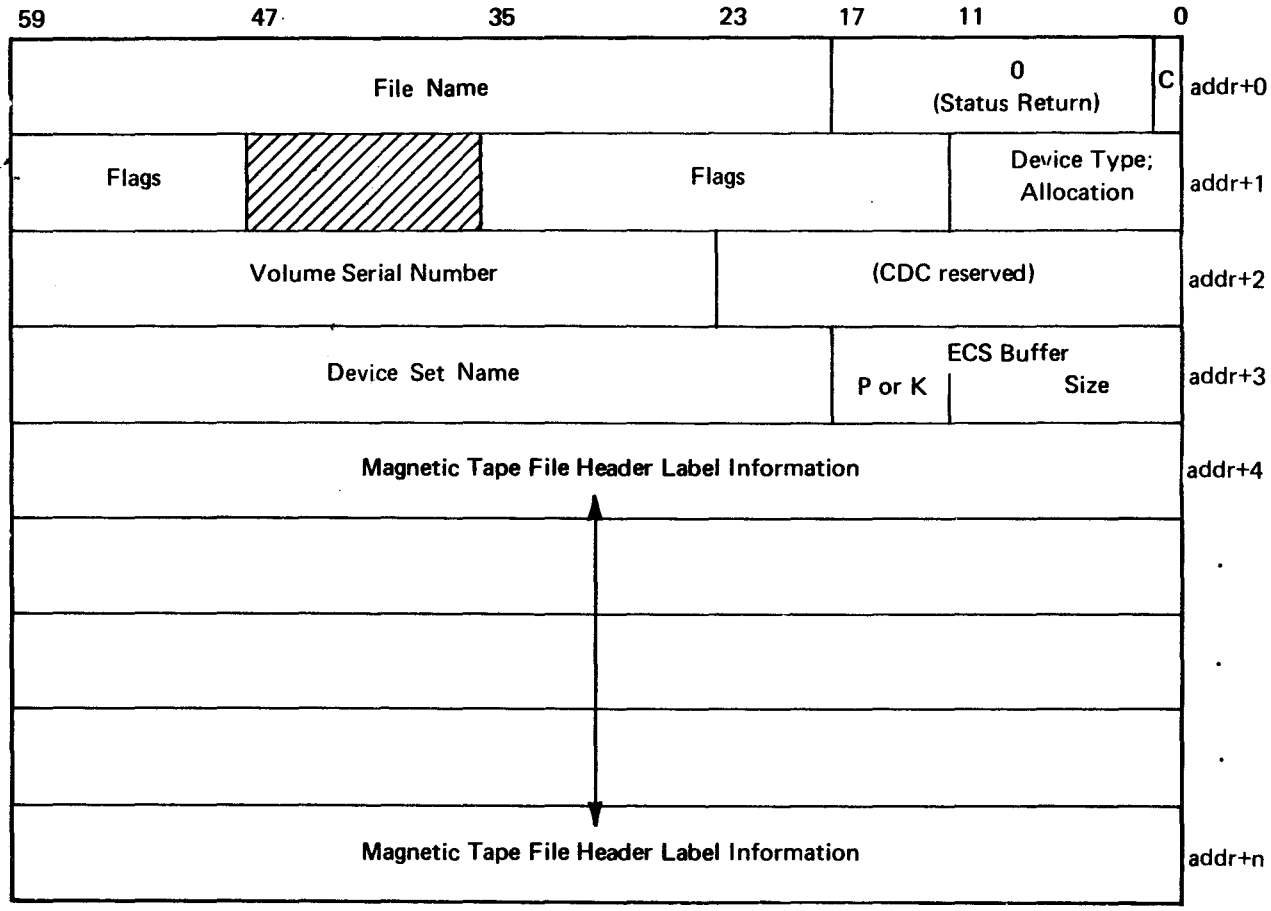
```
REQUEST addr
```

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 (addr+1), in particular, can be changed by the system during processing.

Once the REQUEST function is completed, bit zero of the first word (addr+0) of the parameter list is set to 1. In addition, bits 9-13 of word 1 (addr+0) may show one of the octal codes listed under Status Return after the following Parameter List Format. If so, the REQUEST function has been ignored and control returned to the program.

Parameter List Format:



File Name — File name, left-justified, zero-filled.

Status Return — Initially, user should set to zero. The system returns the following codes.

22 — If INTERCOM or graphics job, a non-allocatable device is specified or default density bit (bit 15) is set.

If batch job, illegal device type is specified. (Refer to table 7-1 following this section for legal device types.)

User issued an RA+1 call for the REQ PP without setting the RECALL bit.

24 — File name table is full.

26 — Device of the requested 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.

C — Completion bit; set to 1 upon completion of REQUEST function.

Flag Fields — Each bit is a flag for a particular condition listed following the explanations of the parameter list fields.

Device Type and Allocation

Bits 6-11 are the device type octal values listed in the device type table at the end of this section (table 7-1). Allocation styles of that device (except tape units) are installation defined. The allocation styles (bits 5 through 0) for tape units are the following:

xx	Seven-Track	Nine-Track
xxxx00	HI density 556 cpi	Reserved for CDC
xxxx01	LO density 200 cpi	GE density 6250 cpi
xxxx10	HY density 800 cpi	HD density 800 cpi
xxxx11	Reserved for CDC	PE density 1600 cpi
xx00xx	Unlabeled	Unlabeled
xx01xx	SI standard U and Z labels	SI standard U and Z labels
xx10xx	3000 Series label (Y)	3000 Series label (Y)
xx11xx	Reserved for CDC	Reserved for CDC
00xxxx	SI data format	SI data format
01xxxx	Reserved for CDC	Reserved for CDC
10xxxx	S data format	S data format
11xxxx	L data format	L data format

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, zero-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.

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. If either bit 48 or bit 49 is set and no VSN is specified, depending on the selection of installation options for automatic tape assigning capabilities, the magnetic tape identified by the file label name is used for automatic assignment. Edition number, creation date, and volume number need not be specified; if they are specified, they must match the fields as read from the label of the candidate tape. The file label name or the VSN must be present to allow automatic assignment.

The flags are individual bits that should be set to 1 to indicate the following conditions; otherwise the bits should be 0.

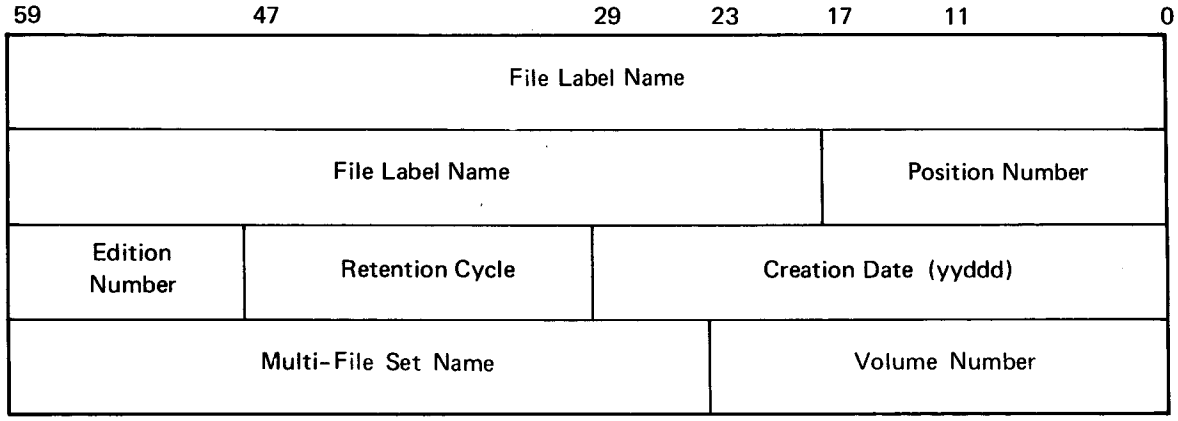
Bit	REQUEST Control Statement Equivalent	Meaning
57	IEC	Inhibit hardware GCR write error correction.
55	*Q	1 = Assign file to queue device. Implies RMS device and causes automatic assignment. Not allowed for private device set.
53	NORING	Write enable ring prohibited in tape.
52	RING	Write enable ring required in tape.
51	MN	Seven-track or nine-track tape can be assigned.
50	A*	Assign any RMS device.

Bit	REQUEST Control Statement Equivalent	Meaning
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 (addr+3). (Private device set files cannot be ECS buffered.)
32	OV	Overflow allowed to different device if that specified in word 2 (addr+1) 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	Nine-track tape conversion to ASCII codes.
29	EB	Nine-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.
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	Nonstandard 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. FNT address is returned to word 2, bits 48-59.
19	— —	Reserved.
18	MF	Request is for a multifile set.

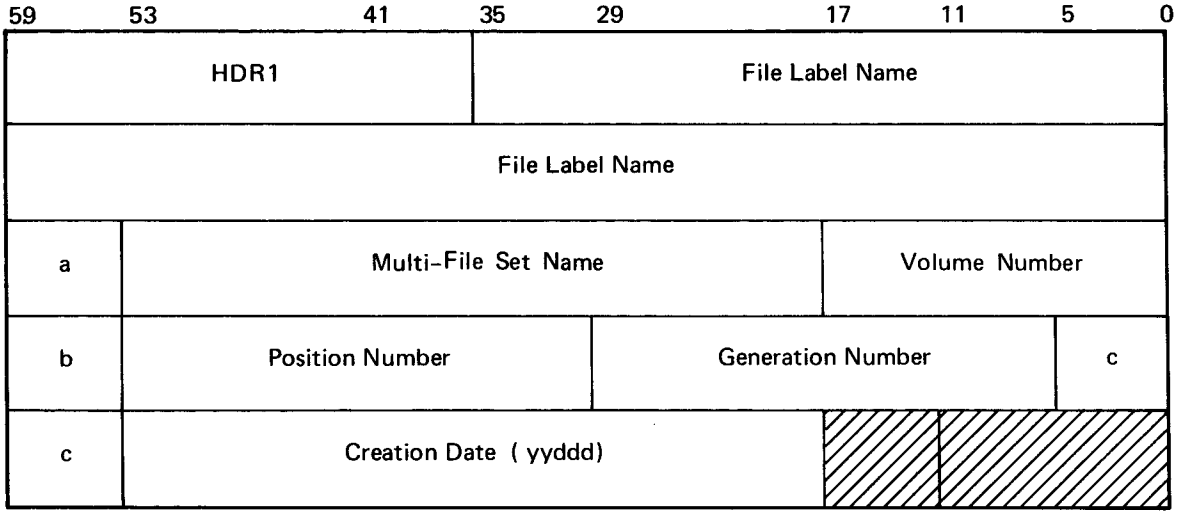
17	SN	Set name for a device set. Must be set to 1 if device set name is specified in word 4.
16	– –	Reserved.
15	absence of explicit density	Magnetic tape is to be written at system default density.
14	SV	Output tape to be saved.
13	IU	Inhibit physical unload of tape.
12	CK	Checkpoint tape request.

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.

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 assignment. 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 seven-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 command.

The following table lists the device types recognized as legal by REQUEST processing. Only device type mnemonics with the first letter A can be specified from an interactive terminal.

TABLE 7-1. REQUEST LEGAL DEVICE TYPES

Mnemonic	Octal value	Device
AH	15	819 Disk.
AJ	17	885 Disk.
AX	20	ECS Resident.
AY	13	844-21 Single Density.
AZ	14	844-41 Double Density.
CP	70	Card Punch.
CR	60	Card Reader.
FM	74	Microfilm.
GC	72	Graphic Display.
HC	73	Hard Copy.
LR	53	580-12 Line Printer.
LS	54	580-16 Line Printer.
LT	55	580-20 Line Printer.
MT	40	Seven-Track Magnetic Tape.
MT	62	Seven-Track Multifile Set Tape.
NT	41	Nine-Track Magnetic Tape.
NT	63	Nine-Track Multifile Set Tape.
TP	45	Paper Tape Punch.
TR	44	Paper Tape Reader.

