CONTROL DATA
CORPORATION

CONTROL DATA®
CYBER 70/MODEL 76 COMPUTER SYSTEM
7600 COMPUTER SYSTEM

SCOPE 2
USER'S GUIDE

REVISION RECORD

REVISION	DESCRIPTION
A	Original printing.
(11-30-72)	
Publication No. 60372600	

Address comments concerning this manual to:

Control Data Corporation
Technical Publications Department
4201 North Lexington Avenue
Arden Hills, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

One of the most frequent criticisms of technical manuals is that the manner in which the material is presented is inconsistent with the needs of those who must use it. For some it is too technical; for others it is too basic. To avoid such misunderstandings, we should be sure that we agree on the intent of this guide.

WHO IS THE USER?

The user is an applications programmer/analyst who

- has experience in the use of FORTRAN or COBOL;
- has had little or no experience in the use of SCOPE 2;
- may have, but is not required to have, experience in the use of 6000 SCOPE.

WHAT IS A GUIDE?

- A guide is a document that goes beyond the presentations of bare facts in a reference manual.
- Its main purpose is to explain; that is, to examine system features, the reasons for their existence, and the conditions under which they are used.
- A guide makes extensive use of examples and illustrations.

WHAT DOES THIS GUIDE DO FOR YOU?

- It describes the operating environment at a SCOPE 2 installation.
- It contains guidelines that will help you to set up and run your programs.
- It defines system features that can make your programs more efficient.

Because this guide is intended for applications programmers, the features of SCOPE 2 that have meaning only to system analysts have been intentionally omitted. Those who want more detailed descriptions of the SCOPE 2 operating system and its related products may refer to the publications listed under Related Publications.

HOW TO USE THIS GUIDE

As a programmer/analyst, you should expect to obtain from this guide the information you need for defining your jobs and running them under SCOPE 2. To accomplish this goal, you should be aware not only of the subjects discussed, but also the format in which the material is presented.

For those who have no experience with SCOPE 2, the introductory material in section 1 provides a brief overview of the hardware and software components of the system and a brief description of how a job progresses through the system. Section 2 contains general information that applies to the structure of all jobs. Section 3 begins with material that describes how compilers are loaded and executed and how the user causes the programs generated by compilers to be loaded and executed. Following this, the user is lead into an analysis of the Loader and information telling how to control the loading process. Section 4 describes many of the options available to the user for controlling his job. If these options are not exercised, the system controls the job according to a set of default parameters. Sections 5 through 9 discuss the organization and transmission of data and the use of permanent and temporary files. Section 10 describes system utility operations such as copying, comparing, and positioning files. Section 11 contains a brief description of file label usage. Features of the system that aid in analyzing the program and debugging it are described in Section 12.

Before you continue, please note the presence of the comment and evaluation sheet at the end of this guide. We invite you to make specific comments and suggestions as you read the guide and to summarize your opinions when you have completed it. Your assessment of this material will help us to improve our guides and provide more of the information you need.

RELATED PUBLICATIONS

For readers who want a more detailed description of the SCOPE 2 system, information about topics not discussed in this guide or information about the members of the product set, a list of related publications follows.

Control Data Documents	Publication Number
CYBER 70/Model 76 Computer System Reference Manual	60367200
SCOPE 2 Reference Manual	60342600
SCOPE 2 Instant Manual	60344300
Record Manager Reference Manual	60307300
Loader Reference Manual	60344200
Loader Instant Manual	60372200
SCOPE 2 Diagnostic Handbook	60344100
SCOPE 2 Installation Handbook	60344000
6000 Series Station SCOPE 3.3 Operator's/Reference Manual	60360400
6000/CYBER 70 Series Station SCOPE 3.4 Operator's Reference Manual	60343800
6000/CYBER 70 Series Station Instant Manual	60360500
COMPASS Version 2 Reference Manual	60279900
COMPASS Version 2 Instant Manual	60282900
COMPASS Version 2 CPU Instruction Card	60279700
COMPASS Version 2 PPU Instruction Card	60279800
UPDATE Reference Manual	60342500
UPDATE Instant Manual	60360200
FORTRAN RUN Reference Manual	60305600
FORTRAN Extended Reference Manual	60384200

Control Data Documents	Publication Number
FORTRAN Extended Instant Manual	60327900
COBOL Reference Manual	60384200
SORT/MERGE Reference Manual	60343900
SORT/MERGE Instant Manual	60344400
Programming Aids	60158600

NOTICE

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 undefined parameters.

CONTENTS

APPENDIXES

FIGURES

EXAMPLES

TABLES

GENERAL DESCRIPTION 1

This section introduces the principles of the SCOPE 2 Operating System, gives a brief description of the hardware and software configurations, and describes how a job progresses through the system. In addition, it introduces the user to the topic of logical files, especially those files defined for every job by the system. Files are described in more detail in Section 5.

INTRODUCTION TO SCOPE 2

An operating system is a group of computer-resident programs or subprograms that monitors the input, compilation or assembly, loading, execution, and output of all other programs processed by the computer. The operating system that directs these operations for the CONTROL DATA® CYBER 70/Model 76 and the CONTROL DATA® 7600 Computer Systems is called SCOPE 2. SCOPE is an acronym for Supervisory Control Of Program Execution.

SCOPE 2 operates on a multiprogramming basis, using the versatility of the computer hardware to direct the simultaneous processing of programs. The maximum number of programs to be executed simultaneously is specified by the system operator. These programs can be written in compiler languages such as FORTRAN or COBOL, or in the COMPASS assembly language.

SCOPE 2 is a collection of programs that monitor and control compilation, assembly, loading, execution, and output of all programs that users submit to the computer. SCOPE 2 controls storage assignment tasks and the sequence in which jobs are processed. For each job, the system prints maps and dumps of memory to aid in debugging, detects errors, and prints diagnostic messages. It allows modification of stored programs through use of special editing routines. Jobs are submitted to SCOPE through operator stations. An operator station is a computer system that has been programmed to communicate with the SCOPE 2 system. SCOPE 2 returns job output and user-created files to the station at which the job originated.

SCOPE 2 features include:

- Multiprogramming of jobs throughout the system
- Record management that supports a variety of tape record and block formats
- Permanent file management to protect information from access by unauthorized persons
- A chronological record called a dayfile for each program run showing the results of each control statement processed and any problems encountered. This record is printed automatically when the program has been run.
- Communication with operator stations that submit job and data files to the SCOPE 2 operating system and receive the job output file and user-created output data files

OPERATING ENVIRONMENT

SCOPE 2 resides in a CONTROL DATA CYBER 70/Model 76 or 7600 Computer System. The hardware configuration of the computer system depends on the needs of the particular installation. The needs will also determine the product set in use. The following paragraphs briefly describe the components of the computer system and the possible hardware/software configurations that may exist under SCOPE 2.

HARDWARE CONFIGURATION

The CYBER 70/Model 76 and 7600 Computing Systems are large-scale, solid-state, general-purpose digital computing systems. The advanced designed techniques incorporated in this system provide for extremely fast and efficient solutions for large-scale, general-purpose processing.

A basic system (Figure 1-1) includes a central processor unit (CPU) and a number of peripheral processor units (PPU). Some of the PPUs are physically located with the CPU and others may be remotely located. The PPU provides a communication and message switching function between the CPU and individual peripheral equipment. Each PPU may have a number of high-speed data links to individual peripheral equipment as well as a data link to the CPU.

The data links may also be used to communicate with a variety of operator stations. Stations are self-contained processing systems that serve primarily as input/output processors for the CPU. The stations may connect through a first level PPU (FLPP) or directly to the CPU, as in the case of the 7611-1 I/O Station.

Central Processor Unit (CPU)

The CPU is a single integrated processing unit. It consists of a computation section, small core memory, large core memory, and an input/output multiplexer. The sections are all contained in the main frame cabinet and operate in a tightly synchronous mode. Communication outside the main frame cabinet is asynchronous.

Computation Section

The computation section of the CPU contains nine functional units and 24 operating registers. The units work together to execute a CPU program. Data moves into and out of the computation section of the CPU through the operating registers.

Core Memory

The CPU contains three types of internal memory arranged in a hierarchy of speed and size.

1. The instruction stack contains 12 60-bit words for issuing of instructions. The stack holds the latest ten instruction words and the two-instruction word look-ahead. Program loops can be held in the stack thereby avoiding memory references.

2. The small core memory (SCM) contains 32,768 or 65,536 60-bit words. Each word holds ten 6-bit characters.

3. The large core memory (LCM) contains 256,000 or 512,000 60-bit words. Instructions cannot be executed directly from LCM.

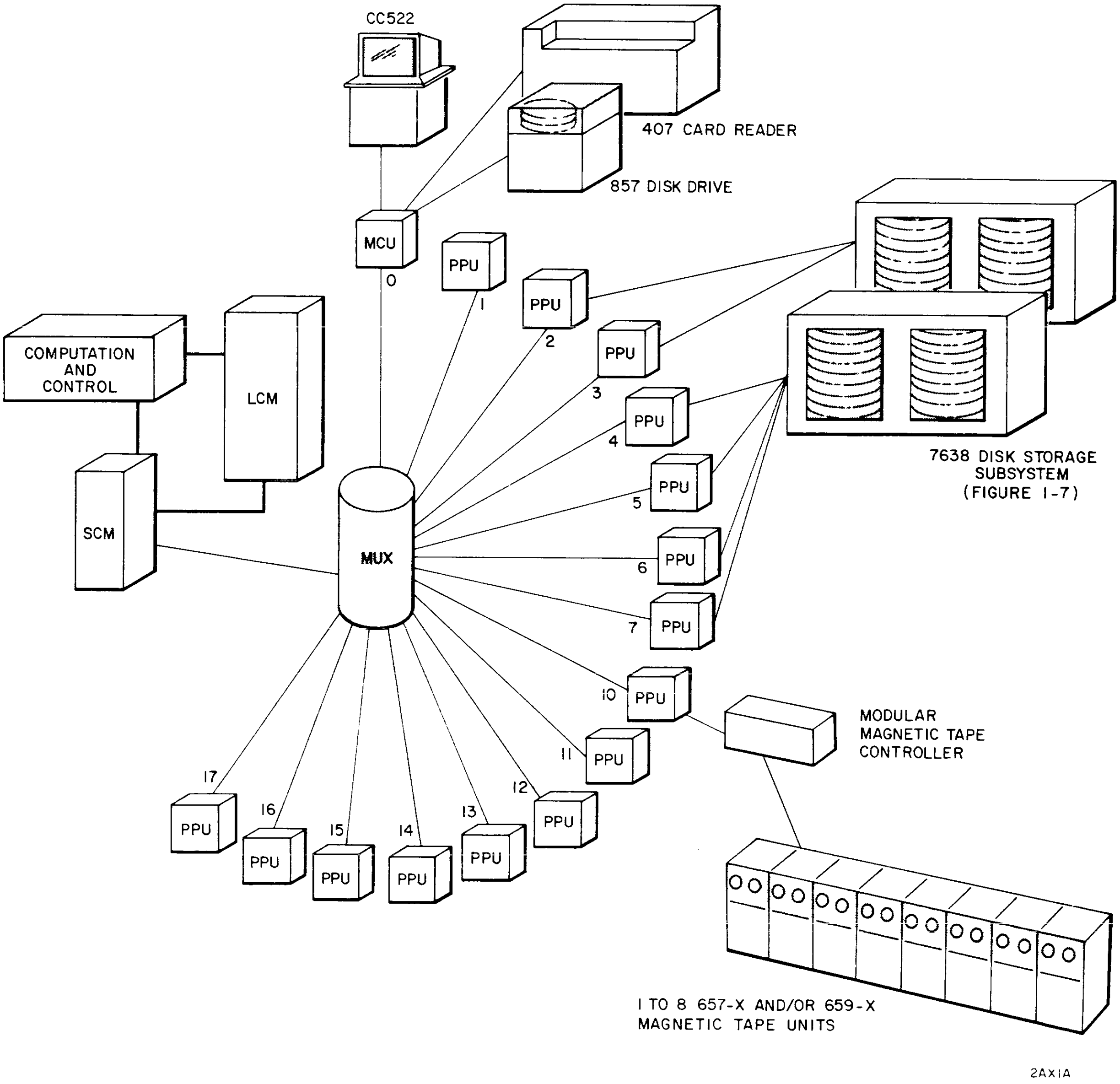


Figure 1-1. Basic System Configuration

The SCM performs certain functions in system operation that the LCM cannot effectively perform. These functions are essentially those requiring rapid random access to unrelated fields of data. The first 4K (octal) addresses in SCM are reserved for the input/output control and data transfer to service the communication channels to the PPU. CPU object programs do not have access to these areas. The remainder of the SCM may be divided between fields of CPU program code and fields of data for the currently executing program. A small portion contains a resident monitor program.

Input/Output Multiplexer

The function of the CPU input/output multiplexer (MUX) is to deliver 60-bit words to SCM for incoming data, to read 60-bit words from SCM for outgoing data, and to interrupt the CPU program for monitor action on the buffer data. In the basic system, the MUX includes eight 12-bit bidirectional channels of which one is reserved for use by the maintenance control unit (MCU).

Peripheral Processor Unit (PPU)

The peripheral processor units (PPU) are separate and independent computers, some of which may reside in the main frame cabinet. Others may be remotely located. A PPU may be connected to the MUX, another PPU, a peripheral device, a controller, or a mix of these. PPUs that connect directly to the MUX, whether on the main frame or remotely located, are termed first level PPUs (FLPP). PPUs drive many types of peripherals without the need of an intermediate controller.

Maintenance Control Unit (MCU)

The maintenance control unit (MCU) is a main frame PPU with specially connected I/O channels. The MCU performs system initialization and basic recovery for the system. It also serves as a maintenance station for directing and monitoring system maintenance activity.

System Mass Storage

System mass storage consists of one or two CONTROL DATA® 7638 Disk Storage Subsystems. Each 7638 Disk Storage Subsystem consists of two controllers and one disk file. Either one or two 7600 first level PPUs can be connected to each controller. The approximate data capacity of the system is 845 million 6-bit characters.

Stations

Several varieties of operator stations may be connected to the CYBER 70/Model 76 or 7600. Each station operates under control of its own operating system and is responsible for supporting unit-record, magnetic tape, or communication equipment.

On-Line Magnetic Tape Units

A configuration optionally includes one to eight CONTROL DATA® 657-X or 659-X Magnetic Tape Units driven by a controller directly connected to one or two first level PPUs.

On-line magnetic tape units are accessed through record manager requests. Information is transferred directly to and from the on-line unit without the intermediate transfer to mass storage that takes place for staged magnetic tapes at stations.

SOFTWARE CONFIGURATION

Software can be considered as consisting of an operating system (SCOPE 2) and a product set that perform as a matched set to translate the user's request into instructions to the hardware.

The product set complements the SCOPE 2 operating system to meet the user's requirements for scientific applications and commercial data management.

The SCOPE 2 product set includes:

COMPASS Assembler

FORTRAN (RUN) Compiler

FORTRAN Extended (FTN) Compiler

COBOL Compiler

Sort/Merge (SORTMRG) Program

UPDATE Library Maintenance Program

7611-11 Service Station Operating System

6000/CYBER 70 Series Station Operating System

7611-1 Input/Output Station Operating System

7611-2 Magnetic Tape Station Operating System

SCOPE 2 consists of a group of program modules. Three of these modules are particularly significant to the user. These are the loader, the segment loader, and the record manager.

Loader

The SCOPE 2 loader loads core image (absolute) modules and object (relocatable) modules in response to calls from the system and from users. Modules, that is, subprograms and data, can be loaded into user SCM and LCM from system and user libraries and from files attached to the job. Programs can be called according to program name, file name, or entry point name. External references made in an object module are satisfied from system or user libraries. A reference to an external symbol causes the module containing the symbol as an entry point to be loaded and linked to the module containing the reference.

Usually, the loader increases or decreases the amount of SCM and LCM available to the user (field lengths) according to the requirements of the program being loaded. This automatic core allocation can be overridden if the user desires.

A number of loader options permits the user to request load maps, presetting of core, execution or no execution following the load, libraries to be used for satisfying externals, etc.

The loader executes in the user field length. Programs that exceed available core memory storage can be loaded by organizing them into subdivisions that can be called, executed, and unloaded through the use of overlays. This process of overlaying is controlled by the user.

Segment Loader (SEGLOAD)

Programs that exceed available core memory storage can be loaded and executed by organizing them into subdivisions called segments. The user controls the segmentation of a program through directives issued to the segment loader (SEGLOAD).

A segmented load is more elaborate than an overlay load. SEGLOAD has the following features.

- A segment can have more than one entry point.
- Segment loads are implicit. Execution of an instruction that refers to an entry point in a currently nonloaded segment automatically results in calling the SEGLOAD resident program (SEGRES) which assumes control of loading of segments.
- A segment load can involve more than one level. This feature allows gaps in memory between segments that are logically connected.
- Calls for the SEGLOAD loader can be made through the control statement only.

Record Manager

The record manager acts as an interface between the user logical input/output functions and SCOPE physical input/output functions. It performs the following functions for the user.

- Recognizes nine record types, four block types, and three file organizations (file access methods)
- Detects the following conditions on a file according to file format:

  beginning of information

  end-of-record

  end-of-section

  end-of-partition

  end-of-information
- Blocks and deblocks records
- Passes data in the form of partial or full records between the user buffer and system I/O buffers in LCM
- Controls the transfer of large blocks of data from the system devices (mass storage and on-line tapes) to LCM and vice versa
- Manipulates tape labels
- Detects errors in format

The record manager does not execute in the user field length; it executes in the SCOPE 2 job supervisor area.

COMPASS Assembler

The COMPASS assembler language allows the user to express all hardware functions of the CYBER 70/Model 76 central and peripheral processors symbolically.

COMPASS produces a binary file used by the loader to establish an executable program. Pseudo instructions provide the user with a variety of options for generating macro instructions and obtaining a listing of source statements and object code ranging from brief to detailed. Errors detected by the assembler are noted.

FORTRAN Compilers

Two compilers are offered for the CYBER 70/Model 76: FORTRAN (RUN) and FORTRAN Extended (FTN).

FORTRAN (RUN) gives mixed mode arithmetic, masking (Boolean), logical and relational operators, shorthand notation for logical operators and constants, expressions as subscripts, variable dimensions, and variable format capability.

FORTRAN also gives conversion formats for all data forms, array references with fewer subscripts than dimensioned, Hollerith constants in arithmetic or relational expressions, and left- or right-justified Hollerith constants. The record manager provides access to files generated by other programming languages.

In addition to the above features, FORTRAN Extended supports random access data files and provides ANSI diagnostics and a debug facility. It also provides three levels of optimization selected by the user: standard compilation, fast compilation, and fast execution.

COBOL Compiler

The COBOL compiler combines with SCOPE and the record manager to simplify the programming of business data processing problems. The COBOL compiler produces easily modifiable source programs decreasing program development time and program conversion costs.

COBOL for the CYBER 70/Model 76 provides full ANSI compatibility through level 3 sort files in conjunction with the sort/merge program. A report writer facilitates flexible formats for printed reports. Segmentation and overlay of the object program extend the flexibility of the COBOL compiler.

SORT/MERGE Program

The sort/merge applications program accepts input from magnetic tape or disk and constructs sorted output to user specification on tape or disk. Sort/merge executes under control of SCOPE in a multiprogramming environment. The user may call the flexible set of routines by a macro instruction in the user's subprogram, by a control statement in the main program, or by the COBOL SORT verb.

UPDATE Program

UPDATE provides a means of maintaining Hollerith card images in conveniently updatable compressed format. The user converts a collection of decks, for example FORTRAN or COMPASS source decks, into a file called the program library. Each card in each deck is assigned an identifier when it is placed on the program library. Later, the user can reference any card for inserting, deleting, or replacing lines of code in the program. After the program is corrected, the new version of the program can be passed to a compiler or assembler for processing. Corrections can be temporary for the purpose of testing new codes or can be permanent modifications to the program library.

7611-11 Service Station Operating System

The 7611-11 Operating System executes on a 7611-11 Service Station (Figure 1-2). It services the 7600 or CYBER 70/Model 76 Computer System by sending job decks to it for batch processing and by processing output files from jobs. Jobs can originate at a card reader at the 7611-11 or at local and remote batch terminals. Two 7611-11 stations can be connected to a first level PPU. Features of the 7611-11 Service Station Operating System are that it:

- Links the SCOPE 2 Operating System with unit record equipment
- Multiplexes data flow
- Serves as local batch entry station for SCOPE 2
- Provides remote batch capability for SCOPE 2

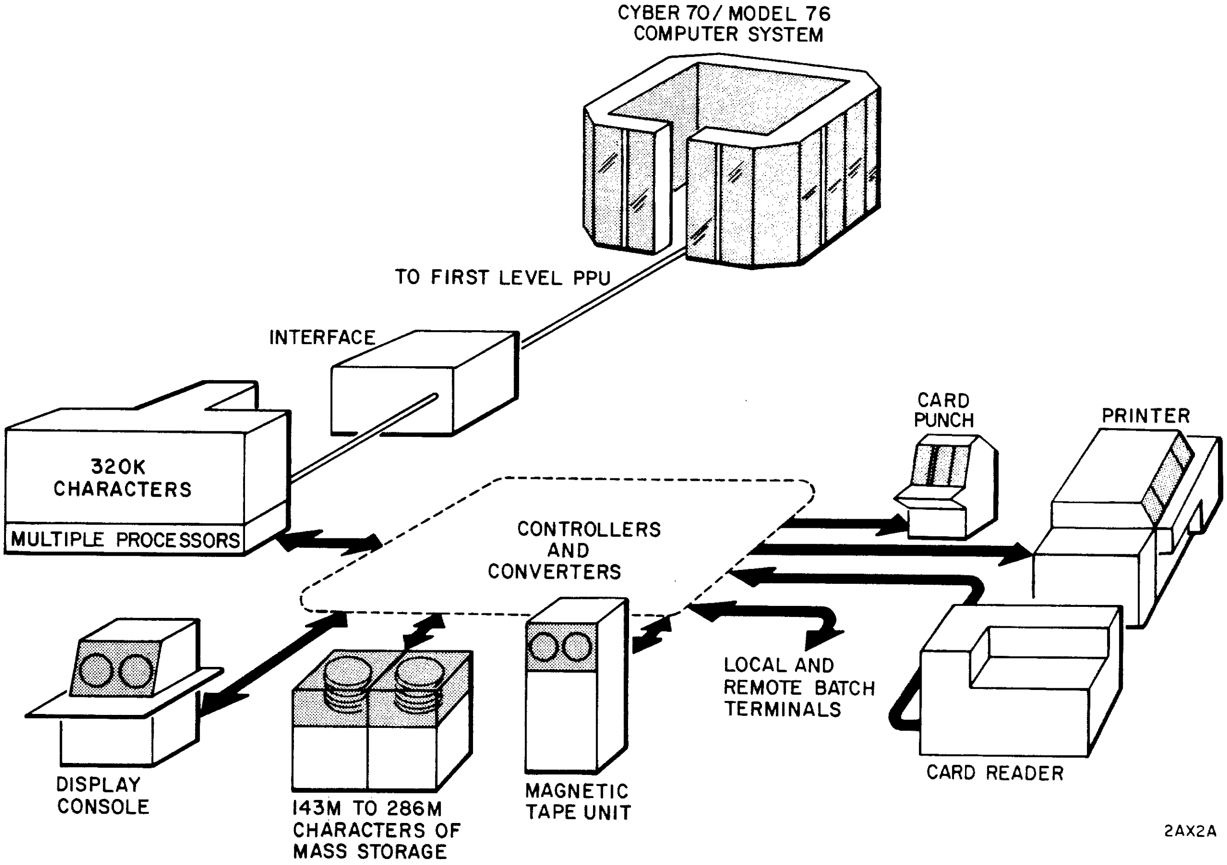


Figure 1-2. 7611-11 Service Station Configuration

6000/CYBER 70 Station Operating System

A site that has both a 7600 Computer System (or CYBER 70/Model 76 Computer System) and a 6000 Series Computer System (or a CYBER 70/Model 72, 73, or 74 Computer System) may choose to link the two systems. The 6000 or CYBER 70/Model 72, 73, or 74 serves as a batch entry station and provides the peripheral processing for the 7600 or CYBER 70/Model 76. The station executes under control of either the SCOPE 3.3 or SCOPE 3.4 operating system (Figure 1-3). It provides the operator with a command language paralleling that available on the 6000 SCOPE operating system. Job decks entered at the 6000 station card reader are routed either to the 7600 or the 6000 Series computer through a parameter supplied by the user on the first card of the job deck (the job identifier card). Each job sent to the 7600 is tagged with its station origin so that instructions for the operator and output from the job can be routed back to the station that originated the job. Two 6000/CYBER 70 stations can be connected to a first level PPU.

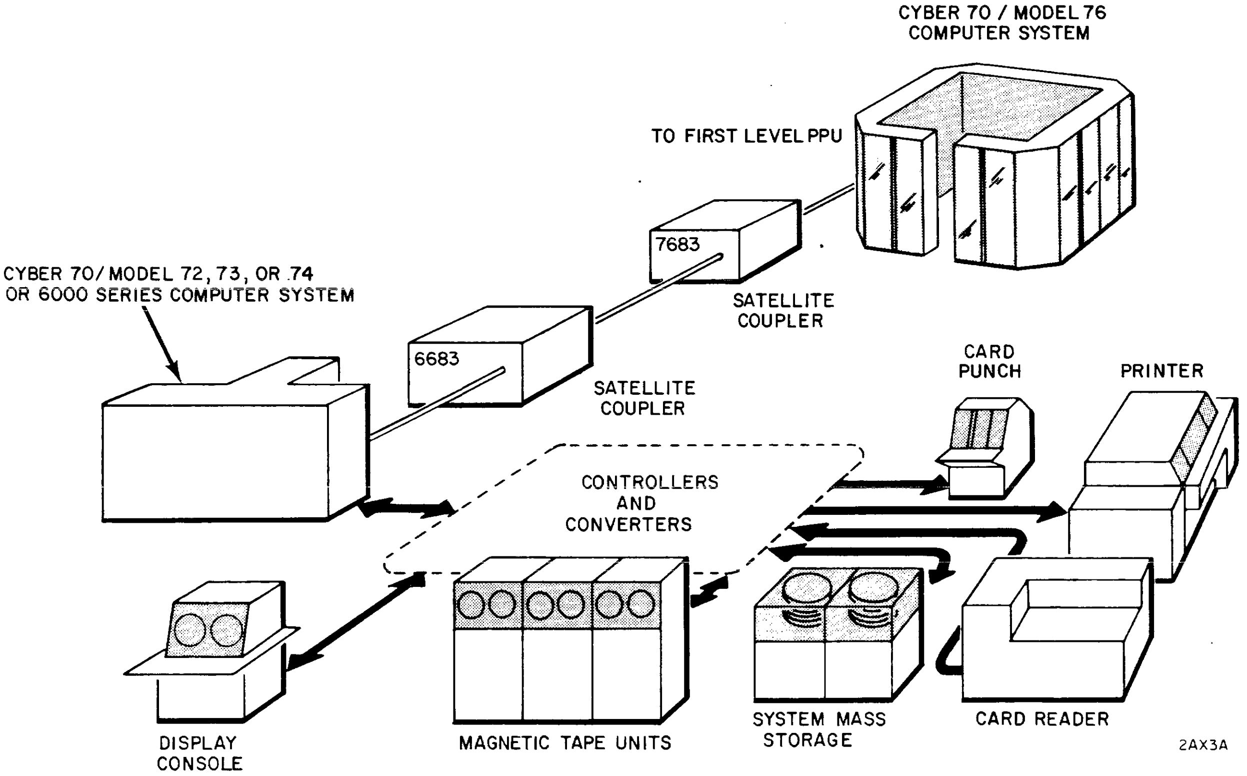


Figure 1-3. 6000 or CYBER 70 Series Station Configuration

7611-1 Input/Output Station Operating System

The 7611-1 I/O Station Operating System services the 7600 or CYBER 70/Model 76 Computer System by sending job decks to it for batch processing and by processing output files from jobs. The I/O station operating system resides in all six PPUs that comprise the 7611-1 I/O Station (Figure 1-4). The normal mode of operation is for the I/O station to be coupled to a channel of the I/O multiplexer and to be in communication with the SCOPE 2 operating system.