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 is the function to be performed.

Parameter	Function (With Octal Code)
READNR	Read, no rewind (100).
READ	Read and rewind (140).
REELNR	Read reel, no rewind (300).
REEL	Read reel and rewind (340).
ALTERNR	Alter, no rewind (120).
absent or ALTER	Alter and rewind (160).
WRITENR	Write, no rewind (104).
WRITE	Write and rewind (144).
NR	No rewind (120).

READ, REEL, ALTER, and absent all perform identical functions and can be used interchangeably, as can READNR, REELNR, ALTERNR, and NR.

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 does not reposition 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.

POSMF MACRO

The POSMF function positions standard labeled multifile sets. The multifile set to be positioned is specified by the multifile name in the name field of the FET. The named multifile set is positioned to a particular file and an OPEN with rewind function is performed. The position number is specified in the label field or label buffer. The position number specifies the file to be opened.

POSMF mfn,recall

mfn	FET name of multifile set.
recall	Nonblank value if for auto recall; otherwise, blank.

The position number is specified in either word 11 (lfn+12) 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 (lfn+12) of the multifile 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 multifile 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.

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 is the function to be performed.

Parameter	Function (With Octal Code)
absent	Rewind (150).
NR	No rewind (130).
UNLOAD	Rewind and unload (170).
RETURN	Rewind and return. For tape files, decreases the number of tape units required for the job (174).

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.

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 is the function to be performed.

Parameter	Function (With Octal Code)
absent	Rewind (350).
NR	No rewind, although the result is the same as octal code 350 (330).
UNLOAD	Rewind and unload (370).
RETURN	Rewind and unload; do not swap reels (374).

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 in bits 0-24 in word 13 (lfn+14) 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 at 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, in steps 1 through 3.

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 after executing the device assignment algorithm to select a device for continuation of the file.

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 $0\text{-}16_8$ 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 (seven-track). For nine-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_8 system-logical-record or level 16_8 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 6.

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	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 (lfn+0) 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 (lfn+6) when the read is complete.

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_8 or 17_8 system-logical-record is read.

End-of-information is encountered.

File action error occurs.

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	File name
lev	Optional level number $0\text{-}17_8$. 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_8 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 (lfn+0) 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.

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 seven-track files, data is converted from external to internal BCD only. Conversion to display code is not made. No conversion takes place for nine-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.

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 (lfn+6)] must be set to the largest physical record that will be encountered. File mode must also be set.

Reading continues until:

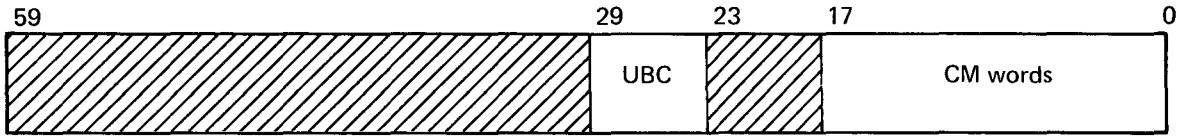
The next record will not fit into the circular buffer.

End-of-file 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.

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 (lfn+6), 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 nonrandom 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 (lfn+7) 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:

59	29	17	0
Executable Instructions	RJ	IOREAD	
0		Ifn	

59	29	17	0
Executable Instructions	RJ	IORR	
0		Ifn	
name		0	

59	29	17	0
Executable Instructions	RJ	IORR	
0		Ifn	
0		n	

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 nine-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	File name
recall	Optional recall parameter consisting of any nonblank 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 in bits 0-23 of word 7 (lfn+6) 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.

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 **File name**

lev Optional level number $0\text{-}17_8$. **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_8 the system changes the WRITER request to a WRITEF.

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_8. 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_8.

For S and L tapes, data in the buffer is written to tape and followed by a physical tape mark.

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 seven-track tape must be in internal BCD codes. Only internal to external BCD conversion is performed before writing; no conversion is performed for nine-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.

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:

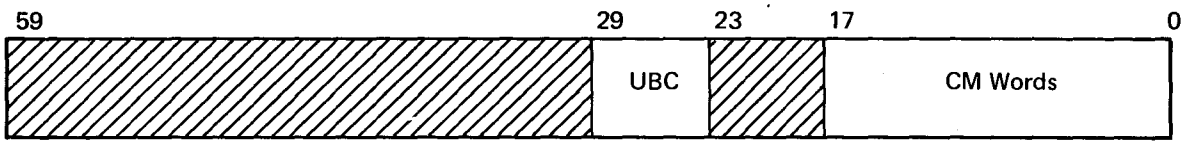
Buffer is empty.

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.

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.

1. 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.

2. 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.

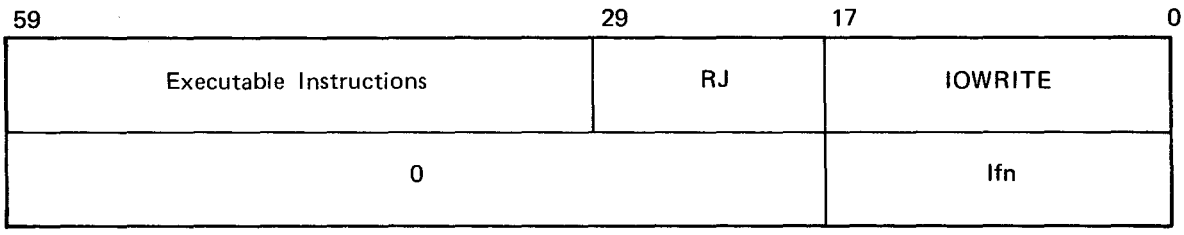
3. The random file index and the eighth word (lfn+7) of the FET are set to the correct record.

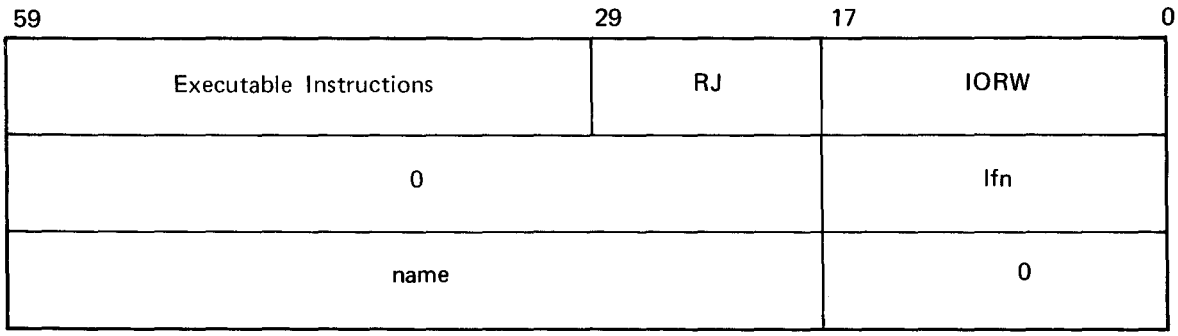
4. The working storage area is transferred to the circular buffer as the beginning of the new record identified in the WRITOUT.

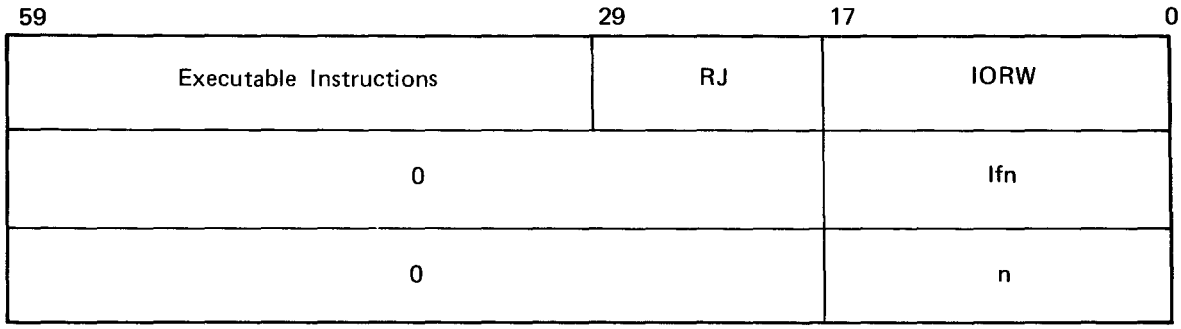
5. If the buffer contains at least one PRU of data, WRITE is called.

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 overwriting or underwriting 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 rewrite operation, REWRITER 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 PRUs 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_8 is written. If issued when the file is positioned at any point, two level 17_8 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.

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

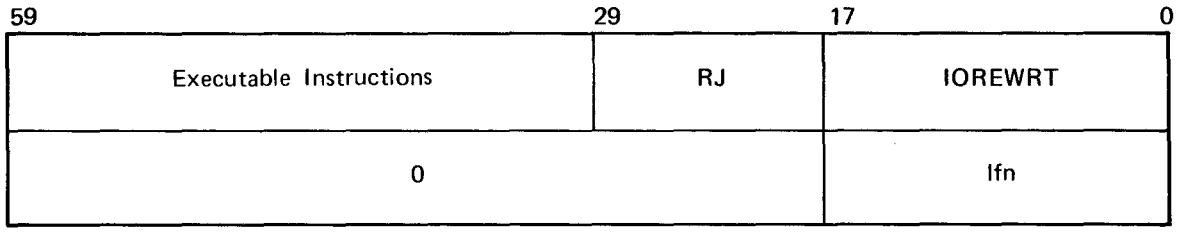
WRITIN lfn,/name/

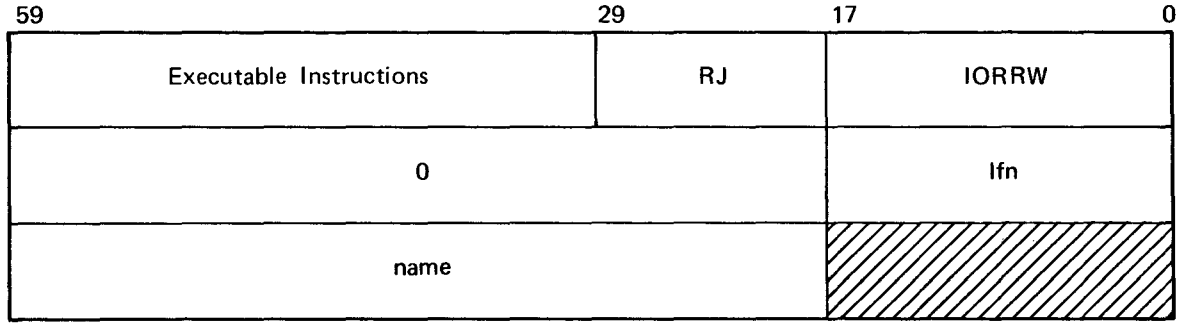
WRITIN lfn,n

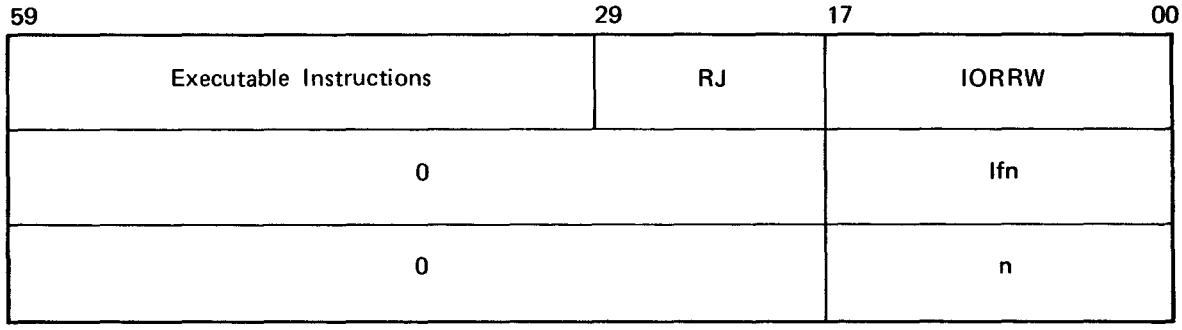
REWRITER is required to terminate a record, except when WRITIN or WRITOUT names another indexed record. In this case, a REWRITER of level 0 is forced before the new record is begun.

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 at lfn+1) 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 tape 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. If the SKIPF macro is extended from CPUTEXT, the maximum octal value of n is 377777.

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_8 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.

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_8 are recognized; any other level specified is assumed 0.

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
```

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.

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 multivolume tapes a REWIND rewinds the current volume and a subsequent forward motion initiates a backward reel swap positioning the file at its beginning. For labeled multivolume, single-file tapes, a REWIND rewinds the current volume and sets the volume number in the system tables 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 multifile labeled tapes, a REWIND rewinds the specified file to its beginning. If necessary, the operator is instructed to mount the previous volume. A REWIND that references a multifile name is illegal; the job terminates.

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
```

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 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.

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 **File name.**

* Optional end-of-job disposition indicator.

x Two-character disposition mnemonic.

Mnemonic	Meaning
PR	Print on any available printer
PE	Print on ASCII 95-character print train
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
IN	Place file in the input queue

†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.

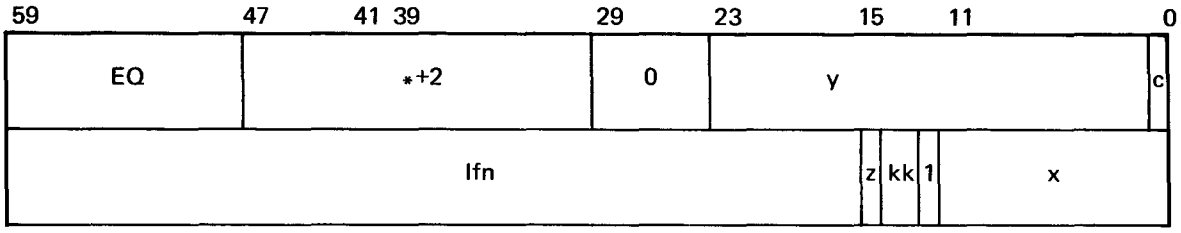
If k is C, two-character alphanumeric installation defined identifier of special forms or paper.

If k is I, two-character user identification.

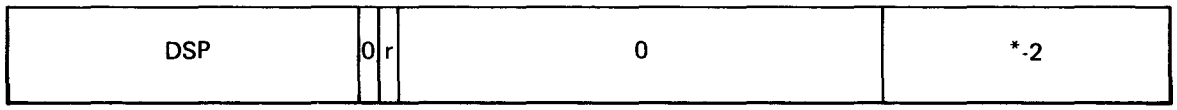
recall — Optional character indicating recall.

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

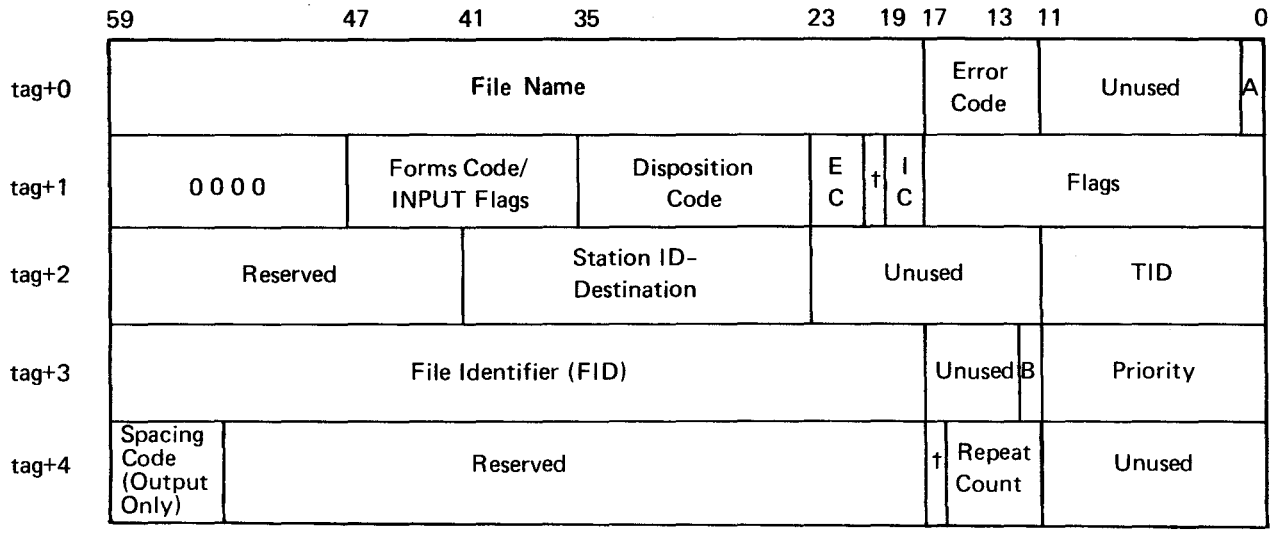
The ROUTE macro places a file in an input or output queue, evicts a file, or specifies attributes the file has when it is placed in an output queue. ROUTE has all the capabilities of DISPOSE. See the ROUTE control statement for a complete description of the ROUTE capabilities. The user must construct 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+0	18-59	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	Completion 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	Meaning
47	Unused
46	Unused
45	Do not catalog INPUT file

† Unused

Word	Bits	Field	Description
tag+1			**Bit** **Meaning**
			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	Two display code characters specifying a disposition code mnemonic as follows:

Mnemonic	Meaning
FR†	Print on microfilm recorder.
FL†	Plot on microfilm recorder.
HL†	Plot on hardcopy device.
HR†	Print on hardcopy device.
IN††	Place file in input queue.
LS	Print on 580-16 printer.
LR	Print on 580-12 printer.
LT	Print on 580-20 printer.
PT†	Plot on any available plotter.
PR	Print on any available printer.
PU	Punch.
SC	Evict the file

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

† Available only if supporting software is supplied by the installation.
†† Use of IN can be restricted by the installation.

Word	Bits	Field	Description
	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	IC=TRANS – Transparent (INTERCOM 5)

	0-17	Flag Bits	Indicate specified parameters when set to 1. (User must set bit 2, 4, 5, 7-10, 14, or 15 to 1 if corresponding parameter is being specified.)

Bit	Meaning
17	File name assigned by system is returned at tag+0, bits 18-59.
16	Unused.
15	Spacing code (tag+4, bits 54-59) is specified.
14	Repeat count (tag+4, bits 12-16) is specified.
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 (tag+1, bits 36-47) is specified.
9	Priority (tag+3, bits 0-11) is specified.
8	Internal characteristics (tag+1, bits 18-19) are specified.
7	External characteristics (tag+1, bits 21-23) are specified.
6	FID=* System appends two unique sequence characters to the caller's jobname (and the file identifier, if bit 5 is set).
5	File identifier (tag+3, bits 18-59) is specified.
4	Disposition code (tag+1, bits 24-35) is specified.
3	Route to remote station.
2	Terminal identifier (tag+2, bits 0-11) is specified.
1	Route to central site.
0	End-of-job (deferred ROUTE).

Word	Bits	Field	Description
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 (tag+0) 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
14	RMT ROUTING,NO ID – CENTRAL SITE ASSUMED
15	E1200 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

Error Code (octal)	Message
27	INVALID SID/DID – ROUTING IGNORED
30	JOBCARD ERROR – ROUTING IGNORED
31	THIS ROUTINE NOT ALLOWED – ROUTINE IGNORED
32	FNT SPACE CRITICAL – ROUTING IGNORED

See the NOS/BE 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	GETPF
ATTACH	PURGE
CATALOG	RENAME
EXTEND	SAVEPF

Information applicable to a control statement call is also applicable to a call through a permanent file macro. In addition, FDB and PERM macros are available.

The parameters used with the macros are the same as the parameters used with the corresponding control statements. Thus, more information is available under the control statement description.

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.

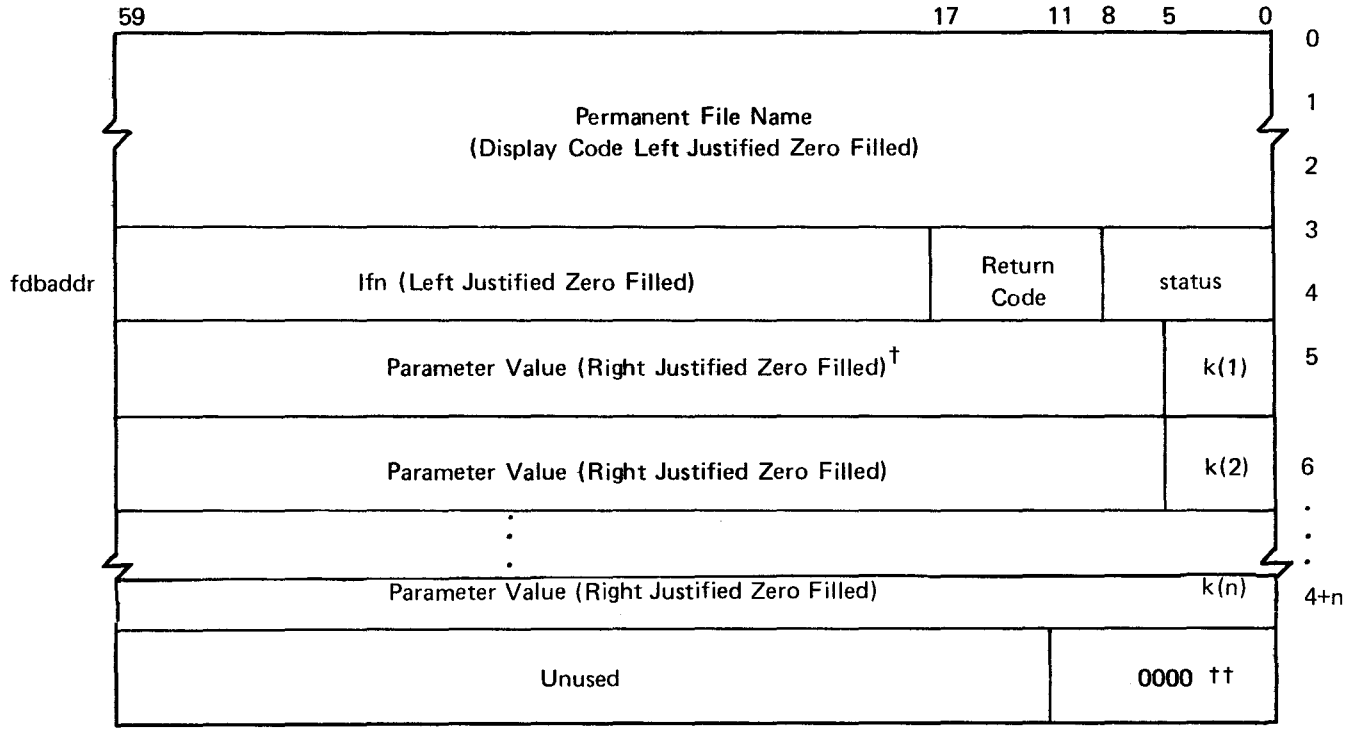
The field to the right of the macro name, FDB, is identical to that which could be on a control statement. Parameters are entered into the FDB as they are encountered in the list. The FDB is generated in-line during assembly whenever the macro is called.

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 a function is not successful, error codes may be returned in the FDB.