Features of the 7611-1 I/O Station Operating System include:

- Isolation of the high-speed CPU from low-speed peripheral devices by transferring all files from the station to system mass storage so that efficient CPU utilization can be realized
- Automatic routing of input files to the CPU and processing of output files from the CPU in a manner that optimizes the use of peripheral equipment
- Accounting information maintained for each job in a system dayfile
- Operator control of utility operations independent of SCOPE 2

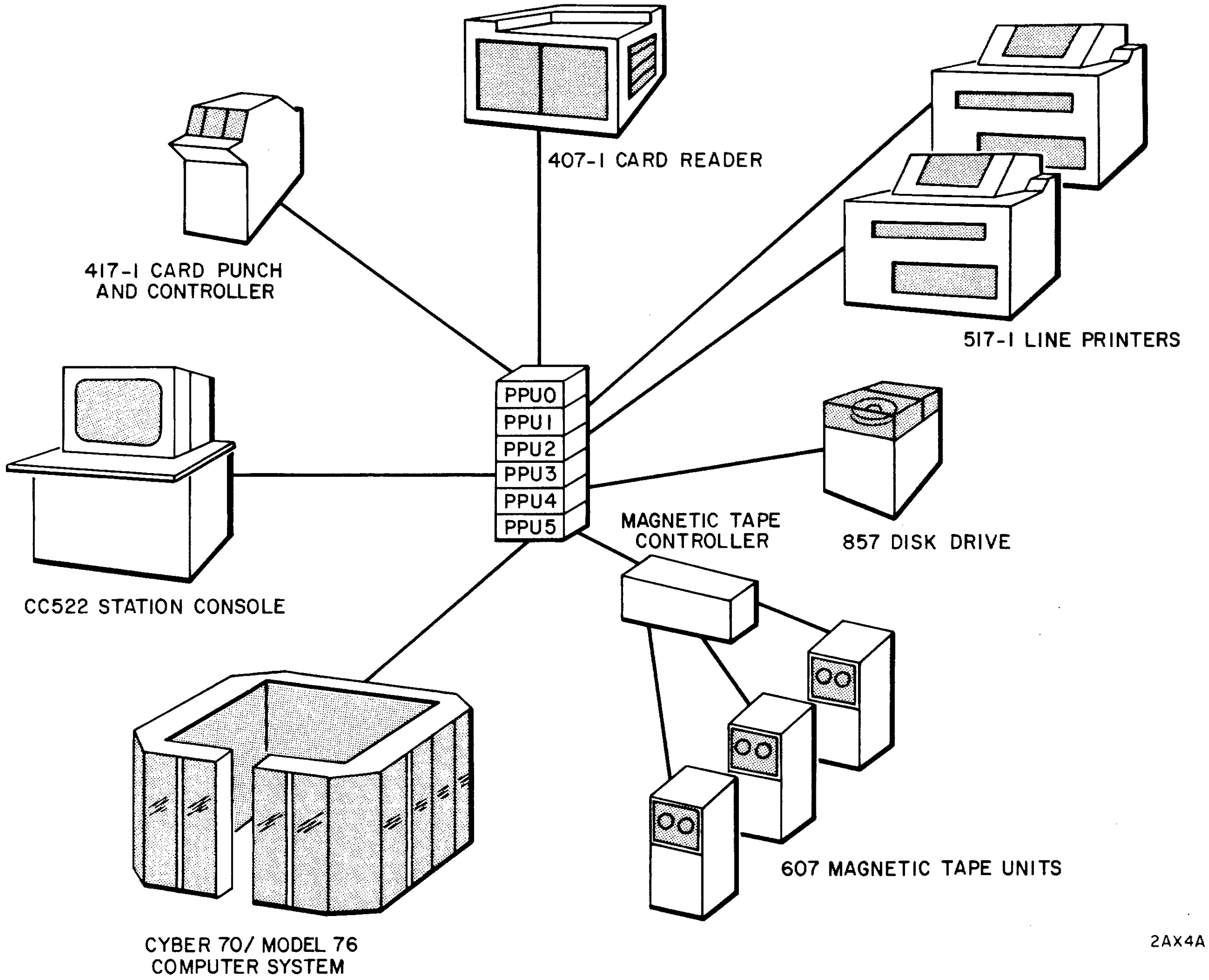


Figure 1-4. 7611-1 I/O Station Configuration

7611-2 Magnetic Tape Station Operating System

The 7611-2 Magnetic Tape Station (Figure 1-5) has the sole purpose of providing efficient tape staging for SCOPE 2. Staging is the process of transferring magnetic tape files to or from system mass storage so that efficient CPU utilization can be realized.

The normal mode of operation is for the 7600 or CYBER 70/Model 76 Computer System to be linked to at least one batch entry station in addition to the 7611-2 Magnetic Tape Station. Jobs submitted through the batch entry station request tape staging through SCOPE 2 control statements. The operator at the 7611-2 Magnetic Tape Station responds to the requests by mounting or dismounting input and output tapes from the units at the station.

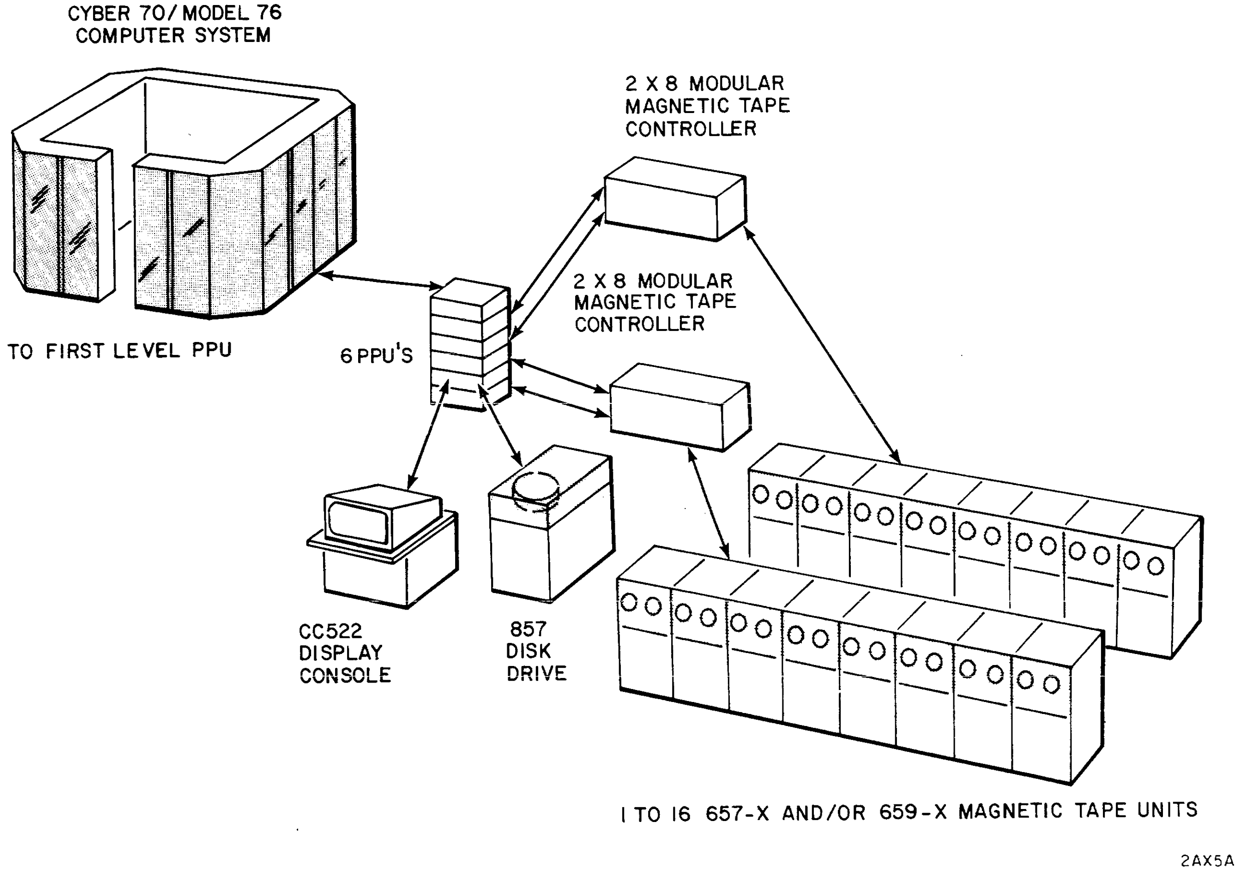


Figure 1-5. 7611-2 Magnetic Tape Station Configuration

JOB FLOW

A job enters the system in the form of a job deck submitted at a local or a remote station. From the station, it is transmitted to the CYBER 70/Model 76 where the job resides in the job input queue. Information concerning station messages is inserted into the job's dayfile. From the job input queue, the job proceeds in three phases: job initiation, job processing, and job termination. Figure 1-6 shows a generalized diagram of job flow.

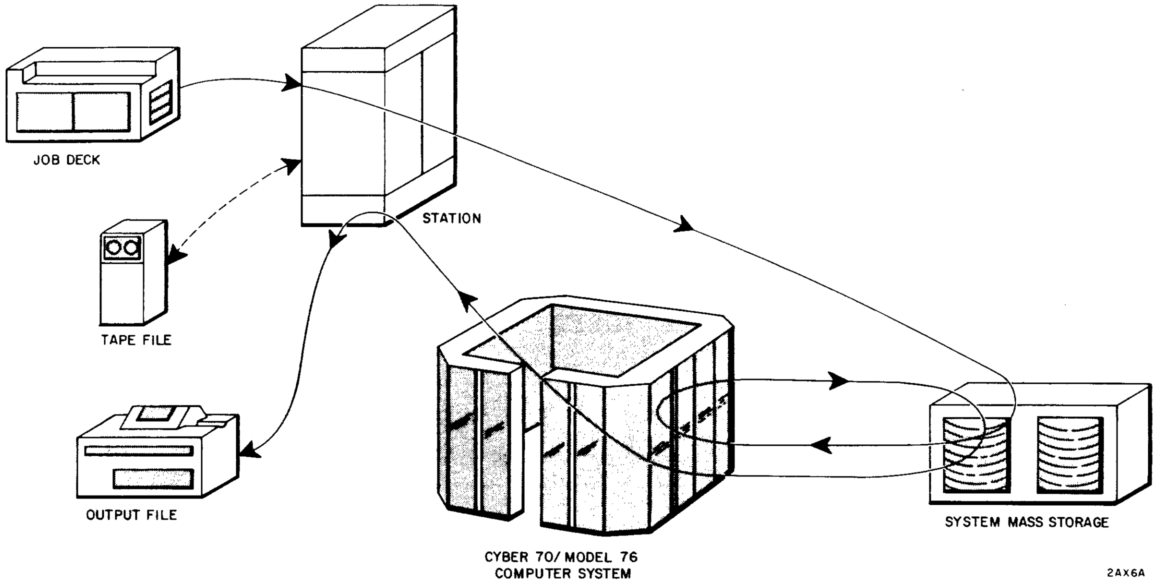


Figure 1-6. Job Flow Through System

JOB INITIATION

The operating system examines the job statement parameters to determine whether any dependencies exist between the job and any other jobs in the system, and to determine the resources needed. The job remains in the job input queue until dependencies are satisfied, until system resources (for example, the number of on-line tape drives) required for initiation are available. The algorithm used for scheduling on-line tape drives eliminates the possibility of system deadlock when the job is executing even though the total number of units required is not available.

SCOPE 2 divides the job deck into two files (Figure 1-7): the control statement section becomes the job control file; the remainder of the deck becomes a file named input.

Initiation of a job includes: preparing a job-related system area and positioning the job control and input files for the first job step, constructing an SCM image in LCM, and placing the job in a waiting queue for the CPU.

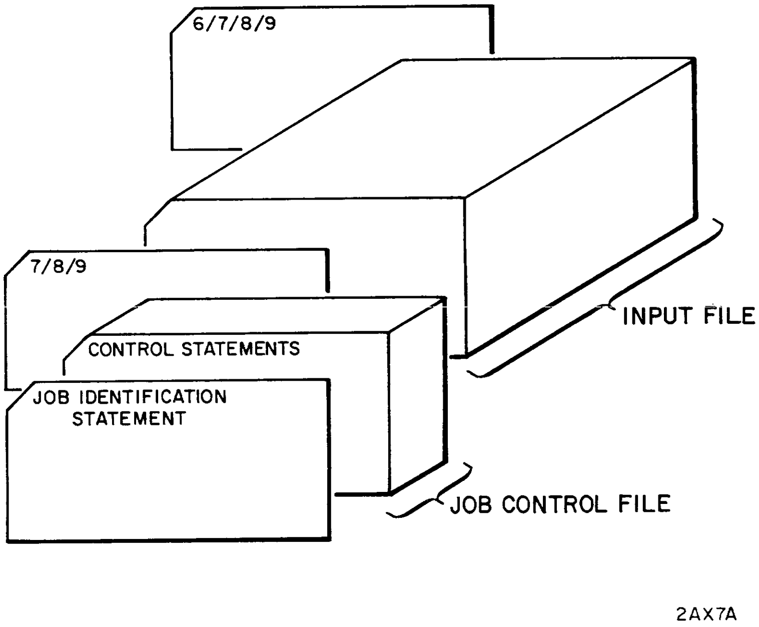


Figure 1-7. Job Deck Translation

JOB PROCESSING

As a job advances from step to step, the operating system reads and interprets control statements in the job control file. It assigns resources as required. While waiting (for example, for assignment of resources, operator intervention, staging of files) a job's residence can change from core (SCM or LCM) to system mass storage in response to the operating system's evaluation of the job's needs. For example, if I/O access requires the user LCM area or LCM system buffers, then the operating system furnishes residence for the job in LCM. The operating system changes residence of a job as needed by the job and as determined by overall considerations of scheduling the CPU. Job scheduling establishes the job's residence progressively through three storage levels: mass storage, LCM, and SCM. Jobs receive an aging increment while waiting in mass storage and in LCM to ensure that every job is given a chance to execute in SCM. Job field length requirements are evaluated against available memory to maximize use of large and small core memory. The CPU scheduling process selects a job to which the CPU is assigned and controls memory so that the jobs selected can be brought into SCM.

The system under the control of installation selected time limits and options ages the priority of a job to ensure every job's opportunity to execute.

A history of the job is maintained in a dayfile for the job. At job end, the dayfile is written in a file named OUTPUT. OUTPUT is also the default name of a file used for list output by compilers and assemblers. OUTPUT is printed automatically at the station of job origin.

JOB TERMINATION

At normal or abnormal completion of a job, the operating system returns all resources stil, assigned to the job to the system for rescheduling. These resources include online tape drives, user field length areas of SCM and LCM, job-related system areas in SCM and LCM, and files not yet unloaded.

THE JOB DAYFILE

The dayfile is the short list of comments at the end of the output for a job. It presents an abbreviated history of the progress of the job through the system. Each control statement is listed in sequence followed by messages associated with the job step. If a job is rerun, the control statements processed prior to the rerun are listed without times. The list is terminated by the message JOB RERUN.

A dayfile usually consists of the following items as illustrated in Figure 1-8.

1. First header line: identifies operating system, its current modification level, and the date the job was run in two forms. The first form is either month, day, year, or day, month, year, depending on an installation option. The second form is Julian notation.