An error message is issued to the job dayfile unless the RT or RC parameter is specified in the function call. Fatal errors (errors with a return code of 70_8 or greater) are written to the job dayfile.

The FDB generated by the macro has the form:



Field	Description
k(n)	Parameter identifier in octal or display code:

k (Octal)	Keyword Parameter	Parameter Value and Description
00		End of FDB list
02	RP	Retention period; days; in binary†††
03	CY	Cycle number; in binary
04	TK	Turnkey password (display code)

†The SN parameter (keyword 40) is left justified with zero fill; the VSN parameter (keyword 41) is contained in a 6-character field (bits 59-24) with leading display code zero fill.

††The system only checks bits 0-11 in this word.

†††If cycle 0 is specified, the actual cycle referenced will be returned in the parameter value upon completion of the request.

Field **Description**

k (Octal)	Keyword Parameter	Parameter Value and Description
05	CN	Control password (display code)
06	MD	Modify password (display code)
07	EX	Extend password (display code)
10	RD	Read password (display code)
11	MR	Multiread parameter (any nonzero binary value)
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 (any nonzero binary value)
32	ST	Station ID (display code)
33	RW	Multiread with single rewrite (any nonzero binary value)
40	SN	Setname (display code, left justified)
41	VSN	Volume serial number (display code)
43	RB	PURGE RB conflict parameter (any nonzero binary value)
53	UV	Universal password (display code)

Status — Status bits:

Bit	Meaning
0	Completion bit
1	Unused
2-5	Function code bits (binary) 0010 GETPF 1000 PURGE 0100 SAVEPF 1010 RENAME 0110 EXTEND 1100 PERM 0111 ALTER
6	Set if RC or RT not specified; issue dayfile message; 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

Field	Description	
	Value (Octal)	Meaning
	004	No room for extra cycle (limit is five)
	005	Permanent file catalog (PFC) is full
	006	No lfn or pfn specified in the FDB
	007	Not used
	010	Latest index was not written for a random file
	011	File is not on PF device
	012	File is not cataloged, SN=setname (setname is the set name of the device set searched)
	013	Archive retrieval aborted
	014	Bad LPF communication
	015	Cycle number limit reached; maximum value of cycle number is 999
	016	Permanent file directory (PFD) is full
	017	Function attempted on nonpermanent file
	020	Function attempted on nonlocal file
	021	Improper archive retrieval call
	022	File was never assigned to a device
	023	Cycle is incomplete or was dumped
	024	PF already attached
	025	File is archived
	026	Illegal character in FDB parameter
	027	Illegal lfn
	030	File dumped
	031	Illegal function code
	032	Purge attempt ignored; use RB parameter
	033	ALTER needs exclusive access
	034	FDB is too large
	035	File already in system
	036	No APF space
	037	Permission conflicts (file is attached elsewhere with exclusive access permissions)
	040	Illegal setname specified
	041	Device set not mounted at this control point
	042	RBT chain is too large for PFC
	043	File resides on unavailable device
	044	File not available
	045-067	Not used
	070	PFM stopped by system

The following conditions will unconditionally cause abnormal job termination.

071	Incorrect permission
072	File definition block address invalid (not returned to FDB)
073	I/O error on PFD/PFC read/write

Unless the RC parameter is specified, all errors terminate the job. Any job that attempts a privacy breach is terminated. All internal permanent file malfunctions are system errors that cause job termination.

PERM MACRO

The PERM function is available only as a system macro. A running program can determine if a file is a non-permanent local file or what permissions have been granted to a currently attached permanent file.

The format of the PERM macro is

```
PERM    fdbaddr,RC
```

The lfn of an attached permanent file should be given in the FDB. This macro produces a 5-bit code in the fdbaddr return code field. The bits represent the following information.

Bit 4	1	Nonpermanent file.
	0	Permanent file.
Bits 3-0	1000	Control permission.
	0100	Modify permission.
	0010	Extend permission.
	0001	Read permission.

A return code of zero signifies that the lfn is not in the FNT for the control point (a nonexistent file) or that some other error occurred.

Perm example:

```
FDBA        FDB         DFLN,PFILE,CY=1,PW=XXX,ID=ABDC
            .
            .
            .
            ATTACH      FDBA,RC
            .
            .
            .
            PERM        FDBA
```

Assuming the file is cataloged with passwords required for control and modify permissions, the ATTACH request generates control permission by the password XXX and read and extend permissions by default. A subsequent PERM request causes an octal 13 value to be returned to the FDB. The code indicates a permanent file is attached with read, extend, and control permission.

For permanent file functions, 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 or any expression that would be valid in the variable field of a set register Xi (SXi) COMPASS command. It must form the address of the fifth word of the FDB.
RC	Optional parameter that causes return codes to be available in FDB and returns control to the program on nonfatal errors. Permanent file queuing occurs when a file cannot be accessed immediately. If RC and RT parameters are both specified, the RC parameter is ignored.
RT	Optional parameter that inhibits permanent file queuing and causes return codes to be available in FDB.
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 parameters RC, RT, and NR are order independent.

ALTER MACRO

The format of the ALTER macro is

ALTER fdbaddr,RC,RT,NR

The ALTER function causes the current position of an attached permanent file (designated by lfn in the FDB) to be recorded as end-of-information in the PFC of that file. Permissions needed to perform the ALTER function depend upon the context in which the function is issued. If the current position of the file is less than the file EOI (as attached), modify permission, extend permission, and exclusive access are required. If the current position is greater than the file EOI, ALTER operates similarly to the EXTEND function and extend permission is required.

ATTACH MACRO

The format of the ATTACH macro is

```
ATTACH  fdbaddr,RC,RT,NR
```

The ATTACH request requires the lfn, pfn, and ID parameters in the FDB. The following parameters are optional.

CY	Cycle number to be attached.
PW	Password list.
MR	Multiread access.
LC	Lowest cycle number.
RW	Multiread/rewrite access.
EC	ECS buffering for I/O.
SN	Set name.
PS	Position of file.
UV	Universal password.

If RC is specified, the user is notified of a nonfatal error condition by an error return code at fdbaddr in the FDB. If RT is specified, the system handles the call as a real time call. The system suppresses the routing of informative and diagnostic messages normally issued to the dayfile when either the RC or RT parameter is specified. If RT is not specified, and any of the following conditions prevail, a job issuing an attach request is queued for the requested file.

- File is not available for exclusive access by requesting job.
- Attached permanent file (APF) table is full.
- Archived file is temporarily unavailable. The ATTACH request causes a LOADPF job to be set up and scheduled through the tape scheduler. The job requesting the ATTACH is swapped out until the file is available.
- Permanent file utility is running.

If the CY parameter is zero or not present, and the permanent file has multiple cycles, the default cycle attached is the one with the largest cycle number, presumably the latest cataloged. If the CY parameter is present, and that particular cycle number is not known to the system, the request cannot be honored. If both LC and CY are specified, LC is ignored and the conflict is resolved.

System evaluation of passwords establishes the type of access granted to the user for each file. Subsequent to an ATTACH request, the user cannot access the file for which he does not have permission in any way. For example, if ATTACH results in only read permission, the user cannot subsequently attempt to modify or extend.

ATTACH does not preclude opening the file. The success of an OPEN request depends upon the permission granted when the file is attached. If the file is attached to another control point and multiread access is not possible, PFM waits for access to the file.

Attach example:

```
FDBZ        FDB             LF,MFILE,MR=1,PW=Y,ID=ABC,CY=1
            .
            .
            ATTACH          FDBZ,RT
```

Permanent file MFILE is referenced again. The CY = 1 specification ensures that cycle 1 is attached.

In the FDB, only the password for turnkey appears; extend and read permissions are granted by default. In this example, the macro contains MR = 1; therefore, all permissions except READ are ignored, making the file available for multiread access.

If the file requested by the macro is unavailable, the presence of the RT parameter causes an octal code 37 to be returned at location FDBZ.

```
            CATALOG         FDB1
            CLOSE           LF1,UNLOAD,RECALL
            .
            .
            .
            ATTACH          FDB1
            .
            .
            .
FDB1        FDB             LF1,PERMF,TK=T,MD=M,EX=E,CN=C,PW=T,ID=ABC
```

The preceding example illustrates several points. Assuming that local file LF1 has been created, it is cataloged as cycle 1 (by default) of permanent file PERMF. The file is protected by turnkey, control, modify, and extend passwords. The PW parameter in the FDB is ignored in cataloging.

After cataloging is complete, a CLOSE/UNLOAD logically detaches the file from the job.

As illustrated, the file PERMF now can be reattached. Although not mandatory, the same FDB is used to conserve CM space. When PERMF is attached, the default cycle number is the largest cataloged; therefore, cycle 1 is the only cycle present. The PW parameter contains the turnkey password giving READ access permission by default. This example illustrates an implicit read-only attach.

The same example is shown with two FDBs.

```
FDB1        FDB             LF1,PERMF,TK=T,MD=M,EX=E,CN=C,ID=ABC
FDB2        FDB             LF1,PERM,PW=T,ID=ABC
            .
            .
            .
            CATALOG         FDB1
            CLOSE           LF1,UNLOAD,RECALL
            .
            .
            .
            ATTACH          FDB2
```

CATALOG MACRO

The format of the CATALOG macro is

```
CATALOG     fdbaddr,RC,RT,NR
```

For this request, the required parameters in the FDB are lfn, pfn, and ID. If the permanent file name is unique to the ID specified, the request is considered an initial catalog. The initial catalog defines the passwords necessary to access any of up to five cycles that can be cataloged with the same pfn and ID. If the CY parameter is not specified, it is assumed to be 1. The following parameters are relevant on an initial catalog.

CY	Cycle number.
XR	Control, modify, extend, common password definition.
CN	Control password.
MD	Modify password.
EX	Extend password.
RD	Read password.
TK	Turnkey password.
RP	Retention period.
FO	File validity check.
RW	Read with rewrite permission.
MR	Multiread access.
PW	Password list.
AC	Account parameter.

If a file with the same pfn and ID has already been cataloged, the request is considered a new cycle catalog. If a CY parameter is not specified, it is assumed to be one larger than the highest cycle. Control permission must be established to do a new cycle catalog. The following parameters are relevant on a new cycle catalog.

CY	Cycle number.
PW	Password list.
RP	Retention period.
FO	File validity check.
RW	Read with rewrite permission.
MR	Multiread access.

If RC is specified, the user is notified of a nonfatal error condition by an error return code at fdbaddr in the FDB.

If RT is specified, the call is regarded as real-time. Specifying the RT option forces the RC option. In both cases, informative and diagnostic messages to the dayfile are suppressed.

Initial catalog example:

```
FDBA        FDB         LF1,MFILE,CN=Z,MD=X,TK=Y,ID=ABC
                .
                .
                .
            CATALOG     FDBA,RC
```

LF1 is assumed to exist on a valid permanent file device as a local file to this control point.

The CATALOG macro references FDBA, which contains the necessary parameters to make LF1 permanent.

Since RC is specified on the CATALOG macro, control is returned to the user on a nonfatal error, and a return code is made available in location FDBA (bits 17 through 9). If RC is not specified, all errors result in termination and a diagnostic message.

New cycle catalog example:

```
            CATALOG     FDB5
                .
                .
                .
FDB5        FDB         LF16,MFILE,CY=12,PW=Y,Z,ID=ABC
```

This job adds a second cycle to permanent file MFILE, created in the preceding example. File LF16 is assumed to be a valid local file on a permanent file device. The PW parameter is used to submit the passwords needed to obtain control permission. Had the initial catalog attempt aborted, MFILE would not exist, and this new cycle attempt would be processed as an initial cataloging. If successful as an initial catalog, the file would be unprotected as no passwords are defined in the FDB. An alternate form of the FDB could be used as follows:

```
FDB5        FDB         LF16,MFILE,CY=12,PW=Y,Z,ID=ABC,TK=Y,CN=Z,MD=X
```

The preceding FDB would perform equally as well for a new cycle catalog because the TK, CN, and MD parameters would be ignored. If initial cataloging has failed, this FDB would catalog the file with protection, and the PW parameter list would be ignored.