2. Second header line: identifies site and contains information determined by an installation parameter.

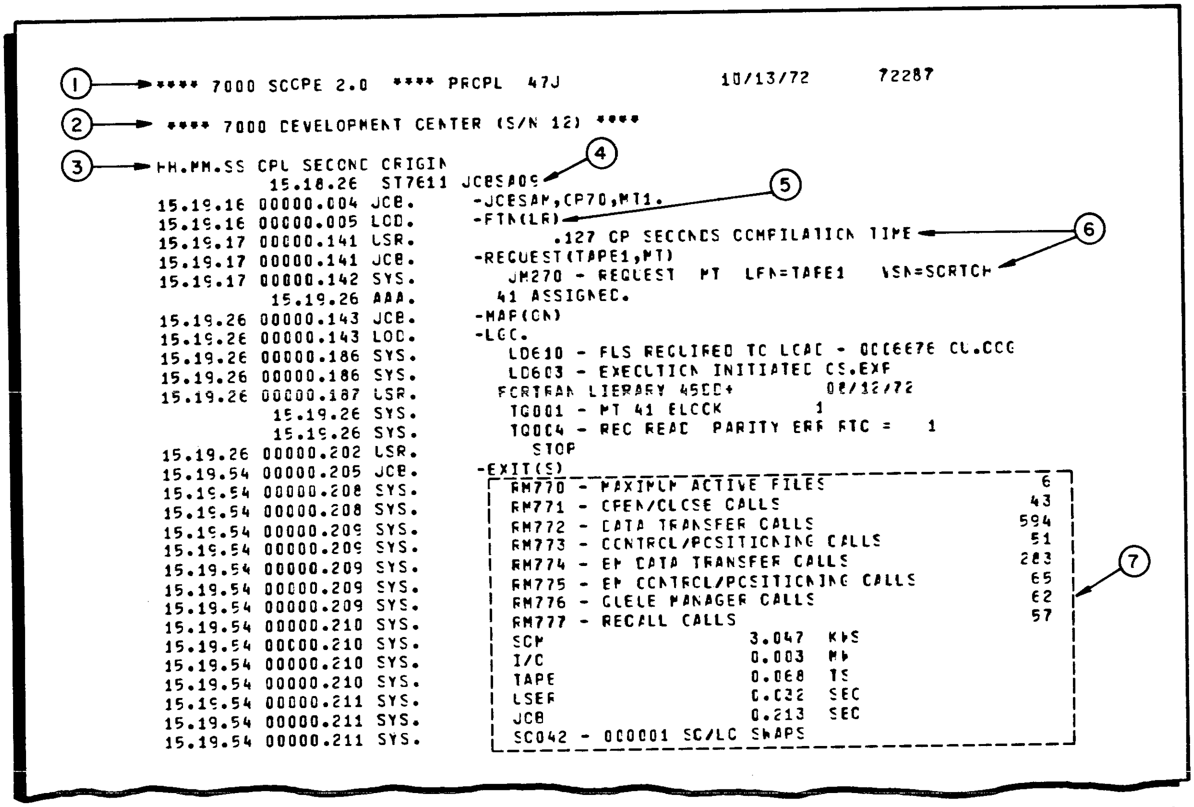
3. Column heads: the leftmost column (up to the first asterisk) identifies the clock time for each job step, the middle column (between the two asterisks) identifies the accumulated CPU time for the job. For some job manager messages there is no CPU time for the job step, and the clock time is in the middle column rather than the left column. The rightmost column identifies the system module that used the CPU time, or if execution is in the user field notes USR. Entries commonly noted are the following.

SYS	System (I/O requests, etc)
USR	User program, including compilers and assembler time
LOD	Loader
JOB	Control statement processing
ggg	Station processing (ggg is station identifier; for example, CCP may indicate the 6000 station)

4. Station subheading: gives the time the job was submitted at the station, the station identification, and the fabricated job name. This line varies according to the station that submitted the job.

5. Control statement: the first statement is always the job identification statement; the last control statement listed is the last one processed by the job. Each control statement is listed in sequence, prefixed by a hyphen. If the job abnormally terminates, not all of the control statements will be listed.

6. Dayfile messages: Any messages related to the control statement processing are indented below the statement. These messages are listed in detail in the SCOPE 2 Diagnostic Handbook.



1
2
3
4
5
6
7
2AX60A

```
**** 7000 SCOPE 2.0  **** PROPL  47J             10/13/72       72287
 **** 7000 DEVELOPMENT CENTER (S/N 12) ****
HH.MM.SS CPU SECOND ORIGIN
         15.18.26  ST7611 JOBSA09
15.19.16 00000.004 JOB.     -JOBSAM,CP70,MT1.
15.19.16 00000.005 LOD.     -FTN(LR)
15.19.17 00000.141 USR.          .127 CP SECONDS COMPILATION TIME
15.19.17 00000.141 JOB.     -REQUEST(TAPE1,MT)
15.19.17 00000.142 SYS.       JM270 - REQUEST  MT  LFN=TAPE1   VSN=SCRTCH
         15.19.26 AAA.       41 ASSIGNED.
15.19.26 00000.143 JOB.     -MAP(ON)
15.19.26 00000.143 LOD.     -LGO.
15.19.26 00000.186 SYS.       LD610 - FLS REQUIRED TO LOAD - 006676 CM.000
15.19.26 00000.186 SYS.       LD603 - EXECUTION INITIATED CS.EXF
15.19.26 00000.187 USR.      FORTRAN LIBRARY 450C+         08/12/72
         15.19.26 SYS.        TG001 - MT 41 BLOCK         1
         15.19.26 SYS.        TG004 - REC READ  PARITY ERR RTC =    1
15.19.26 00000.202 USR.         STOP
15.19.54 00000.205 JOB.     -EXIT(S)
15.19.54 00000.208 SYS.      RM770 - MAXIMUM ACTIVE FILES                  6
15.19.54 00000.208 SYS.      RM771 - OPEN/CLOSE CALLS                     43
15.19.54 00000.209 SYS.      RM772 - DATA TRANSFER CALLS                 594
15.19.54 00000.209 SYS.      RM773 - CONTROL/POSITIONING CALLS            51
15.19.54 00000.209 SYS.      RM774 - BM DATA TRANSFER CALLS              283
15.19.54 00000.209 SYS.      RM775 - BM CONTROL/POSITIONING CALLS         65
15.19.54 00000.209 SYS.      RM776 - QUEUE MANAGER CALLS                  62
15.19.54 00000.210 SYS.      RM777 - RECALL CALLS                         57
15.19.54 00000.210 SYS.      SCM                      3.047  KWS
15.19.54 00000.210 SYS.      I/O                      0.003  MW
15.19.54 00000.210 SYS.      TAPE                     0.068  TS
15.19.54 00000.211 SYS.      USER                     0.022  SEC
15.19.54 00000.211 SYS.      JOB                      0.213  SEC
15.19.54 00000.211 SYS.      SC042 - 000001 SC/LC SWAPS
```

Figure 1-8. Sample Dayfile Listing

Accounting Information

When a job reaches completion, SCOPE writes a summary of basic accounting data onto the dayfile for the job (Figure 1-8). Accounting information consists of the following data in decimal. Items marked by ** are not included when an installation parameter is set to inhibit them. An entry is omitted if it is zero or is irrelevant for the job (for example, the average for on-line tape units appears only for jobs using on-line tapes).

** 1. Maximum number of files active at any one time during the job. This number may be higher than expected because it always includes the files that the system creates for the job: INPUT, OUTPUT, job control file, and the dayfile.

** 2. File open and close requests

** 3. Data transfer requests (COMPASS GET/PUT and READ/WRITE macro calls)

** 4. File control and/or positioning requests (BKSP, SKIPF, etc)

** 5. Record manager/buffer manager data transfer requests for next buffer

** 6. Record manager/buffer manager control and/or positioning requests when no data is in buffer

** 7. Mass storage requests to queue manager by record manager

** 8. I/O recall requests; number of times job waits for I/O

** 9. SCM used expressed in kiloword seconds. Each kiloword second means that the job used a thousand words for a second.

10. LCM expressed in kiloword seconds. Each kiloword second means that the job used a thousand words for a second. This does not include LCM system I/O buffers.

11. Number of I/O words transferred by SCOPE for the job in millions of words

** 12. Mass storage used expressed in megaword seconds

** 13. On-line tape unit usage expressed as tape seconds, that is, each second represents the CPU time for which the job had possession of an on-line tape unit.

14. User execution time, that is, the CPU time used for executing programs in the SCM field. This value is expressed to the nearest millisecond.

15. CPU time used by the job expressed to the nearest millisecond. This value includes system overhead and user execution time.

16. The number of times the job was transferred (swapped) between SCM and LCM

INTRODUCTION TO LOGICAL FILES

A SCOPE 2 logical file is a collection of information identified by a name and delimited by a beginning of information and an end of information. SCOPE 2 regards all groups of information in the system as files and is, therefore, said to be a file-oriented system. Files directly accessible to the computer system can reside on system mass storage and on on-line magnetic tape units. Other files can exist in the form of punched card decks or magnetic tapes when they are entered into the system and copied onto mass storage. Data on files leaving the system can be written on magnetic tape, punched on cards, or printed.

Users accustomed to 6000 SCOPE must realize that many of the file terms and concepts with which they are familiar do not apply for SCOPE 2. For example, SCOPE 2 has no parallel to the physical record unit (PRU).

Information on files is divided into units of data called logical records. SCOPE 2 recognizes a wide variety of logical record formats to allow information interchange with other computer systems. Logical records, depending on their definition, consist of a fixed or variable number of 6-bit characters.

NAMING FILES

A SCOPE 2 logical file name (lfn) is a 1 to 7 alphanumeric character symbol, the first character of which must be A to Z. Any reference to the file (for example, to read from it, write on it, position it, or change its characteristics) must use the file name.

The name of the job input file (INPUT) is assigned by the system and cannot be changed. In addition to INPUT, the file names listed in Table 1-1 have special meaning to SCOPE 2. A file assigned one of these names is automatically processed at job termination.

TABLE 1-1. SYSTEM FILE NAMES

lfn	Action	
OUTPUT	Line printer listing	
PUNCH	Punching on 80-column Hollerith cards	
PUNCHB	Punching in SCOPE binary on 80-column cards	
FILMPR	Microfile printing	Reserved for future use
FILMPL	Microfile plotting	
HARDPR	Hardcopy printing	
HARDPL	Hardcopy plotting	

INPUT, OUTPUT, PUNCH, and PUNCHB are described in Section 9. Processors for the microfile and hardcopy files are not part of the standard SCOPE 2 system. The file names are reserved for future use.

The system libraries are also assigned names by the system and are available to any job.

Most standard programs and product-set members have an established set of file names that they use for input and output. For example, the COMPASS assembler and the FORTRAN and COBOL compilers assume source language input is on INPUT, that list output is on OUTPUT, and that executable binary output is on LGO. The compilers and assembler all permit the user to substitute other files for the standard set.

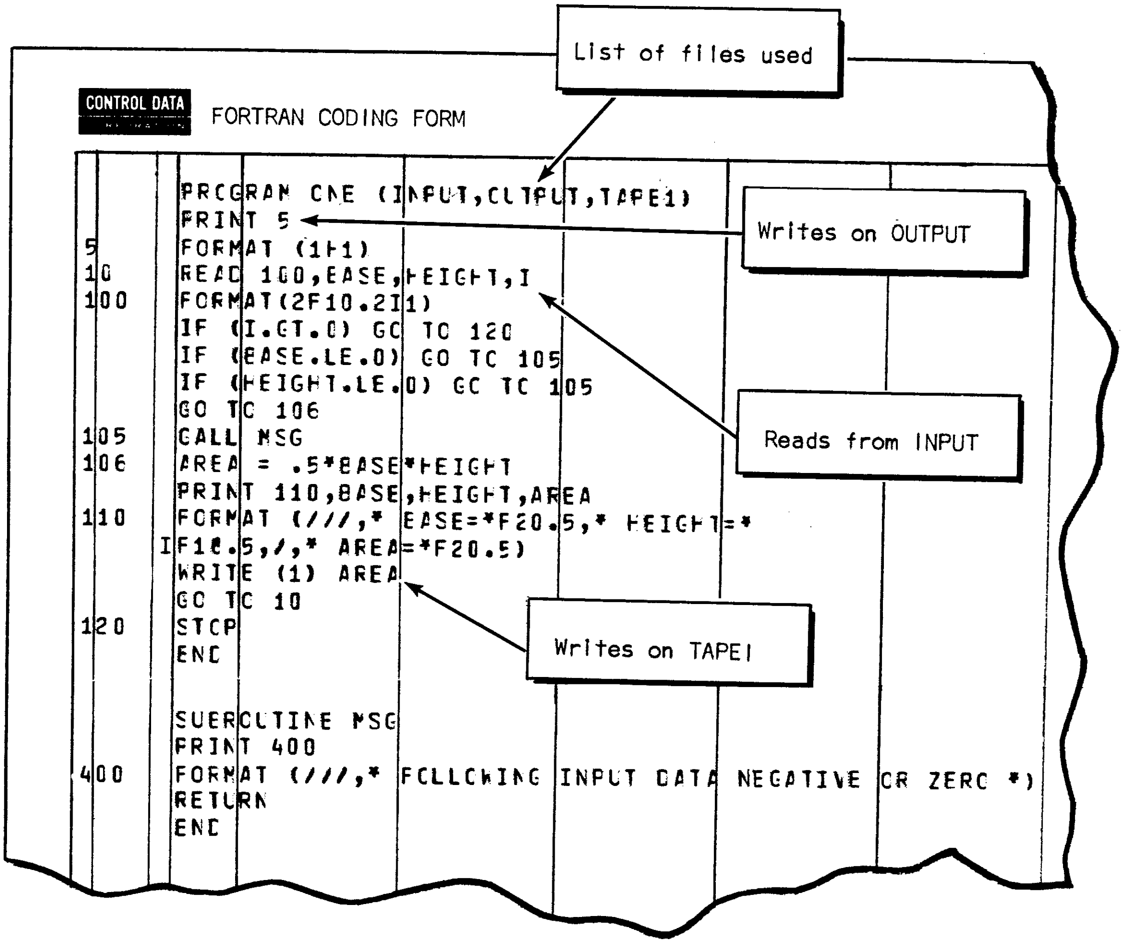
Files used by object programs have names assigned by the programmer in the source language program.

FORTRAN OBJECT-TIME FILE NAMES

The FORTRAN language does not refer to a file directly by its file name. Some of the I/O statements imply certain system file names; others refer to a file by a unit number. Thus, the READ fn,iolist statement refers to file INPUT. The PRINT statement refers to file OUTPUT, and the PUNCH statement refers to file PUNCH. Other READ and WRITE statements, positioning statements, and unit checking statements use unit numbers, where the number can be 1 through 99.

You must correlate the FORTRAN language references with the actual file names through the file list on the PROGRAM statement. If you use a FORTRAN statement such as PRINT, PUNCH, or READ fn,iolist, you must list the implied file (OUTPUT, PUNCH, or INPUT) on the PROGRAM statement. If an I/O statement refers to a unit number, you must list the file name as TAPEn, where n is the unit number. That is, a reference to unit 16 is listed as TAPE16, the lfn by which the system knows it. This does not mean that TAPE16 must be a magnetic tape file.

The program illustrated in Example 1-1 contains READ and PRINT statements and a WRITE statement referring to unit 1. Thus, its PROGRAM statement lists INPUT, OUTPUT, and TAPE1.