EXTEND MACRO

The format of the EXTEND macro is

```
EXTEND      fdbaddr,RC,RT,NR
```

Local extensions can be written at the end-of-information point of an attached permanent file and an extend function can be issued, thus extending the length of the permanent file. The file must be attached with extend permission granted.

In the FDB, the required parameter is lfn; the extend password, if defined, must appear in the PW list. The extended section of the file acquires the privacy controls of the permanent file.

If lfn is an indexed file, the current index is assumed to be the only valid index for the entire file. Random files must be closed before an EXTEND request is made.

Extend example:

```
                ATTACH          FDBX
                .
                .
                .
                EXTEND          FDBX
                .
                .
                .
FDBX            FDB             LF1,PROGLIB1,ID=XYZ
```

The program that attaches permanent file PROGLIB1 as local file LF1 writes beyond the end-of-information. Assuming that no password was required for extend permission, it is given by default. Prior to program termination, the EXTEND request would make permanent any additions written to the file.

GETPF MACRO (MULTIMAINFRAME ONLY)

The format of the GETPF macro is

```
GETPF           fdbaddr,RC,RT,NR
```

The GETPF request requires the lfn, pfn, ID, and ST parameters in the FDB. Specifying both SN and VSN parameters allows access to a file on a private set at the specified mainframe. Other optional parameters are the same as those for the ATTACH function except for RW, which is ignored if present. Refer to the ATTACH function for more detail.

GETPF prepares for staging the requested permanent file from the mainframe specified by the ST parameter and attaches a local copy of that file to the calling routine. The file must be opened before actual staging occurs. The mainframe specified by the ST parameter can be the linked mainframe or the local mainframe. In either case, two files exist after the file is staged, the permanent file and a local copy of the file attached to the calling routine.

The local copy of the file is processed, even if the permanent file specified resides at the same mainframe.

PURGE MACRO

The format of the PURGE macro is

```
PURGE           fdbaddr,RC,RT,NR
```

A cycle of a file can be removed from the catalog of permanent files by the PURGE function. In the macro, fdbaddr is required; RC and RT are as for CATALOG. In the FDB, the lfn is the only required parameter if the file was attached before the purge function was issued; all other parameters are ignored. The optional parameters are CY, SN, LC, EC, RB, PW, and UV. Control permission must be granted, or the job is terminated.

For the macro request, the FDB used when the file was attached can be referenced or a new FDB can be used. Only one cycle of a file can be purged at a time. When the last cycle of the file is purged, the entire permanent file name entry is removed from the directory and catalog.

A user attempting to purge a permanent file already attached must specify the lfn under which the file was attached. This purge is by local file name, and the user need provide only the lfn in the FDB.

To purge a file not already attached, the user must specify a local file name not in use at his control point. The permanent file name, ID, and password list must be given. If the file resides on a private device set, the SN parameter must be given.

To purge a permanent file at a linked mainframe, the user must specify the ST parameter. Specifying both SN and VSN parameters with the ST parameter allows a file to be purged from a private set at the specified mainframe. If RB=1 is specified in the FDB, and if the RB conflict flag is set in the PFC by RECOVER, the RBs in the chain are zeroed, and storage is not released to the system. To prevent further use of the file, the FNT permissions are reduced to control only.

RENAME MACRO

The format of the RENAME macro is

```
RENAME      fdbaddr,RC,RT
```

Any or all information cataloged by the user can be replaced through the RENAME function. The file must be attached to the requesting job with all permissions granted. A file owner can change permanent file name, cycle numbers, passwords, and even the user ID.

In the FDB for this request, lfn is the only required parameter. The specified parameters cause replacement of existing parameter information if they contradict the cataloged information. If they duplicate the cataloged information, the parameters are ignored.

Parameters which could result in replacement are:

pfn	Permanent file name.
ID	Owner identification.
RP	Retention period.
CY	Cycle number.
TK	Turnkey password.
RD	Read password.
EX	Extend password.
MD	Modify password.

CN Control password.

AC Account name.

XR Common password definition.

The PW parameter may be specified to submit the public password if the file ID is to be renamed PUBLIC. Other parameters in the FDB are ignored. Changing the ID, pfn, or passwords for any cycle cataloged changes all cycles. No ID, pfn, or CY changes are permitted if any of the cycles have been dumped (mode 2 dump) or archived, as retrieval of such files would be impossible. RC and RT are as for CATALOG.

An attempt to rename pfn/ID is ignored when the new pfn/ID pair currently exist; however, the remainder of the specified changes still occur.

Rename example:

```
                ATTACH          FDBA
                RENAME          FDBB
                .
                .
                .
FDBA            FDB             DFILE,MFILE,PW=Z,Y,X,ID=ABC
FDBB            FDB             DFILE,PFILE2,RD=W,MD=,CN=ZZ
```

Assuming that MFILE is cataloged with X, Y, and Z as passwords for modify, turnkey, and control, read access is given by default when DFILE is attached as a local file. By RENAME action, the cataloged permanent file name is replaced with PFILE2. A new password, ZZ, replaces the existing password for control permission; a read password, W, is cataloged for the nonexistent read password. The password for modify permission is removed, and none replaces it. The owner's ID remains unchanged. Since no cycle number is given in FDBA, the cycle with the largest number is attached; renaming does not change the existing cycle number, as no replacement is given in FDBB.

```
FDB1            FDB             LFILE,MFILE,CY=9,RD=Y,ID=ABC
FDB2            FDB             LFILE,MFILE,CY=8,PW=X,Y,Z,ID=ABC
FDB3            FDB             LFILE,MFILE,RD=Z
                .
                .
                .
                ATTACH          FDB2
                RENAME          FDB1
                .
                .
                .
                RENAME          FDB3
```

This example illustrates that for renaming purposes, the same file can be called more than once in a job. If the read password was originally cataloged as X, it is changed to Y when the file is renamed as cycle 9, and then finally changed to Z. The appearance of an identical ID parameter in FDB1 is ignored.

SAVEPF MACRO (MULTIMAINFRAME ONLY)

The format of the SAVEPF macro is

```
SAVEPF        fdbaddr,RC,RT,NR
```

The SAVEPF request requires the lfn, pfn, ID, and ST parameters in the FDB. SN and VSN parameters allow a copy of the file to be made permanent on a private set at the specified mainframe. Other optional parameters are the same as those for the CATALOG function. Refer to the CATALOG function for more details.

SAVEPF stages a local file specified by the lfn to be cataloged at the mainframe associated with the ST parameter. The mainframe where the file is to be cataloged can be a linked mainframe or the local mainframe. In either case, two files exist after the call, a new or updated permanent file and the local file that is attached to the calling routine.

SYSTEM TEXTS

System texts provide commonly used macro, micro, and symbol definitions for use in COMPASS source programs. The system 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 statements. S or G parameters can also be used on FTN5 control statements when FORTRAN 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 NOS/BE Installation Handbook.

COMMON DECKS

ACTCOM - System action request macros:

IXi Xj/Xk	DATE	GETJCI	RECALL	STATUS
IXi Xj/Xk,Bn	DISPOSE	IOTIME	RECOVR	SYSCOM
ABORT	ENDRUN	JDATE	REQUEST	SYSTEM
ACQUIRE	FILESTAT	LOADREQ	ROUTE	TIME
CHECKPT	FILINFO	MEMORY	RTIME	TRANSF
CLOCK	GETACT	MESSAGE	SETJCI	VERIFYJ
CONTRLC				

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 using SYS= and CIO=:

BKSP	READC	READSKP	SKIPB	WRITEC
BKSPRU	READEI	READW	SKIPEI	WRITEF
CHECKF	READH	RETURN	SKIPF	WRITEH
CLOSE	READLS	REWIND	SKIPFB	WRITEN
CLOSER	READN	REWRITE	SKIPFF	WRITED
EVICT	READO	REWRITEF	UNLOAD	WRITER
OPEN	READNS	REWRITER	WPHR	WRITES
POSMF	READS	RPHR	WRITE	WRITEW
READ				

COMAFET - File environment table generation macros:

FILEB	LABEL	RFILEB
FILEC		RFILEC

COMAREG - Replacement for R= pseudo-instruction.

COMCECS - Contains the ECS interpretive routines (refer to appendix E).

COMCECM - Contains the redefinition of the RE and WE COMPASS instructions (refer to appendix E).

COMSRAS - System communication symbols:

Contains definitions of the system communications symbols described in this section under the heading SYSTEM COMMUNICATION MACROS.

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 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, ECSCOM, CIOCOM, CMRDEF, and ECSDEF.
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.

Common decks ACTCOM, COMACIO, COMAFET, COMAREG, and COMSRAS.

ECSTEXT — System text which contains the redefinition of the RE and WE COMPASS instructions. Common deck COMCECM (refer to appendix E).

IOTEXT — System text for central processor programs using CDC CYBER Record Manager.

Common decks ACTCOM, COMSRAS, and 6RMCOM.

IPTEXT — Installation parameter system text.

Contains the macros CEQU, CMICRO, and IPARAMS, whose body is the IPARAMS common deck.

LDRTEXT — System text for central processor programs using CDC 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.

SRMSTXT — Cataloged as an additional system text when multi-mainframe shared RMS is installed.

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.

The following additional system texts are provided by product set members.

ALGTEXT — Contains COMPASS-coded macros used to expand application areas of ALGOL.

SMTEXT — Contains macros for central processor programs that call Sort/Merge.

RMERTXT — Contains CYBER Record Manager error dictionary.

A character set is composed of graphic and/or control characters. A code set is a set of codes used to represent each character within a character set.

A graphic character may be displayed at a terminal or printed by a line printer. Examples are the characters A through Z and the digits 0 through 9. A control character initiates, modifies, or stops a control operation. An example is the backspace character that moves the terminal carriage or cursor back one space. Although a control character is not a graphic character, a terminal may produce a graphic representation when it receives a control character.

All references within this manual to ASCII character sets or ASCII code set refer to the character sets and code set defined in the American National Standard Code for Information Interchange (ASCII, ANSI Standard X3.4-1977).

NOS/BE supports the following character sets.

- CDC graphic 64- (or 63-) character set.
- ASCII 128-character set.
- ASCII graphic 64- (or 63-) character set.
- ASCII graphic 95-character set.

Each installation selects either the 64-character set or the 63-character set. The differences between the two are described in Character Set Anomalies later in this appendix.

NOS/BE supports the following code sets.

- Display code.
- 12-bit ASCII code.

Display code is a set of 6-bit codes from 00_8 to 77_8.

The 12-bit ASCII code is the ASCII 7-bit code (as defined by ANSI Standard X3.4-1977) right-justified in a 12-bit byte. Assuming that the bits are numbered from the right starting with 0, bits 0 through 6 contain the ASCII code, bits 7 through 10 contain zeros, and bit 11 distinguishes the 12-bit ASCII 0000_8 code from the end-of-line byte. The 12-bit codes are 0001_8 through 0177_8 and 4000_8.

CHARACTER SET ANOMALIES

NOS/BE interprets the codes for the colon and the percent graphic characters differently when the installation selects the 63-character set rather than the 64-character set. In tables A-1 and A-2 the codes for the colon and percent graphic characters in the 64-character set are unshaded; the codes for the colon and percent graphic characters in the 63-character set are shaded. If an installation uses the 63-character set, the colon graphic character is always represented by a 63_8 code.

Also, two or more consecutive 00_8 codes may be confused with an end-of-line byte and should be avoided.