CONTROL DATA
FORTRAN CODING FORM

```
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
    5 FORMAT (1H1)
   10 READ 100,BASE,HEIGHT,I
  100 FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
      IF (HEIGHT.LE.0) GO TO 105
      GO TO 106
  105 CALL MSG
  106 AREA = .5*BASE*HEIGHT
      PRINT 110,BASE,HEIGHT,AREA
  110 FORMAT (///,* BASE=*F20.5,* HEIGHT=*
     IF10.5,/,* AREA=*F20.5)
      WRITE (1) AREA
      GO TO 10
  120 STOP
      END

      SUBROUTINE MSG
      PRINT 400
  400 FORMAT (///,* FOLLOWING INPUT DATA NEGATIVE OR ZERO *)
      RETURN
      END
```

List of files used

Writes on OUTPUT

Reads from INPUT

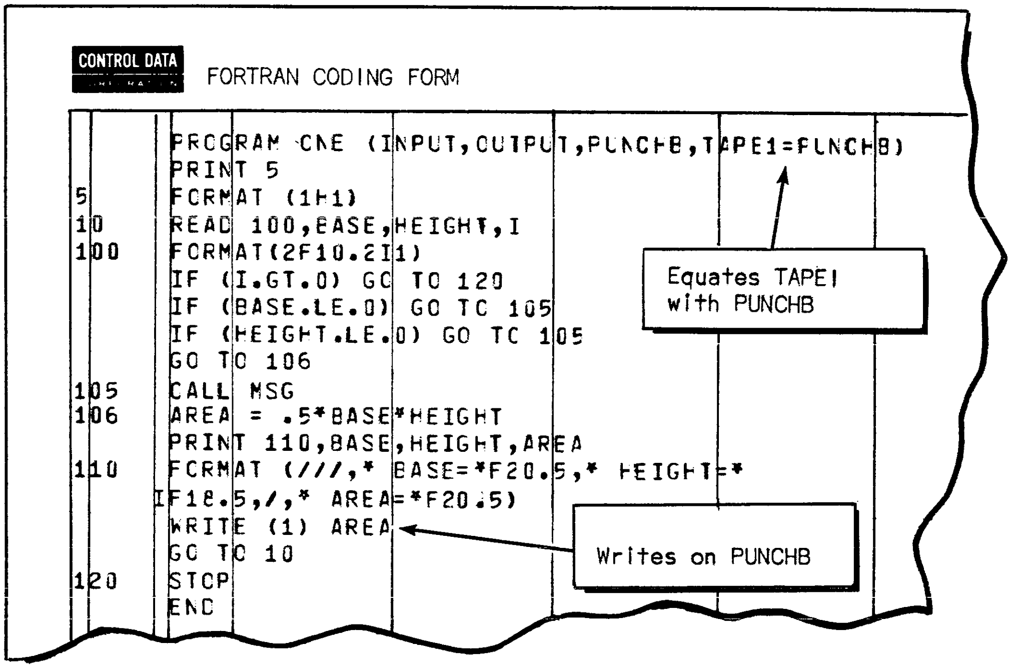
Writes on TAPEI

Example 1-1. Correlating File Names with FORTRAN I/O Statements

Equating File Names

FORTRAN allows you to equate two logical file names. One application of this feature is to make a READ or WRITE statement more flexible. For example, if the I/O statement refers to a unit number, the user can specify it as INPUT, OUTPUT, PUNCH, or PUNCHB simply by renaming it on the PROGRAM statement. It is also convenient where the system or some other subroutines refers to a file by a name that is illegal in the FORTRAN language.

Example 1-2 illustrates a FORTRAN program that uses the INPUT, OUTPUT, and PUNCHB files. The statement that writes on unit 1 now writes on PUNCHB. The file named PUNCHB is automatically punched in binary when the job terminates.



CONTROL DATA FORTRAN CODING FORM

```
      PROGRAM ONE (INPUT,OUTPUT,PUNCHB,TAPE1=PUNCHB)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
      IF (HEIGHT.LE.0) GO TO 105
      GO TO 106
105   CALL MSG
106   AREA = .5*BASE*HEIGHT
      PRINT 110,BASE,HEIGHT,AREA
110   FORMAT (///,* BASE=*F20.5,* HEIGHT=*
     IF18.5,/,* AREA=*F20.5)
      WRITE (1) AREA
      GO TO 10
120   STOP
      END
```

Equates TAPE1 with PUNCHB

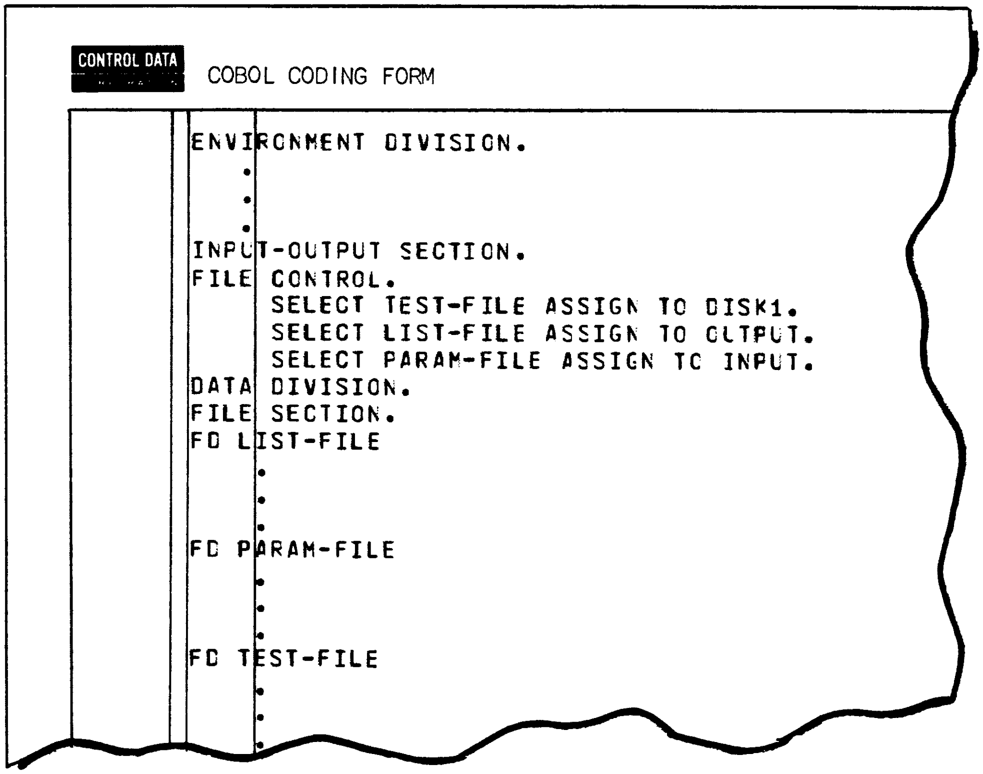
Writes on PUNCHB

Example 1-2. Equating File Declarations on FORTRAN PROGRAM Statement

COBOL OBJECT-TIME FILE NAMES

The user assigns a SCOPE logical file name to each COBOL inplementor name through the ASSIGN clause in the FILE-CONTROL paragraph in the INPUT-OUTPUT section. Any legal SCOPE 2 name can be used.

Example 1-3 illustrates a COBOL program that has FD entries for COBOL files LIST-FILE, PARAM-FILE, and TEST-FILE. The ASSIGN clauses assign these files to SCOPE files named OUTPUT, INPUT, and DISK1.



CONTROL DATA
COBOL CODING FORM

```
ENVIRONMENT DIVISION.
   .
   .
   .
INPUT-OUTPUT SECTION.
FILE CONTROL.
    SELECT TEST-FILE ASSIGN TO DISK1.
    SELECT LIST-FILE ASSIGN TO OUTPUT.
    SELECT PARAM-FILE ASSIGN TO INPUT.
DATA DIVISION.
FILE SECTION.
FD LIST-FILE
    .
    .
    .
FD PARAM-FILE
    .
    .
    .
FD TEST-FILE
    .
    .
    .
```

Example 1-3. COBOL File Name Assignments

USING SCOPE CONTROL STATEMENTS 2

In the typical case, a programmer writes a program in some language (for example, FORTRAN Extended) and submits it to the computer operator in the form of a job deck. Where terminals are available, card images may replace the card deck. The same rules apply for job decks and card images.

In addition to the source language program, the job deck must include SCOPE control statements through which the programmer provides the information SCOPE needs to supervise the job and perform job-related and file-related functions. This section describes how the source program and control statements are organized into job decks.

THE JOB NAME

Assignment of a name to the job is the first step in preparing any job deck. Looking at Example 2-1 you will see that the name of the sample job is JOBSAM. This name is punched on the job identification statement.

A job name must begin in the first column of the job identification statement. A job name can be any combination of up to seven letters and numbers but the first character must be a letter. Blanks cannot be embedded within a job name. When the job name is by itself (unaccompanied by the optional parameters), it must be terminated by a period or right parenthesis. For example, each of the following job statements shows the job name correctly terminated.

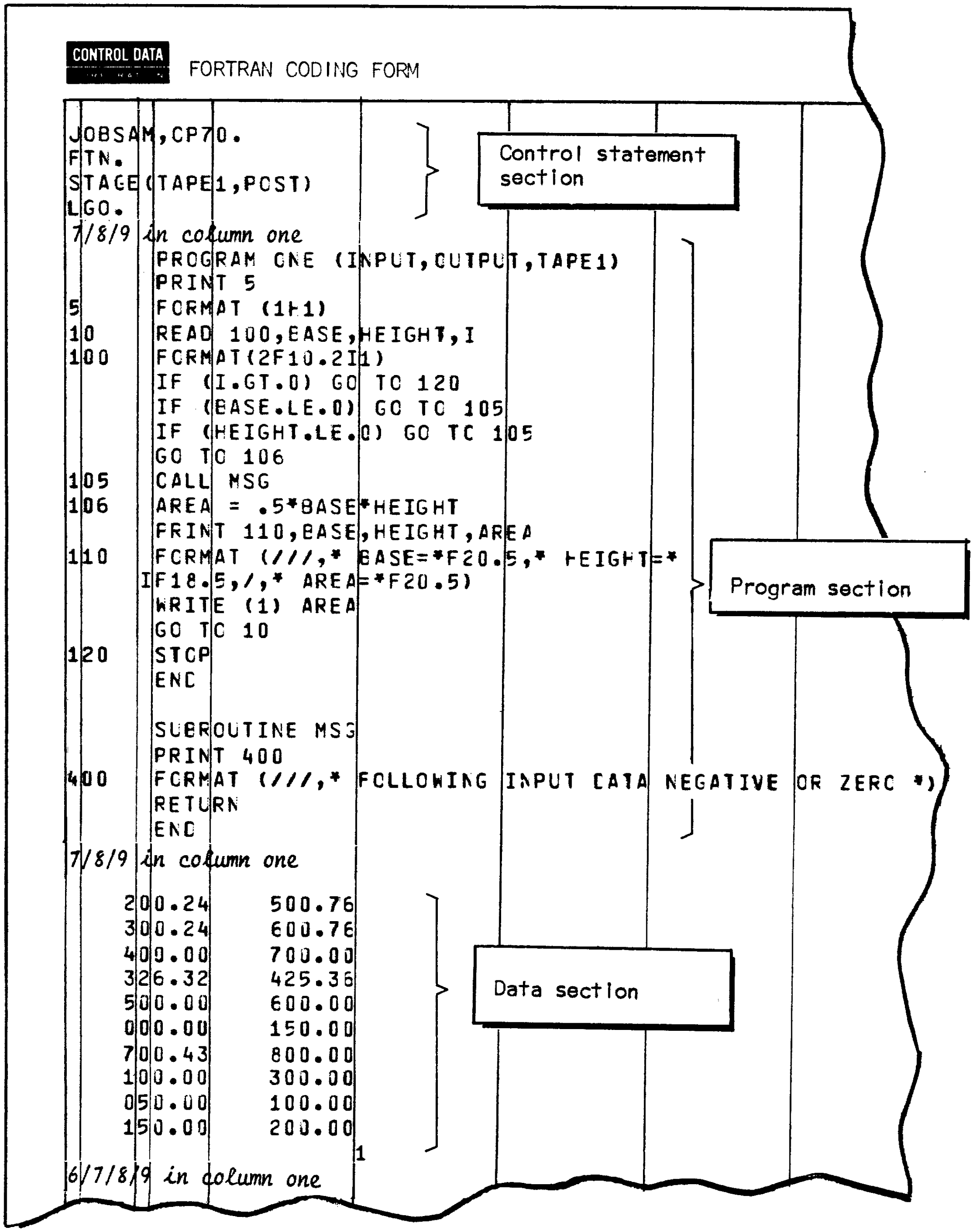
```
JOBNAME.          JOBNAME)
```

Each job must be identified by a unique name both at the station and at the central computer. Suppose you select the name PROCEED for your job and a job with this name has already been submitted. The problem is twofold. First, how does the station differentiate your job from other jobs of the same name, and second, how does SCOPE differentiate jobs submitted from one station with those of the same name from some other station?

When the job enters a station, the station automatically replaces the sixth and seventh characters in the name with two characters unique for each job at the station. As a result, a job named PROCEED might be processed with the name PROCE14. If the name consists of fewer than seven characters, the station fills the unused character positions with zeros and adds the sixth and seventh characters. Thus, a job named K might be changed to K00006M.

Next, when the station sends the job to the central computer system, it sends a 6-character internal identifier. The first three characters identify the station originating the job; the next three characters identify a terminal at the station.

Even though two or more stations might submit a job named K00006M concurrently, SCOPE 2 does not confuse the jobs; each job is uniquely identified.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,PCST)
LGO.
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ 100,EASE,HEIGHT,I
100   FCRMAT(2F10.2I1)
      IF (I.GT.0) GO TC 120
      IF (EASE.LE.0) GC TC 105
      IF (HEIGHT.LE.0) GO TC 105
      GO TC 106
105   CALL MSG
106   AREA = .5*BASE*HEIGHT
      FRINT 110,EASE,HEIGHT,AREA
110   FCRMAT (///,* EASE=*F20.5,* HEIGHT=*
     IF18.5,/,* AREA=*F20.5)
      WRITE (1) AREA
      GO TC 10
120   STCP
      ENC

      SUBROUTINE MSG
      PRINT 400
400   FCRMAT (///,* FCLLOWING INPUT CATA NEGATIVE OR ZERC *)
      RETURN
      END
7/8/9 in column one
    200.24    500.76
    300.24    600.76
    400.00    700.00
    326.32    425.36
    500.00    600.00
    000.00    150.00
    700.43    800.00
    100.00    300.00
    050.00    100.00
    150.00    200.00
6/7/8/9 in column one
```

Control statement section

Program section

Data section

Example 2-1. Sample Job

SCOPE 2 uses the station and terminal identifier to route output from the job back to the originating station and terminal. The fabricated job name (without the appended station and terminal identifier) appears on all of the printer and punch output returned for a job.

OPTIONAL JOB IDENTIFICATION STATEMENT PARAMETERS

The job name is the only information required on the job identification statement. You can optionally supply additional information. When supplying parameters, separate each parameter with a comma and terminate the parameter list with either a period or a right parenthesis. Comments can follow the terminator. The sequence in which parameters are listed is unimportant.

If you supply no other information, the SCOPE 2 system uses default parameters for the job. Default values and the maximum values allowed for these parameters are determined by the installation manager at the time SCOPE 2 is installed in the computer system. The values may vary from site to site. Generally, a system analyst can tell you the values at your site. Record the default and maximum values for your site in the table on the inside back cover of this guide.

The following parameters are allowed.

CP70 or CP76 — Processor code for 6000/CYBER 70 Station. This parameter is described in the following text.

Tn — CPU time limit in octal. This parameter is described on page 2-4.

Pn — Processing priority in octal. This parameter is described on page 2-5.

CMn — Fixed number of words of small core memory allocated for the job. This parameter is not usually specified since it overrides dynamic memory management. The parameter is described with Using Core Memory, Section 4, page 4-3.

ECn — Fixed number of words (expressed in octal thousands) of large core memory allocated for the job. This parameter is not usually specified since it overrides dynamic memory management. The parameter is described with Using Core Memory, Section 4, page 4-6.

Dym — Job dependency string parameter. This parameter is described with the TRANSF control statement with which it is used (see page 4-12).

Rn — Job rerun limit. This parameter is described in Section 4, page 4-13.

MTn — Octal number of on-line 7-track magnetic tape units used by the job. This parameter is described with the magnetic tape REQUEST statement with which it is used (see Using On-Line Tapes, Section 6, page 6-13).

NTn Octal number of on-line 9-track magnetic tape units used by the job. This parameter is described with the magnetic tape REQUEST statement with which it is used (see Using On-Line Tapes, Section 6, page 6-13).

6000/CYBER 70 STATION PROCESSOR CODE

The CP parameter is relevant for jobs entered through a 6000/CYBER 70 station. If the parameter is used for a job submitted through some other type of station, the parameter is ignored.

When a job is entered through the 6000/CYBER 70 station, the 6000 SCOPE operating system must determine from the job identification statement which central processor is to process your job. When the parameter is omitted, 6000 SCOPE uses the system default, which is most commonly set for processing at the 6000 Series Computer System rather than the 7600 Computer System.

7600 Processing

Use CP70 to unconditionally specify processing at the 7600 (or CYBER 70/Model 76). If the 7600 is not currently communicating with the 6000 Series station, the job waits indefinitely for communication to be established. For this publication, use of CP70 is conventional.

The following job identification statement allows the job to be processed only at the 7600 Computer System.

```
JOB,CP70.
```

7600 or 6000 Processing

Use CP76 to specify processing at the 7600 as a first choice but at the 6000/CYBER 70 station if the 7600 is not currently linked to the 6000 station. It is not used for load leveling. That is, the size of the job queues for the two systems is not a factor in assigning a processor.

The following job identification statement requests that the job be processed at the 7600 if it is available; otherwise, it is to be processed at the 6000 station.

```
JOBNAME,CP76.
```

EXECUTION TIME LIMIT

For each job in the system, SCOPE 2 monitors the amount of time that programs for the job occupy the central processor unit (CPU). This time does not include the time spent in the input queue, staging files, waiting for access to the CPU, or waiting for completion of I/O requests. When SCOPE detects that the execution time has expired, it terminates job processing. The default time limit is usually 8 seconds. If the system default time limit is insufficient for your job, supply the T parameter on your job identification statement. You should also set a time limit if you feel that the default is too high.

The time is expressed in seconds as an octal value prefixed by the letter T. You can either calculate the octal value or you can use the following rule to arrive at an approximation of the octal value.

Rule: The time in octal seconds equals the approximate time in minutes multiplied by 100. Note, however, that a decimal value of 8 or more must be converted to the octal equivalent.

$$\mathrm{sec}_8 = 100 \times \mathrm{min}$$

For example, if your job requires 4 minutes of CPU time, you would convert this time to 400 octal seconds for use on the job identification statement. Enter the value as T400. For 9 minutes, you would enter T1100, having converted the 9 to its octal equivalent.

The following job identification statement sets the time limit for the job to approximately 9 minutes.

```
BIGJOB,CP70,T1100.
```

If a job contains an EXIT statement, and the job abnormally terminates because of having used its time, SCOPE 2 extends the limit by 8 seconds to permit you to obtain a dump or save valuable data.

To be certain that your job will have access to the CPU until it has completed processing, regardless of the requested time limit, set the execution time parameter to T7777. This special value acts as an infinite time limit. It also represents the maximum possible value for the T parameter.

CAUTION

Use caution when setting high or infinite time limits. If your job contains an error such as an infinite loop, the program will continue to execute and you will be charged for the time used.

JOB PRIORITY

The P parameter is rarely specified. Default priority is adequate for most applications. A job with very high resource requirements, however, will sometimes warrant an increase in processing priority (for example, if it uses all the on-line units, requires a large amount of CPU time, uses a large percentage of LCM, or is heavily I/O bound).

To override the default priority specify the letter P followed by 1 to 4 octal digits. The highest priority a user can assign is set by an installation parameter (usually 7000_8). The priority assigned causes the job to be processed ahead of all jobs having lower priorities.

The lowest priority that can be assigned and still have the job processed is 1. Any value between 1 and 777_8 is automatically reassigned to 1000_8. If the priority is 0, the job will not be processed until the operator assigns a valid priority.

On the following job identification statement, the priority is set to 2000_8.

```
SWIFTY,CP70,P2000.
```

CONTROL STATEMENT SECTION

All control statements applying to a job must be in the control statement section, which is always the first section in the deck. Control statements cannot appear in any other part of the job deck. They are processed one at a time and determine all operations performed on subsequent sections of the job deck.

The control statement section consists of the following kinds of statements.

- SCOPE 2 control statements
- Loader control statements
- Record manager control statements

These statements serve the following purposes.

- Identify the job and some of its characteristics
- Request devices needed for job processing and specify other file-related activities
- Call for compilation or assembly of the source language program
- Direct the loading of programs into small core memory and loading of data into large core memory
- Call for execution of the object program resulting from compilation or assembly
- Specify exit paths and job termination conditions

All statements must be prepared observing the following syntax rules for control statements.

1. Each statement must consist of a 1- to 7-character statement name and a terminator, or must consist of a name followed by a separator, a parameter list, and a terminator.

2. The terminator can be either a period or a right parenthesis.

   name.   or   name)

3. The separator following the statement name is conventionally a comma or a left parenthesis. However, one or more blanks following the statement name are also interpreted as a separator; elsewhere, blanks are ignored.

   name(parameters)   or   name,parameters.   or   name parameters.

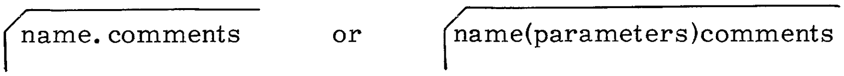
4. The parameter list consists of one or more fields of information separated by commas.

   name($p_1,p_2,\ldots,p_n$)   or   name,$p_1,p_2,\ldots,p_n$ or   name $p_1,p_2,\ldots,p_n$.

Parameters in the list are often in keyword form, that is, each p_i could be expressed as $x = y$ or $x = y_1/y_2/\ldots/y_n$. Thus, commas, equal signs, and slant bars are conventional delimiters in parameter lists.

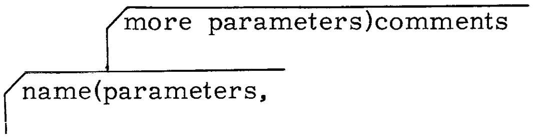
5. Literals permit any of the characters otherwise illegal or interpreted as separators to be used in the parameter list. Blanks within a parameter are deleted except within a literal. A literal is any character string delimited by a pair of dollar signs. Two consecutive dollar signs within a literal constitute a single dollar sign. That is, the literal $ab$$cd$ is interpreted as ab$cd.

6. Any characters can follow the control statement terminator. This allows the remainder of the line to be used for comments.



```
name. comments          or          name(parameters)comments
```

7. Continuation cards are allowed for statements too long for a single card. To continue a statement, the card containing the statement to be continued must end with a (, / or =. It must not contain a terminator. The continuation card must have a nonblank character in column 1. Comments cannot be continued because they follow a terminator.



```
        more parameters)comments
name(parameters,
```

It is not unusual for a program called by a control statement to require a secondary kind of control statement known as a directive. Directives for a program can be in the job deck in a separate section from control statements, or they can be on some other file. In SCOPE 2, the system routines LIBEDT, TRAP, ANALYZE, and UPDATE each has its own set of directives. The syntax of these directives is tailored to the needs of each program.

Each control statement is referred to as a job step. After the operation requested by the statement has occurred, SCOPE advances to processing the next control statement, that is, performing the next step in your job.

Sections following the SCOPE control statement section consist of source language decks, binary object decks, data, and directives required by specific job steps. If no job step requires input from the job deck, the deck consists of only the control statement section.

Example 2-1 represents the typical case in which the first job step that requires input from the job deck is the FORTRAN Extended compiler (called by the name FTN). Thus, the FORTRAN language program is the second section in the deck. The next job step requiring input from the job deck is the object program. Its execution is called for by the control statement LGO. Therefore, data for this program forms the third and final section.

SEPARATOR CARDS

Cards with certain unique patterns define the structure of a deck and separate decks in the card reader.

END-OF-SECTION CARD

Terminate each section with an end-of-section card (Figure 2-1). This card has rows 7, 8, and 9 punched in column 1. Columns 2 and 3 optionally contain the Hollerith punch for octal codes 00 through 16. These are section level numbers. They are irrelevant in SCOPE 2 but have significance to SCOPE 3.4. End-of-section cards are often referred to as end-of-record cards.

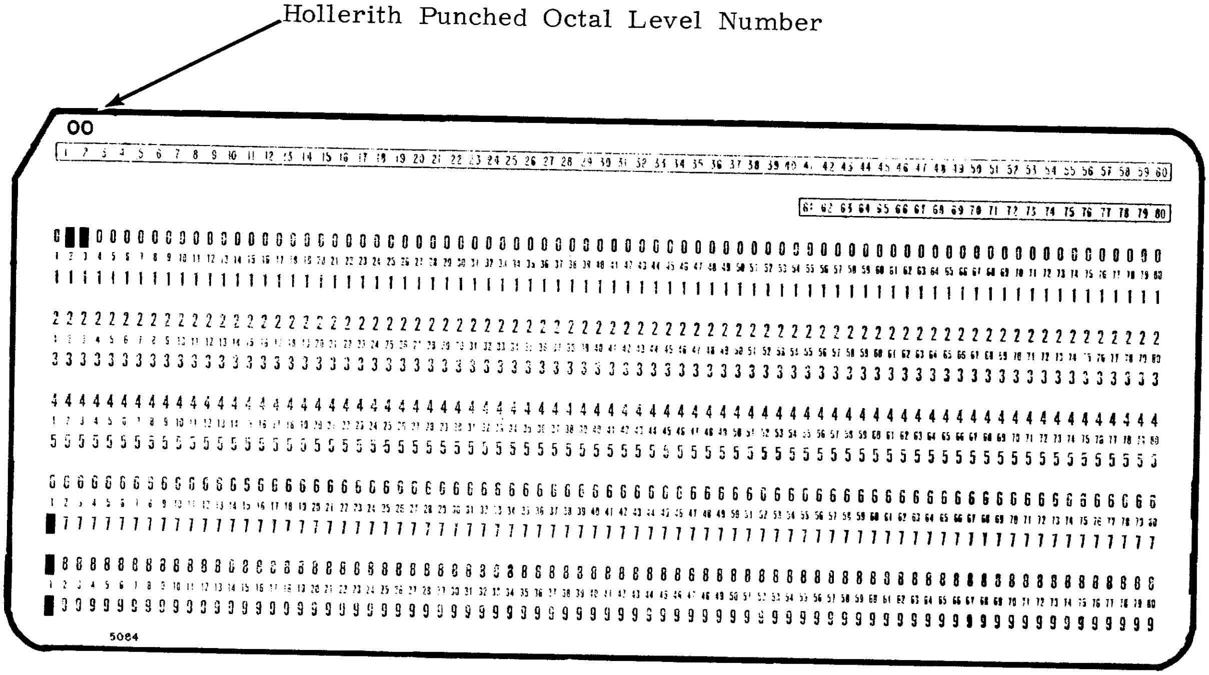


Figure 2-1. End-Of-Section Card (EOS)

END-OF-PARTITION CARD

Terminate a formatted binary deck (core image or object module) with a single end-of-partition card or two end-of-section cards. The end-of-partition card (Figure 2-2) has rows 7, 8, and 9 punched in column 1 and has the Hollerith punch for octal code 17 in columns 2 and 3. This card is equivalent to the 7600 SCOPE 1.1 end-of-file card (multipunch 6/7/9). As will be shown later, FORTRAN and COBOL object-time routines also recognize the EOP separator. In previous 6000 SCOPE systems, the end-of-information card is referred to as an end-of-file card.

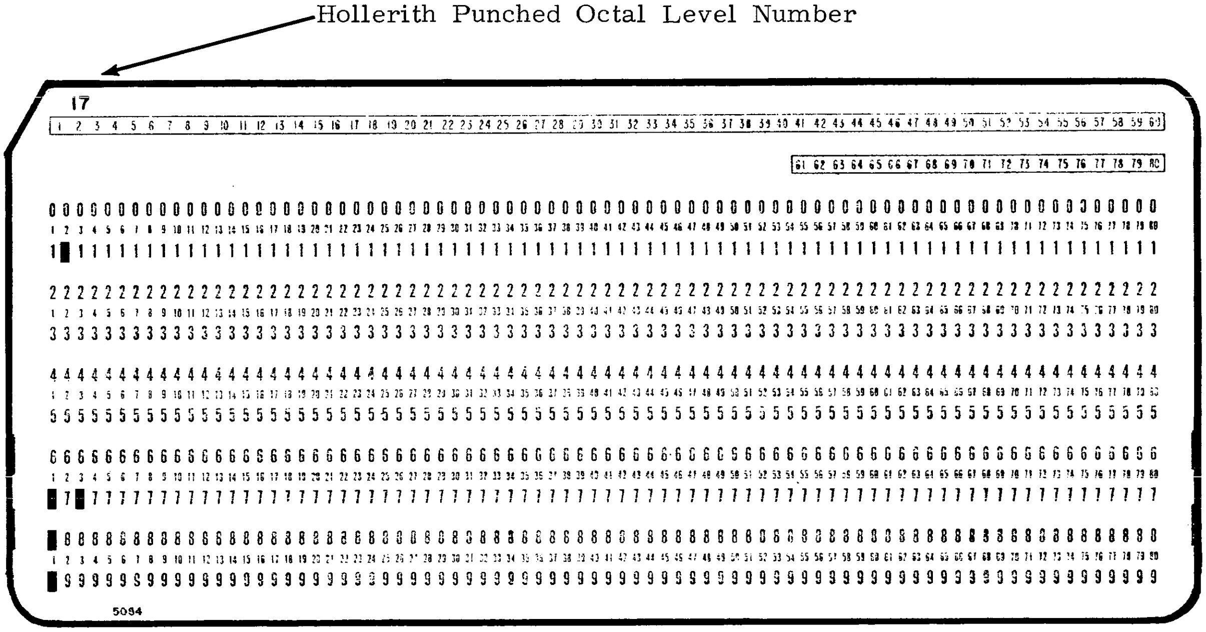


Figure 2-2. End-Of-Partition Card (EOP)

END-OF-INFORMATION CARD

Terminate the last section of the job deck with an end-of-information (EOI) card. This must be the only EOI card in your deck. The end-of-information card (Figure 2-3) has rows 6, 7, 8, and 9 punched in column 1. Some programmers make a practice of including an EOS card before the EOI card. This practice is unnecessary since the EOI serves to terminate the last section and the job deck.

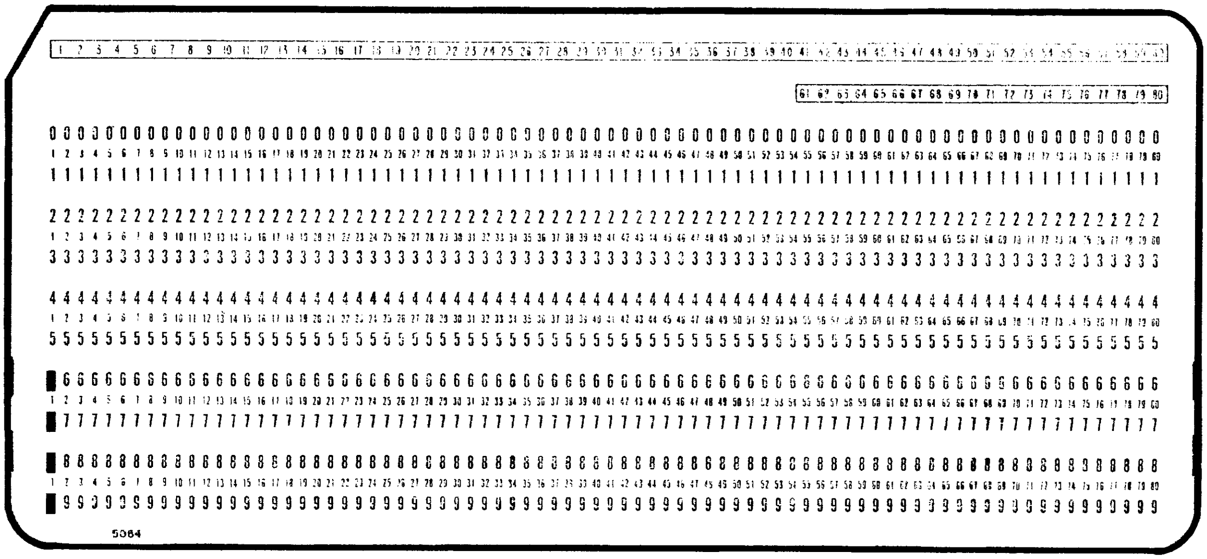


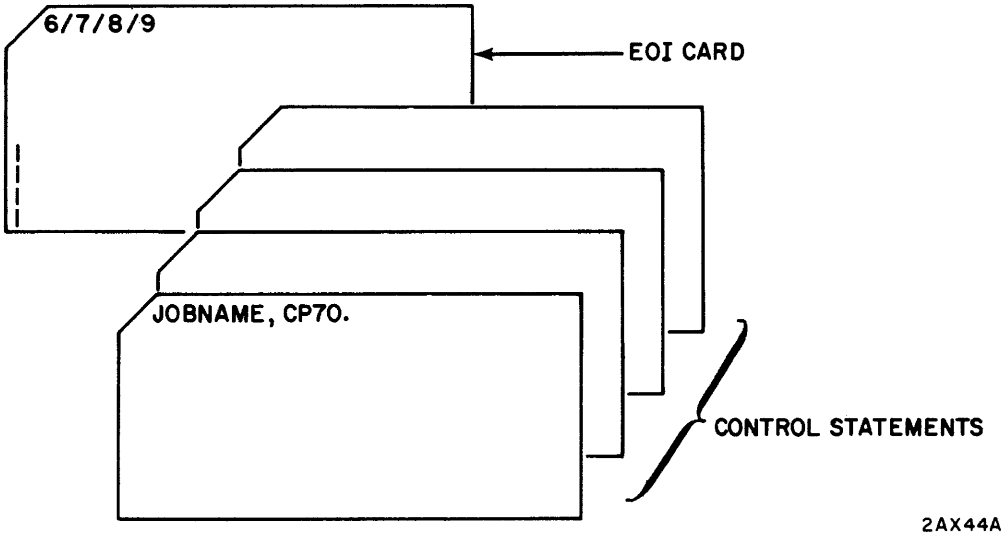
Figure 2-3. End-Of-Information Card (EOI)

EXAMPLES

CONTROL STATEMENT SECTION

In the simplest case, a job consists of only one section, the control statement section. This happens when no job step requires card input with the job deck.

Example 2-2 illustrates a job that consists of only the one section.



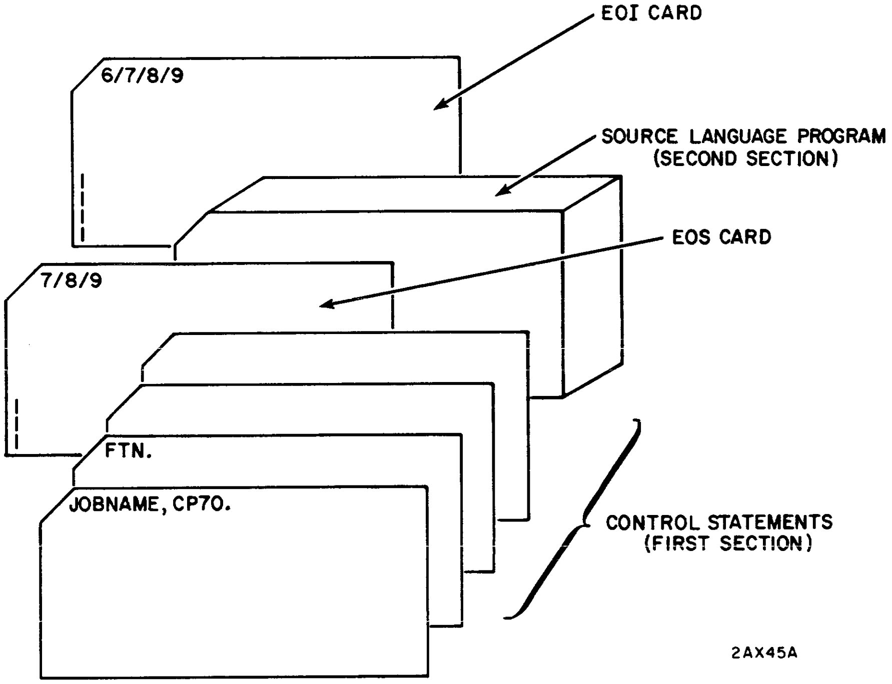
Example 2-2. Job Containing Control Statements Only

COMPILE SOURCE LANGUAGE PROGRAM

A job consists of more than one section if one or more of the programs called by the control statements, that is, job steps, requires card input from the job deck.

Usually, a program requiring input submitted as cards looks for these cards as the next section in the job deck. Thus, if a compiler is the first program executed, it seeks the source language program deck in the section immediately following the control statements.

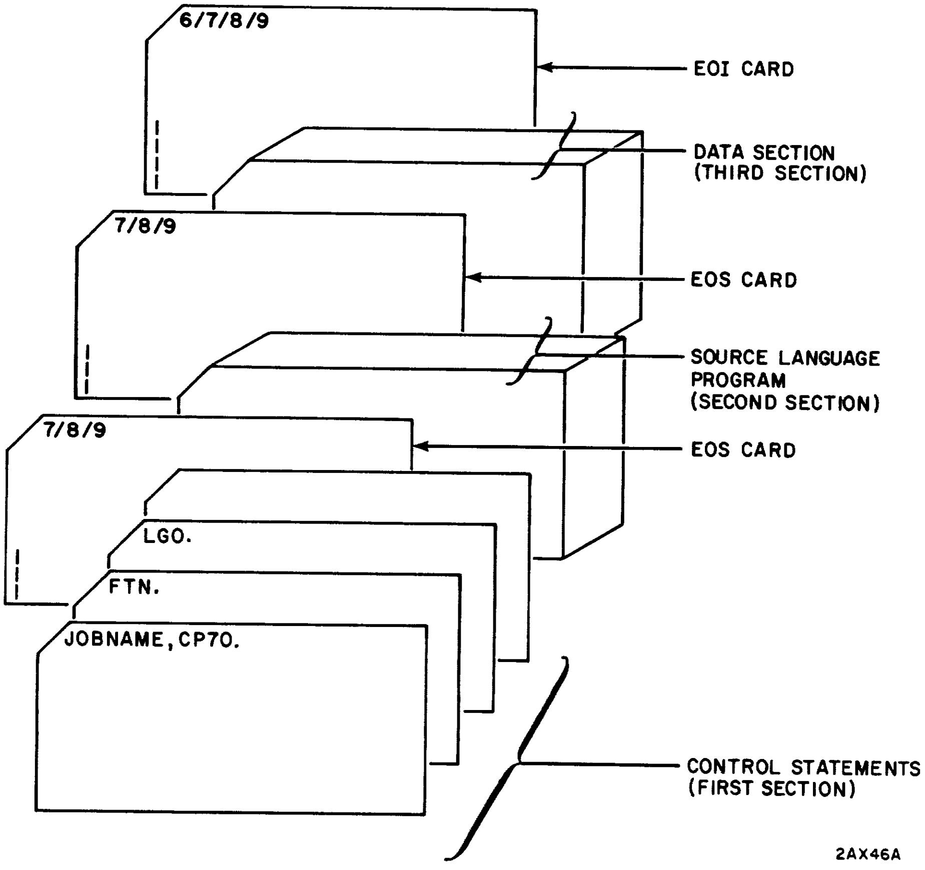
Example 2-3 illustrates a job that calls for compilation of a FORTRAN source language program.


EOI CARD
6/7/8/9
SOURCE LANGUAGE PROGRAM (SECOND SECTION)
EOS CARD
7/8/9
FTN.
JOBNAME, CP70.
CONTROL STATEMENTS (FIRST SECTION)
2AX45A

Example 2-3. Job With Source Language Program

COMPILE AND EXECUTE

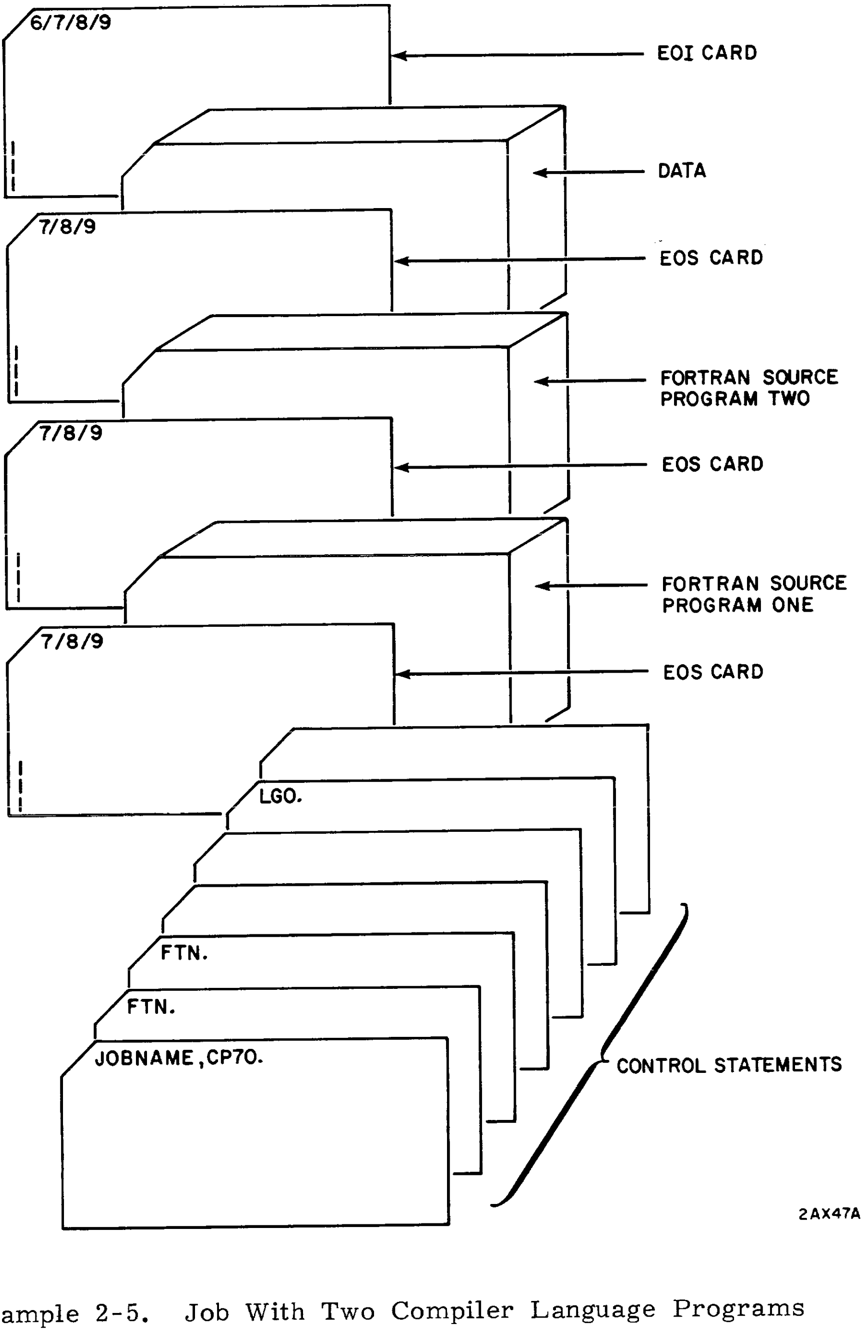
Typically, execution of the program compiled by the compiler is called for following the compilation. In this example, execution is requested through the LGO statement. LGO is a file on which all the compilers and assemblers place object programs unless some other file is specified. This statement is described in detail under File Name Calls. If this object program requires data, it is the next section after the source language section, as shown in Example 2-4. Again, each section other than the last terminates with an EOS card and the final section terminates with an EOI card. Notice that the deck illustrated in Example 2-4 parallels the job illustrated in Example 2-1. That is, it contains three sections, the control statement section, the source language section, and the data section.



Example 2-4. Job With Source Language Program and Data

TWO COMPILATIONS WITH COMBINED EXECUTION OF THE OBJECT PROGRAMS

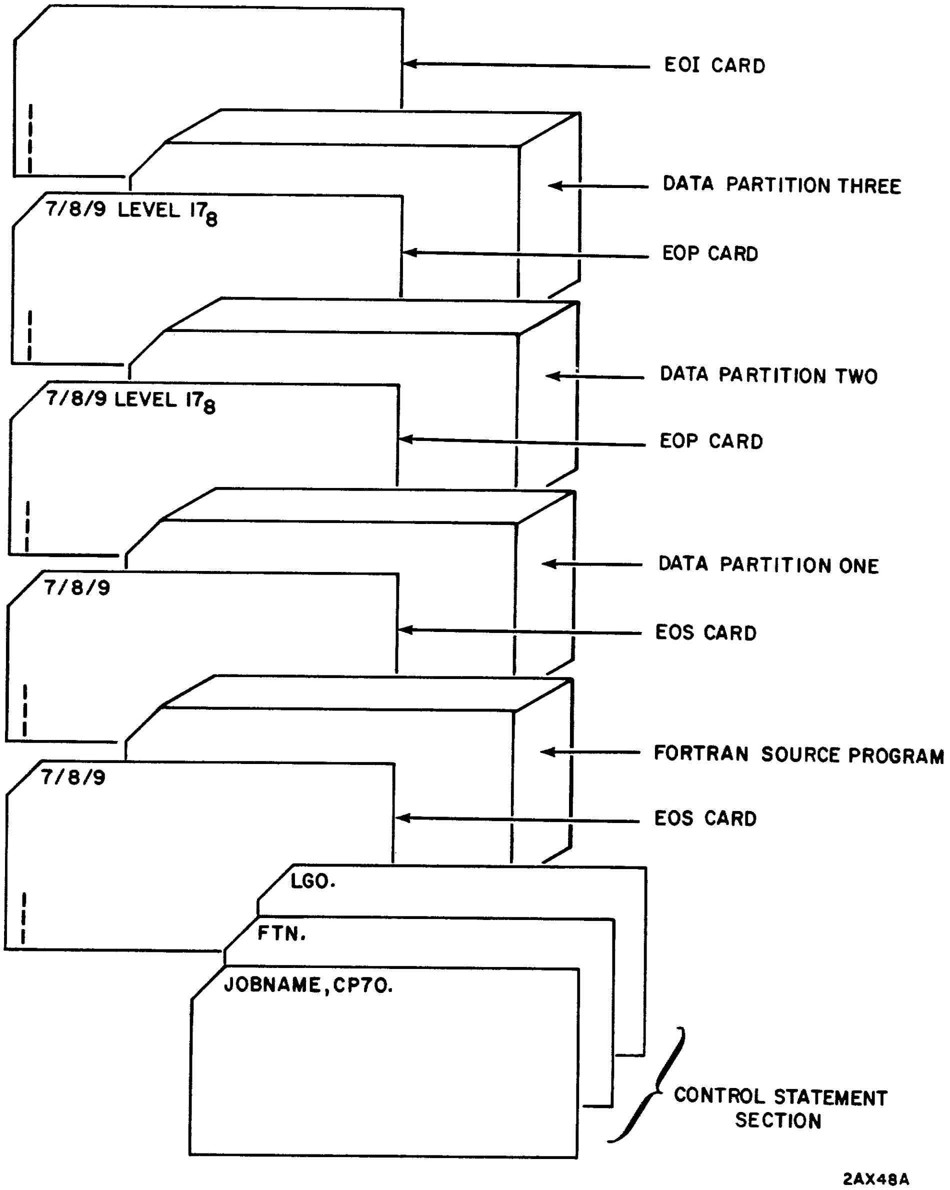
Very elaborate jobs are possible, such as those that include compilation and execution of more than one program. Example 2-5 illustrates a job containing two FORTRAN programs. Both calls to the compiler write the object programs on a file named LGO. The object programs are loaded and executed as a single program through use of the LGO control statement.



Example 2-5. Job With Two Compiler Language Programs

COMPLEX DATA STRUCTURE

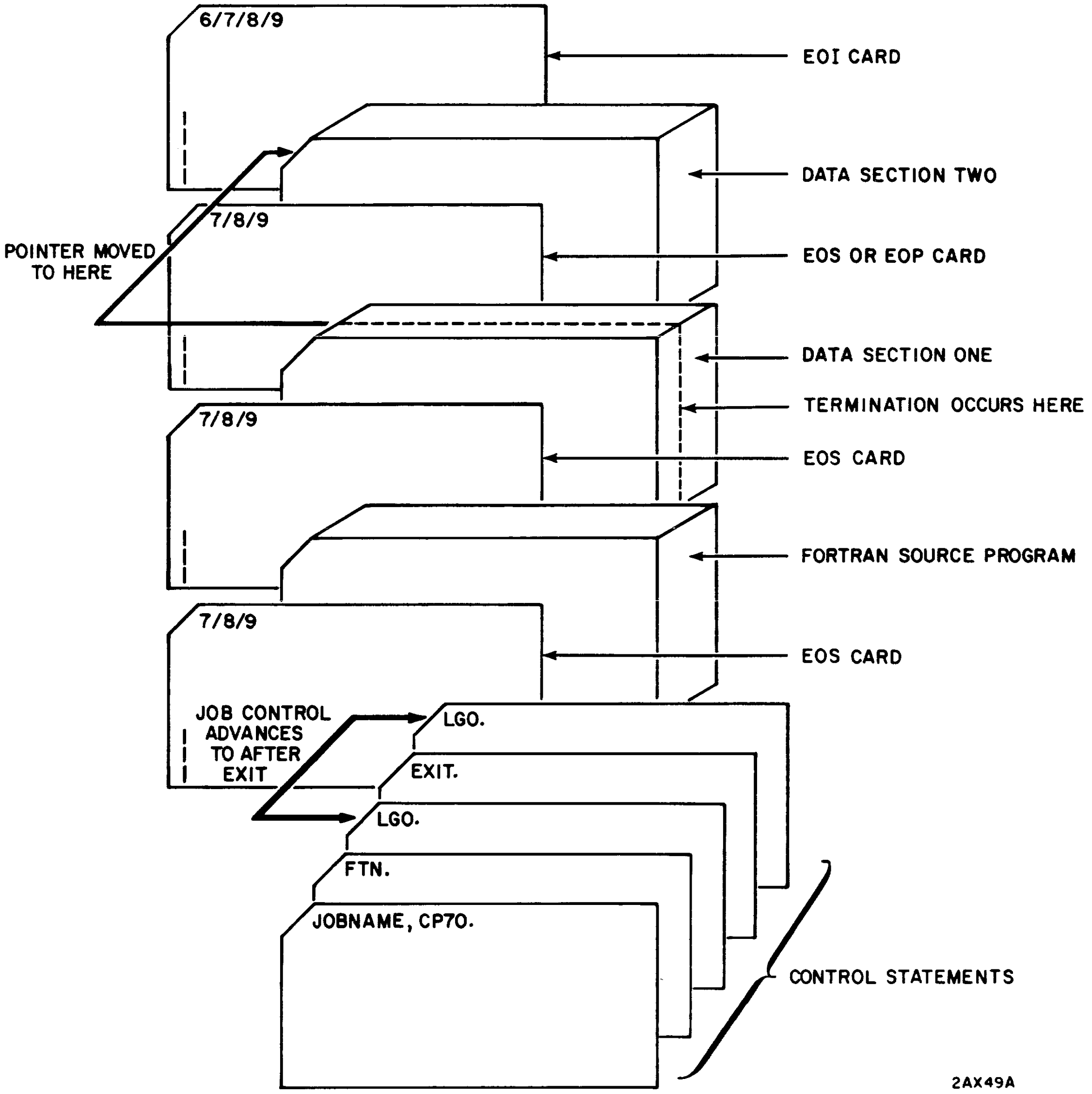
The data in the job deck need not be confined to a single section or partition. It depends entirely on what the program is doing. Example 2-6 illustrates a job deck containing three partitions of data. In this example, the data is divided into partitions rather than sections because the FORTRAN object program considers an EOS equivalent to EOP.



Example 2-6. Job With Complex Data Structure

JOB STEP TERMINATION

Example 2-7 illustrates a program consisting of a compilation (FTN statement) and execution of the compiled program (LGO statement). If the compiled program terminates before it has read the last data card, the next-card pointer is advanced to the beginning of the next section. In this example, the EXIT statement causes job processing to resume with the second LGO statement calling for re-execution of the compiled program. This time, the program begins reading at the beginning of the second data section. The FORTRAN and COBOL compilers generate instructions in the compiled programs that cause this automatic advancement of the next-card pointer. If your job aborts before the compiled program reads any of the data cards, this automatic advancement of the next-card pointer does not take place. The pointer is positioned at the beginning of the first data section.



Example 2-7. Job Step Termination

JOB PROCESSING

This section describes the most common steps in a job and gives further information on the principles and techniques involved in loading and executing programs.

COMPILING OR ASSEMBLING PROGRAMS

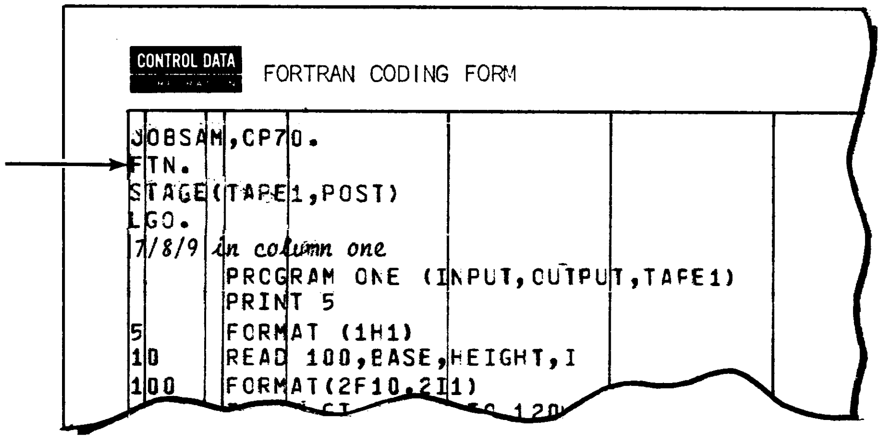
The most common first step in job processing is translation of the source language program into an object program, that is, into machine language. This occurs through compilation if the source language program is written in a compiler language such as FORTRAN or COBOL, or by assembly if it is written in the COMPASS assembly language.

The request for compilation or assembly is a request for SCOPE to load the compiler into small core memory and execute the compiler program. The compiler translates the source language program into machine language. Table 3-1 shows the requests needed to compile or assemble programs written in the languages available with the SCOPE 2 Operating System. Such requests often resemble the name of the compiler or assembler called.

TABLE 3-1. REQUESTS FOR COMPILATION OR ASSEMBLY

Language Used for Source Program	Request Issued for Compilation/Assembly
FORTRAN Extended	FTN.
FORTRAN (RUN)	RUN(S)
COBOL	COBOL.
COMPASS	COMPASS.

The arrow in Example 3-1 points to the request statement that results in the load and execution of the FORTRAN Extended compiler.


CONTROL DATA
FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,POST)
LGO.
7/8/9 in column one
      PRCGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
```