CHARACTER SET TABLES

This appendix contains character set tables for INTERCOM users, batch users, and magnetic tape users. Table A-1 is for INTERCOM users, and table A-2 is for batch users. Tables A-3, A-4, and A-5 are for magnetic tape users and list the magnetic tape codes and their display code equivalents.

The character set tables are designed so the user can either find the character represented by a code (such as in a dump) or find the code that represents a character. To find the character represented by a code, the user looks up the code in the column listing the appropriate code set and then reads across the table to find the character on that line in the column listing the appropriate character set. To find the code that represents a character, the user first looks up the character and then reads across the table to find the code on the same line in the appropriate code column.

INTERCOM USERS

Table A-1 shows the character sets and code sets available to an ASCII code terminal user. When communicating with a terminal, NOS/BE displays by default the ASCII graphic 64- or 63-character set and interprets all input and output as display code. COMPASS and FORTRAN users can elect to use 12-bit ASCII code if the terminal in use will support the code set selected. Refer to the INTERCOM Reference Manual.

BATCH USERS

Table A-2 lists the CDC graphic 64- or 63-character set, the ASCII graphic 64- or 63-character set, and the ASCII graphic 95-character set. It also lists the code sets and card punch codes (026 and 029) that represent the characters.

The 64- or 63-character sets use display code as their code set; the 95-character set uses 12-bit ASCII code. The 95-character set is composed of all the characters in the ASCII 128-character set that can be printed at a line printer (refer to Line Printer Use later in this appendix). Only 12-bit ASCII code files can be printed using the ASCII graphic 95-character set.

LINE PRINTER USE

The print train used determines which batch character set is printed (refer to the ROUTE control statement in section 4). Following is a list of the print trains and their corresponding batch character sets.

Character Set	Print Train
CDC graphic 64- or 63-character set	596-1
ASCII graphic 64- or 63-character set	596-5
ASCII graphic 95-character set	596-6

The characters of the default 596-1 print train are listed in table A-2 in the column labeled CDC Graphic (64 or 63 Characters); the 596-5 print train characters are listed in table A-2 in the column labeled ASCII Graphic (64 or 63 Characters); and 596-6 print train characters are listed in table A-2 in the column labeled ASCII Graphic (95 Characters).

If a transmission error occurs when printing a line, the system stops printing and alerts the operator, who must decide what action to take. The operator ususally decides to rewind the print file and return it to the print queue. An installation option is available which allows print errors to be automatically overridden.

If an unprintable character exists in a line (that is, a 12-bit ASCII code outside the range 0040_8 through 0176_8), the number sign (#) appears in the first printable column of a print line, and a space replaces the unprintable character.

PUNCHED CARD INPUT AND OUTPUT

Punched card data falls into two categories.

- Coded data.
- Binary data.

Coded data is data converted from (or to) a punched card code to (or from) a character set code recognizable by a software product as representing a conventional character. Binary data does not require such conversion. Binary data in this context is usually manipulated in offline operations involving card-to-tape or tape-to-card transmissions, storage of relocated programs, and so forth.

Under NOS/BE, alternative card keypunch codes are available for input of the CDC characters and or their ASCII equivalents ! and .

Depending on which (if any) installation option is selected, the system assumes an input deck has been punched either in 026 or in 029 keypunch mode (regardless of the character set in use). The alternative mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job card or any 7/8/9 card. The mode remains in effect throughout the job unless it is changed by a mode specified on a subsequent 7/8/9 card.

MAGNETIC TAPE USERS

Coded data to be copied from mass storage to magnetic tape is assumed to be represented in display code. NOS/BE converts the data to external BCD code when writing a coded seven-track tape and to ASCII or EBCDIC code (as specified on the tape assignment statement) when writing a coded nine-track tape.

Because only 63 characters can be represented in seven-track even parity, one of the 64 display codes is lost in conversion to and from external BCD code. The following shows the difference in conversion depending on the character set (63 or 64 characters) which the system uses. The ASCII character for the specified character code is shown in parentheses. The output arrow shows how the display code changes when it is written on tape in external BCD. The input arrow shows how the external BCD code changes when the tape is read and converted to display code.

63-Character Set

Display Code		External BCD		Display Code
00		16 (%)		00
33 (0)	Output →	12 (0)	→ Input	33 (0)
63 (:)		12 (0)		33 (0)

64-Character Set

Display Code		External BCD		Display Code
00 (:)		12 (0)		33 (0)
33 (0)	Output →	12 (0)	→ Input	33 (0)
63 (%)		16 (%)		63 (%)

If lowercase ASCII or EBCDIC code is read from a nine-track coded tape, it is converted to its uppercase 6-bit display code equivalent. To read and write lowercase ASCII or EBCDIC characters, the user must assign the tape in binary mode and perform his own conversion of the binary data.

Tables A-3 and A-4 show the character set conversions for nine-track tapes. Table A-3 lists the conversions to and from the ASCII character code and display code. Table A-4 lists the conversions between the EBCDIC character code and the display code. Table A-5 shows the character set conversions between external BCD and display code for seven-track tapes.

TABLE A-1. INTERCOM CHARACTER SETS

ASCII Graphic (64 Char)	ASCII Character (128 Char)	Display Code	12-Bit ASCII Code	ASCII Graphic (64 Char)	ASCII Character (128 Char)	Display Code	12-Bit ASCII Code
: colon †		00†		# num. sign	# num. sign	60	0043
Display code 00 is undefined at sites using the 63-character set.				[l. bracket	[l. bracket	61	0133
				] r. bracket	] r. bracket	62	0135
A	A	01	0101	% †	% †	63†	0045
B	B	02	0102	: colon	: colon	63	0072
C	C	03	0103	" quote	" quote	64	0042
D	D	04	0104	_ underline	_ underline	65	0137
E	E	05	0105	!	!	66	0041
F	F	06	0106	& ampersand	& ampersand	67	0046
G	G	07	0107				
				' apostrophe	' apostrophe	70	0047
				?	?	71	0077
H	H	10	0110	<	<	72	0074
I	I	11	0111	>	>	73	0076
J	J	12	0112	@		74	
K	K	13	0113	\ rev. slant	\ rev. slant	75	0134
L	L	14	0114	^ circumflex		76	
M	M	15	0115	; semicolon	; semicolon	77	0073
N	N	16	0116				
O	O	17	0117		@		0100
					^ circumflex		0136
					: colon †		0072
P	P	20	0120		%		0045
Q	Q	21	0121		` grave accent		0140
R	R	22	0122				
S	S	23	0123		a		0141
T	T	24	0124		b		0142
U	U	25	0125		c		0143
V	V	26	0126		d		0144
W	W	27	0127		e		0145
					f		0146
X	X	30	0130		g		0147
Y	Y	31	0131				
Z	Z	32	0132		h		0150
0	0	33	0060		i		0151
1	1	34	0061		j		0152
2	2	35	0062		k		0153
3	3	36	0063		l		0154
4	4	37	0064		m		0155
					n		0156
					o		0157
5	5	40	0065				
6	6	41	0066		p		0160
7	7	42	0067		q		0161
8	8	43	0070		r		0162
9	9	44	0071		s		0163
+	+	45	0053		t		0164
-	-	46	0055		u		0165
*	*	47	0052		v		0166
					w		0167
/	/	50	0057		x		0170
(	(	51	0050		y		0171
)	)	52	0051		z		0172
$	$	53	0044		{ left brace		0173
=	=	54	0075		| vert. line		0174
space	space	55	0040		} right brace		0175
, comma	, comma	56	0054		~ tilde		0176
. period	. period	57	0056		DEL		0177

†The interpretation of this character or code may depend on its context. Refer to Character Set Anomalies elsewhere in this appendix.

00028
1 OF 2

TABLE A-1. INTERCOM CHARACTER SETS (Contd)

ASCII Graphic (64 Char)	ASCII Character (128 Char)	Display Code	12-Bit ASCII Code	ASCII Graphic (64 Char)	ASCII Character (128 Char)	Display Code	12-Bit ASCII Code
	NUL		4000		DLE		0020
	SOH		0001		DC1		0021
	STX		0002		DC2		0022
	ETX		0003		DC3		0023
	EOT		0004		DC4		0024
	ENQ		0005		NAK		0025
	ACK		0006		SYN		0026
	BEL		0007		ETB		0027
	BS		0010		CAN		0030
	HT		0011		EM		0031
	LF		0012		SUB		0032
	VT		0013		ESC		0033
	FF		0014		FS		0034
	CR		0015		GS		0035
	SO		0016		RS		0036
	SI		0017		US		0037

00028
2 OF 2

TABLE A-2. BATCH CHARACTER SETS

CDC Graphic (64 Char)	ASCII Graphic (64 Char)	ASCII Graphic (95 Char)	Display Code	12-Bit ASCII Code	Punch Code	
					026	029
: colon†	: colon†		00†		8-2	8-2
Display code 00 is undefined at sites using the 63-character set.						
A	A	A	01	0101	12-1	12-1
B	B	B	02	0102	12-2	12-2
C	C	C	03	0103	12-3	12-3
D	D	D	04	0104	12-4	12-4
E	E	E	05	0105	12-5	12-5
F	F	F	06	0106	12-6	12-6
G	G	G	07	0107	12-7	12-7
H	H	H	10	0110	12-8	12-8
I	I	I	11	0111	12-9	12-9
J	J	J	12	0112	11-1	11-1
K	K	K	13	0113	11-2	11-2
L	L	L	14	0114	11-3	11-3
M	M	M	15	0115	11-4	11-4
N	N	N	16	0116	11-5	11-5
O	O	O	17	0117	11-6	11-6
P	P	P	20	0120	11-7	11-7
Q	Q	Q	21	0121	11-8	11-8
R	R	R	22	0122	11-9	11-9
S	S	S	23	0123	0-2	0-2
T	T	T	24	0124	0-3	0-3
U	U	U	25	0125	0-4	0-4
V	V	V	26	0126	0-5	0-5
W	W	W	27	0127	0-6	0-6
X	X	X	30	0130	0-7	0-7
Y	Y	Y	31	0131	0-8	0-8
Z	Z	Z	32	0132	0-9	0-9
0	0	0	33	0060	0	0
1	1	1	34	0061	1	1
2	2	2	35	0062	2	2
3	3	3	36	0063	3	3
4	4	4	37	0064	4	4
5	5	5	40	0065	5	5
6	6	6	41	0066	6	6
7	7	7	42	0067	7	7
8	8	8	43	0070	8	8
9	9	9	44	0071	9	9
+	+	+	45	0053	12	12-8-6
-	-	-	46	0055	11	11
*	*	*	47	0052	11-8-4	11-8-4

† The interpretation of this character or code may depend on its context. Refer to Character Set Anomalies elsewhere in this appendix.

00029
1 OF 3

TABLE A-2. BATCH CHARACTER SETS (Contd)

CDC Graphic (64 Char)	ASCII Graphic (64 Char)	ASCII Graphic (95 Char)	Display Code	12-Bit ASCII Code	Punch Code	
					026	029
/	/	/	50	0057	0-1	0-1
(	(	(	51	0050	0-8-4	12-8-5
)	)	)	52	0051	12-8-4	11-8-5
$	$	$	53	0044	11-8-3	11-8-3
=	=	=	54	0075	8-3	8-6
space	space	space	55	0040	no punch	no punch
, comma	, comma	, comma	56	0054	0-8-3	0-8-3
. period	. period	. period	57	0056	12-8-3	12-8-3
≡ equiv.	# num. sign	# num. sign	60	0043	0-8-6	8-3
[l. bracket	[l. bracket	[l. bracket	61	0133	8-7	12-8-2
] r. bracket	] r. bracket	] r. bracket	62	0135	0-8-2	11-8-2
% †	% †	% †	63 †	0045	8-6	0-8-4
: colon	: colon	: colon	63	0072	8-2	8-2
≠	" quote	" quote	64	0042	8-4	8-7
↱	_ underline	_ underline	65	0137	0-8-5	0-8-5
∨	!	!	66	0041	11-0	12-8-7
∧	& ampersand	& ampersand	67	0046	0-8-7	12
↑	' apostrophe	' apostrophe	70	0047	11-8-5	8-5
↓	?	?	71	0077	11-8-6	0-8-7
<	<	<	72	0074	12-0	12-8-4
>	>	>	73	0076	11-8-7	0-8-6
≤	@		74		8-5	8-4
≥	\ rev. slant	\ rev. slant	75	0134	12-8-5	0-8-2
¬	^ circumflex		76		12-8-6	11-8-7
; semicolon	; semicolon	; semicolon	77	0073	12-8-7	11-8-6
		@		0100		
		^ circumflex		0136		
		: colon †		0072		
		%		0045		
		` grave accent		0140		
		a		0141		
		b		0142		
		c		0143		
		d		0144		
		e		0145		
		f		0146		
		g		0147		

† The interpretation of this character or code may depend on its context. Refer to Character Set Anomalies elsewhere in this appendix.

00029
2 OF 3

TABLE A-2. BATCH CHARACTER SETS (Contd)

CDC Graphic (64 Char)	ASCII Graphic (64 Char)	ASCII Graphic (95 Char)	Display Code	12-Bit ASCII Code	Punch Code	
					026	029
		h		0150		
		i		0151		
		j		0152		
		k		0153		
		l		0154		
		m		0155		
		n		0156		
		o		0157		
		p		0160		
		q		0161		
		r		0162		
		s		0163		
		t		0164		
		u		0165		
		v		0166		
		w		0167		
		x		0170		
		y		0171		
		z		0172		
		{ left brace		0173		
		| vert. line		0174		
		} right brace		0175		
		~ tilde		0176		

00029
3 OF 3

TABLE A-3. ASCII NINE-TRACK CODED TAPE CONVERSION

ASCII				Display Code		ASCII				Display Code	
Code Conversion†		Character and Code Conversion††				Code Conversion†		Character and Code Conversion††			
Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)	Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)
20	space	00	NUL	space	55	3E	>	1E	RS	>	73
21	!	7D	}	!	66	3F	?	1F	US	?	71
22	"	02	STX	"	64	40	@	60	`	@	74
23	#	03	ETX	#	60	41	A	61	a	A	01
24	$	04	EOT	$	53	42	B	62	b	B	02
25	%	05	ENQ	%	63	43	C	63	c	C	03
25	%	05	ENQ	space†††	55	44	D	64	d	D	04
26	&	06	ACK	&	67	45	E	65	e	E	05
27	'	07	BEL	'	70	46	F	66	f	F	06
28	(	08	BS	(	51	47	G	67	g	G	07
29	)	09	HT	)	52	48	H	68	h	H	10
2A	*	0A	LF	*	47	49	I	69	i	I	11
2B	+	0B	VT	+	45	4A	J	6A	j	J	12
2C	,	0C	FF	,	56	4B	K	6B	k	K	13
2D	-	0D	CR	-	46	4C	L	6C	l	L	14
2E	.	0E	SO	.	57	4D	M	6D	m	M	15
2F	/	0F	SI	/	50	4E	N	6E	n	N	16
30	0	10	DLE	0	33	4F	O	6F	o	O	17
31	1	11	DC1	1	34	50	P	70	p	P	20
32	2	12	DC2	2	35	51	Q	71	q	Q	21
33	3	13	DC3	3	36	52	R	72	r	R	22
34	4	14	DC4	4	37	53	S	73	s	S	23
35	5	15	NAK	5	40	54	T	74	t	T	24
36	6	16	SYN	6	41	55	U	75	u	U	25
37	7	17	ETB	7	42	56	V	76	v	V	26
38	8	18	CAN	8	43	57	W	77	w	W	27
39	9	19	EM	9	44	58	X	78	x	X	30
3A	:	1A	SUB	:	00	59	Y	79	y	Y	31
Display code 00 is undefined at sites using the 63-character set.						5A	Z	7A	z	Z	32
3A	:	1A	SUB	:	63	5B	[	1C	FS	[	61
3B	;	1B	ESC	;	77	5C	\	7C	|	\	75
3C	<	7B	{	<	72	5D	]	01	SOH	]	62
3D	=	1D	GS	=	54	5E	^	7E	~	^	76
						5F	_	7F	DEL	_	65

† When these characters are copied from/to a tape, the characters remain the same and the code changes from/to ASCII to/from display code.

†† These characters do not exist in display code. Therefore, when the characters are copied from a tape, each ASCII character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 61_{16}, from tape, it writes an uppercase A, 01_8.

††† A display code space always translates to an ASCII space.

00030

TABLE A-4. EBCDIC NINE-TRACK CODED TAPE CONVERSION

EBCDIC				Display Code		EBCDIC				Display Code	
Code Conversion†		Character and Code Conversion††				Code Conversion†		Character and Code Conversion††			
Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)	Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)
40	space	00	NUL	space	55	C4	D	84	d	D	04
4A	¢	1C	IFS	[	61	C5	E	85	e	E	05
4B	.	0E	SO	.	57	C6	F	86	f	F	06
4C	<	C0	{	<	72	C7	G	87	g	G	07
4D	(	16	BS	(	51	C8	H	88	h	H	10
4E	+	0B	VT	+	45	C9	I	89	i	I	11
4F	|	D0	}	!	66	D1	J	91	j	J	12
50	&	2E	ACK	&	67	D2	K	92	k	K	13
5A	!	01	SOH	]	62	D3	L	93	l	L	14
5B	$	37	EOT	$	53	D4	M	94	m	M	15
5C	*	25	LF	*	47	D5	N	95	n	N	16
5D	)	05	HT	)	52	D6	O	96	o	O	17
5E	;	27	ESC	;	77	D7	P	97	p	P	20
5F	¬	A1	~	^	76	D8	Q	98	q	Q	21
60	-	0D	CR	-	46	D9	R	99	r	R	22
61	/	0F	SI	/	50	E0	\	6A	|	\	75
6B	,	0C	FF	,	56	E2	S	A2	s	S	23
6C	%	2D	ENQ	%	63	E3	T	A3	t	T	24
6C	**%**	**2D**	**ENQ**	**space†††**	**55**	E4	U	A4	u	U	25
6D	_	07	DEL	_	65	E5	V	A5	v	V	26
6E	>	1E	IRS	>	73	E6	W	A6	w	W	27
6F	?	1F	IUS	?	71	E7	X	A7	x	X	30
7A	:	3F	SUB	:	00	E8	Y	A8	y	Y	31
Display code 00 is undefined at sites using the 63-character set.						E9	Z	A9	z	Z	32
						F0	0	10	DLE	0	33
7A	**:**	**3F**	**SUB**	**:**	**63**	F1	1	11	DC1	1	34
7B	#	03	ETX	#	60	F2	2	12	DC2	2	35
7C	@	79	\	@	74	F3	3	13	TM	3	36
7D	'	2F	BEL	'	70	F4	4	3C	DC4	4	37
7E	=	1D	IGS	=	54	F5	5	3D	NAK	5	40
7F	"	02	STX	"	64	F6	6	32	SYN	6	41
C1	A	81	a	A	01	F7	7	26	ETB	7	42
C2	B	82	b	B	02	F8	8	18	CAN	8	43
C3	C	83	c	C	03	F9	9	19	EM	9	44

† When these characters are copied from/to a tape, the characters remain the same (except EBCDIC codes 4A, 4F, 5A, and 5F) and the code changes from/to EBCDIC to/from display code.

†† These characters do not exist in display code. Therefore, when the characters are copied from a tape, each EBCDIC character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 81_{16}, from tape, it writes an uppercase A, 01_8.

††† All EBCDIC codes not listed translate to display code 55_8 (space). A display code space always translates to an EBCDIC space.

00031

TABLE A-5. SEVEN-TRACK CODED TAPE CONVERSION

External BCD	ASCII Character	Octal Display Code	External BCD	ASCII Character	Octal Display Code
01	1	34	40	-	46
02	2	35	41	J	12
03	3	36	42	K	13
04	4	37	43	L	14
05	5	40	44	M	15
06	6	41	45	N	16
07	7	42	46	O	17
10	8	43	47	P	20
11	9	44	50	Q	21
12†	0	33	51	R	22
13	=	54	52	!	66
14	"	64	53	$	53
15	@	74	54	*	47
16†	%	63	55	'	70
17	[	61	56	?	71
20	space	55	57	>	73
21	/	50	60	+	45
22	S	23	61	A	01
23	T	24	62	B	02
24	U	25	63	C	03
25	V	26	64	D	04
26	W	27	65	E	05
27	X	30	66	F	06
30	Y	31	67	G	07
31	Z	32	70	H	10
32	]	62	71	I	11
33	,	56	72	<	72
34	(	51	73	.	57
35		65	74	)	52
36	#	60	75	\	75
37	&	67	76	^	76
			77	;	77

†As explained previously in this section, conversion of these codes depends on whether the tape is being read or written.

00032

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.

Alphanumeric

The letters of the alphabet (A-Z) and the digits (0-9).

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 CYBER 170, CYBER 70, and 6000 Series computers.

Control Points

The concept by which the multiprogramming capability of 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 and NOS/BE 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, FORTRAN 5, Sort/Merge 4, ALGOL 4,

and the DMS-170 products is implemented through CYBER Record Manager. Neither the input/output of the NOS/BE operating system nor any of the system utilities such as COPY or SKIPF is implemented through CYBER Record Manager. 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 reinitialization 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 Dym 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.

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. CDC 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.

Extended Core Storage (ECS)

ECS contains 60-bit words. ECS has a large amount of storage and fast transfer rates.

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 file name. All files have at least one partition, which is delimited by a system-logical-record of level 17_8 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 Name (lfn)

The 1-7 display coded alphabetic or numeric characters by which the operating system recognizes a file. Every lfn in a job must be unique and begin with a letter.

File Set

One or more related files recorded on one or more volumes.

Full track (FT)

Reading/writing sequential sectors on an 844 or 885 disk drive (1:1 interlace).

Half track (HT)

Reading/writing alternate sectors on an 844 or 885 disk drive (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 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.

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 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_8 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, a PRU 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 file name which causes the file to be punched on cards in Hollerith format when the job terminates.

PUNCHB

A 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

CDC CYBER Record Manager defines a record or a portion thereof as the smallest collection of information passed between CDC 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.

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 CDC 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 from 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.

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 CDC CYBER Record Manager S type record.

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 line printer, card punch, and card reader) runs under control of JANUS. A nonstandard unit record device (such as graphic consoles, plotters, and paper tape readers and punches) runs 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_8 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.

1xPPU

Memory speed is 1 microsecond.

2xPPU

Memory speed is 0.5 microseconds. 2xPPU is available only on CYBER 170. Memory speed is set by the installation.