Example 3-1. Request for FORTRAN Extended Compilation

For a job that contains both a COMPASS language program and a compiler language program, the requests and deck arrangements required for compilation and assembly vary. They depend on the language used and the order in which the programs are to be assembled and compiled, as well as other factors.

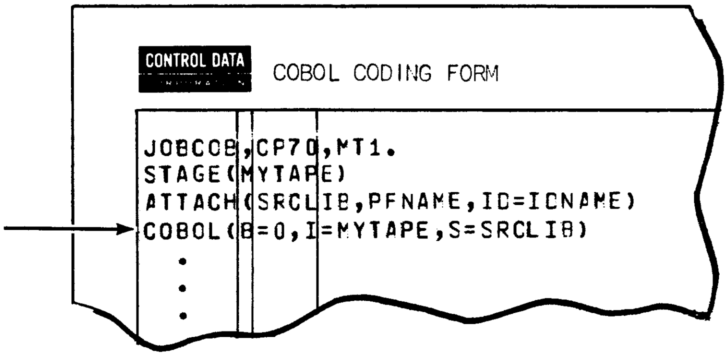
As you become more familiar with the compiler or assembler language, you will want to take advantage of the several programming options available on the compiler or assembler request statement. These options have a direct effect on the compilation or assembly.

Some options provide different kinds of program listings, others allow you to specify different files for input and output. All such options are described in detail in the reference manuals for each language. The most commonly used options are listed in Table 3-2.

The codes used for the options vary according to the language used. Options are listed after the word (for example, RUN, FTN, COMPASS, COBOL) that calls the compiler or assembler. The options can be in any order. The first option is preceded by a comma or left parenthesis; the last option is followed by a period or right parenthesis. All other options are separated by commas.

Analyze Table 3-2 to see what happens when all optional parameters are omitted. For COBOL, COMPASS, or FORTRAN Extended, your program will be read from the job deck and translated into a machine language object program written on a file named LGO. A listing of your source program and any errors that may have occurred during assembly or compilation will be written on the OUTPUT file and automatically printed. For FORTRAN RUN, parameters are required to produce the above results; a RUN statement with no parameters results in compilation and execution without a source listing.

Example 3-2 illustrates a COBOL request statement that calls for the binary output from compilation to be suppressed, for the source language program to be on a staged tape named MYTAPE, and a source library to be on a permanent file named SRCLIB. Use of staged magnetic tapes is described in detail in Section 6; attaching of permanent files is described in Section 8.



CONTROL DATA COBOL CODING FORM

```
JOBCOB,CP70,MT1.
STAGE(MYTAPE)
ATTACH(SRCLIB,PFNAME,ID=IDNAME)
COBOL(B=0,I=MYTAPE,S=SRCLIB)
   •
   •
   •
```

Example 3-2. Request for COBOL Compilation With Options Specified

TABLE 3-2. OPTIONS AVAILABLE DURING COMPILATION/ASSEMBLY

Option	COBOL	COMPASS	FORTRAN RUN††	FORTRAN Extended
Takes program from INPUT file	omitted or I or I ≠ INPUT	omitted or I = INPUT	omitted or I = INPUT	omitted or I or I = INPUT

Takes program from file named lfn.†	I = lfn
	or
	INPUT = lfn
		I = lfn	I = lfn	I = lfn
				or
				INPUT = lfn

Translates source program into object (binary) program and writes it on LGO in preparation for loading and execution. Also produces normal listing of source decks.	omitted or B or B = LGO or L	omitted or B or L	S	omitted or B or B = LGO or L
Translates source program into object (binary) program and writes it on file lfn in preparation for loading and execution. Also produces normal listing of source decks.	B = lfn	B = lfn	B = lfn	B = lfn
Punches binary cards of object programs.	B = PUNCHB	B = PUNCHB	B = PUNCHB	B = PUNCHB
Compiles and executes			G or all optional parameters omitted	G

† lfn = logical file name, 1 to 7 characters, first character must be alphabetic

†† FORTRAN RUN also recognizes an order-dependent form of the RUN statement for which missing optional parameters are indicated by commas.

LOADING AND EXECUTING PROGRAMS

The request to load and execute a program can be very simple, requiring the use of a single load and execute statement, or can be very complex, involving the use of an elaborate sequence of control statements called a loader control statement sequence.

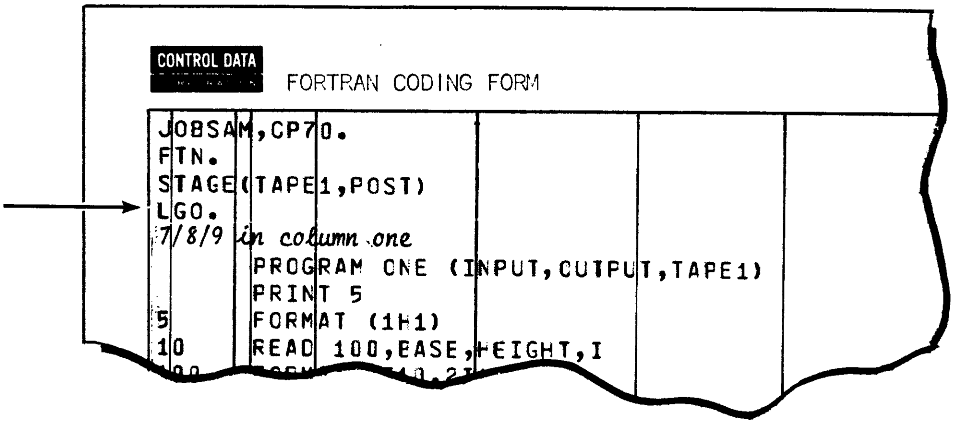
COMBINED LOAD AND EXECUTE REQUEST

The LGO Statement

After the program has been compiled or assembled, the most direct way to load the object program into small core memory and have it executed is with a simple one-statement request. If your request for compilation or assembly does not explicitly name a binary output file, your program is written on a file named LGO. In this case, you can use the following file name statement for loading and execution.

```
LGO.
```

Literally, this request tells the computer "load and go". In Example 3-3 the arrow points to the LGO statement.



CONTROL DATA FORTRAN CODING FORM