PUNCH CARD AND TAPE FORMAT C

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

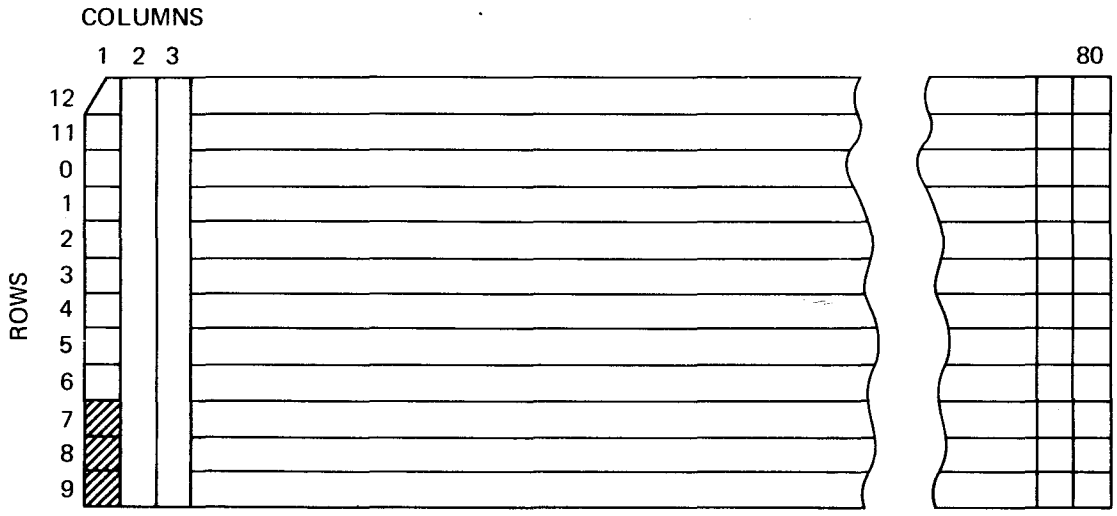
Hollerith cards are often called coded cards. Each column can be punched to represent codes of any given character set (see appendix A). The hole code is translated by card reading devices into the binary code for the character. Blank columns are translated into a binary code representing a blank space.

Hollerith punch cards can be in O26 or O29 format. O26 mode is a 63- or 64-character set defined by Control Data. O29 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 O26 and O29 modes and discusses how to change modes within a job deck.

END-OF-RECORD CARD

A card containing octal 0007 (7/8/9) in column 1 separates logical records 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:



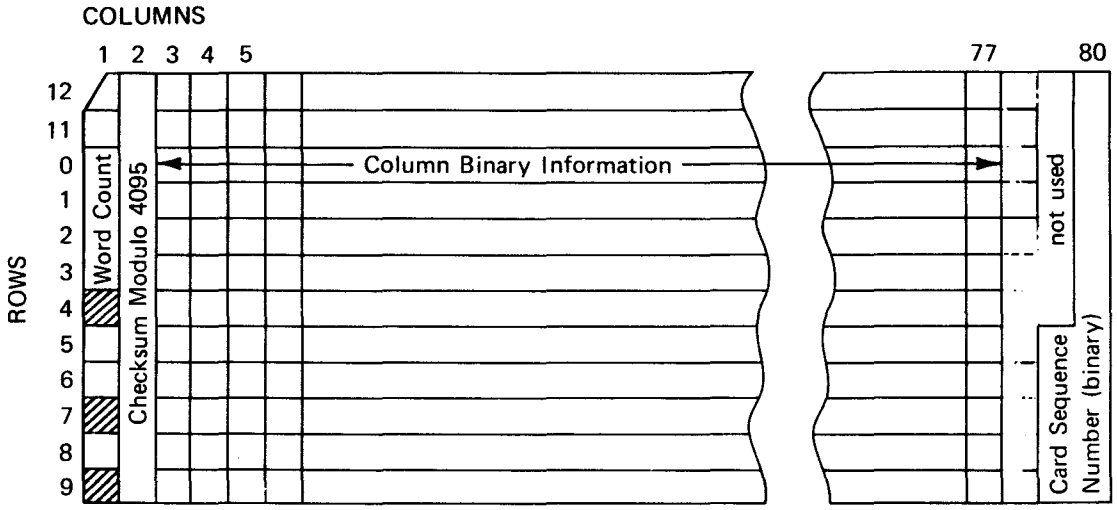
STANDARD BINARY CARDS

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 through 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 column 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.

A series of Hollerith cards.

A start free-form flag card (7777 in columns 1 and 80) with no other punches.

A series of free-form binary cards without a standard 6/7/8/9 card or any card identical with 2.

An end free-form flag card identical with 2.

A start free-form flag card, which might be the same as or different from 2 and 4.

A series of free-form binary cards as in 3.

An end free-form flag card identical with 5.

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.

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 63-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.

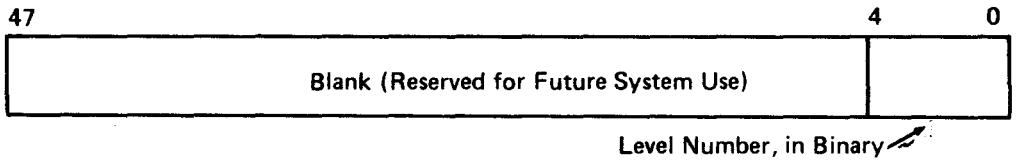
Peculiarities for coded tape for the 64-character set:

	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 63_8. 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:

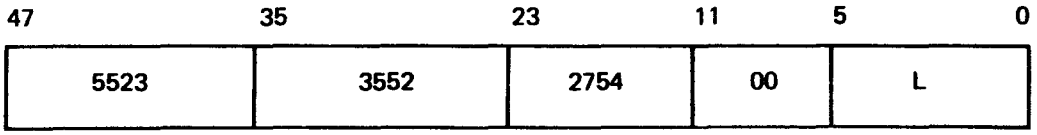


The level number is the low-order 5 bits of the last character. The upper 2 bits of this character are always zero except for level zero which is represented by 010000 (binary). For example, in external BCD, level 5 would be represented by 2020202020202005 and level 0 would be represented by 2020202020202020.

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. Only full central memory (CM) words can be written or read on SI format tapes. If a short or zero-length PRU is written on tape, it is terminated by a 48-bit system-logical-record terminator. Data is written on tape in frames of 8 bits, packed mode. For example, one CM word (60 bits) is written on tape as 7.5 frames, and two CM words are written as 15 frames. If a short PRU contains an odd number of CM words, the last 4 bits of the last frame are not used. 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 C-1 summarizes tape file characteristics.

TABLE C-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 cpi	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 that a 60-bit word is changed to or from 7.5 8-bit frames on tape.

† Short or zero length PRU with a 48-bit marker containing a level number ≤ 16B.

†† Zero length PRU with a 48-bit marker containing a level number of 17B.

††† For unlabeled tapes: on a write, 4 tape marks and 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.

CDC CYBER 170 MODEL 176 DIFFERENCES D

Major hardware differences between CDC CYBER 170 Model 176 and other CDC CYBER 170 models are as follows:

CDC CYBER 170 Model 176 extended memory is analogous to the CDC 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.

CDC CYBER 170 Model 176 is not compatible with the CDC 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.

CDC CYBER 70 Model 76 LCM/LCME memory replaces ECS as extended memory on CDC CYBER 170 Model 176. The maximum number of 60-bit words that can be transferred in the block copy instruction is 1023 decimal. CDC 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.

INTERPRETIVE MODE READING AND WRITING OF ECS E

Interpretive mode processing of ECS read and write operations gives the COMPASS programmer an effective means of breaking up large block ECS transfers and processing recoverable ECS errors. The efficiency of long ECS transfers tends to be degraded because of PP-initiated exchange jumps which force ECS transfers to be completely restarted. Interpretive mode processing breaks up large blocks into smaller, 400_8-word blocks, thereby minimizing the effects of these exchanges. ECS transfer errors are retried as a block transfer and then as single word transfers if necessary. If the error is recovered, it is logged in the system error log and is transparent to the user program. Unrecoverable errors are also logged and must be processed by the user program.

The interpretive routines are available on common deck COMCECS for absolute COMPASS programs and as relocatable routines in SYSLIB for relocatable COMPASS programs. Additionally, common deck COMCECM is provided to redefine the RE and WE COMPASS instructions. COMCECM is also available on systems text ECSTEXT. Thus, for absolute COMPASS programs, the user must either make specific calls to both common decks or call COMCECS and specify S=ECSTEXT on the COMPASS control statement (refer to the COMPASS Reference Manual). For relocatable COMPASS programs, the user need only specify ECSTEXT as an alternate systems text on the COMPASS statement.

Programs using interpretive mode reading and writing of ECS do so with the usual RE and WE COMPASS instructions. If, while in interpretive mode, the user desires to perform noninterpretive reading and writing of ECS, the RD and WT instructions must be used. These instructions are defined on common deck COMCECM. These instructions read and write ECS directly while in interpretive mode, as in the normal execution of the RE and WE instructions.

The instructions defined in ECSTEXT and common deck COMCECM are in the following formats.

```
inst    Bj
inst    K
inst    Bj+K
```

Instruction inst is one of the following:

inst	Description
RE	Read ECS in interpretive mode.
WE	Write ECS in interpretive mode.
RD	Read ECS noninterpretively in interpretive mode.
WT	Write ECS noninterpretively in interpretive mode.

TYPES AND NAMES OF RECORDS F

The type and the name of a record are determined by the COPYL and ITEMIZE utilities from information contained within the record. If the record begins with a prefix table, the record name is obtained from that table and the type of the record is determined from the first word following the prefix table. If the first word in the record is not a prefix table, but is a recognizable format, the format determines type. Any record that has neither a prefix table nor a recognizable format is classed as a DATA type record.

PREFIX TABLE USE

Prefix tables exist, unless they have been specifically suppressed, for programs assembled or compiled under any operating system and system text overlays.

The prefix table is the first of the ordered set of binary tables that form object programs. The tables consist of a header word with an octal table type identifier followed by varying amounts of control information that instruct system routines, such as the loader, or that contain the program code.

The prefix table is identified by octal digits 7700 in bits 48-59 of its first word; consequently, it is often referred to as a 77 or 7700 table. Information in the prefix table, which originates with the assembler or other system routine that creates the table, specifies items such as the date created and the system on which the job was executed.

Although some of the records may contain display coded data (loader directives, for instance, are coded), they are considered binary records.

OTHER RECORD IDENTIFIERS

If a prefix table is not present, the first word in a record is examined in a search for a recognizable format. If a record meets the criteria for a given type of record, the utilities identify it as such. An Update sequential program library, for example, is identified by the characters CHECK in the first word.

Table F-1 summarizes types of records and the criteria used to determine them.

RECORD NAMES

If a record begins with a prefix table, bits 18-59 of the second word of the table determine the record name. If a record does not begin with a prefix table, bits 18-59 of the first word of the record are used as the record name.

Records typed as DATA, UCF, and UPL do not have names.

TABLE F-1. DETERMINING TYPES OF RECORDS

Type of Record	Record Description	Type Determined By
ABS	Central processor overlay with one or more named entry points.	51 table; 53 table with bit 17=1; or 54 table with (0,0) overlays.
CAP	Capsule.	77 table followed by 6000 table.
DATA	Not any other described record type.	Unrecognizable by criteria defined in these tables.
LIBNT	Library name table record.	NOS/BE deadstart tape position.
OVL	Central processor overlay with one unnamed entry point (no ENTRY statement in program); system text.	50 table; 53 table with bit 17=0; or 54 table with non-(0,0) overlays.
PPNT	Peripheral processor program name table.	NOS/BE deadstart tape position.
PROC	CYBER control language procedure file.	.PROC followed by comma.
REL	Relocatable central processor program.	34 table.
SDR	Special deadstart record.	NOS/BE deadstart tape position.
TEXT	Text record.	No 77 table and first word has all zeros in bits 0-17.
UCF	Update compressed compile file.	77 table with 0 word count.
UPLx	Update sequential program library with x master control character.	No prefix table and characters CHECK in bits 30-59: control character obtained from bits 0-5.
6PP	6000 Series peripheral processor overlay.	77 table with three-character name in header word.
7PP	7000 Series peripheral processor overlay.	52 table.

INDEX

COMMENT SHEET

MANUAL TITLE: CDC NOS/BE Version 1 Reference Manual

PUBLICATION NO.: 60493800 **REVISION:** M

NAME: _____

COMPANY: _____

STREET ADDRESS: _____

CITY: _____ **STATE:** _____ **ZIP CODE:** _____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

☐ Please Reply ☐ No Reply Necessary

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD FOLD

MACRO INDEX

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION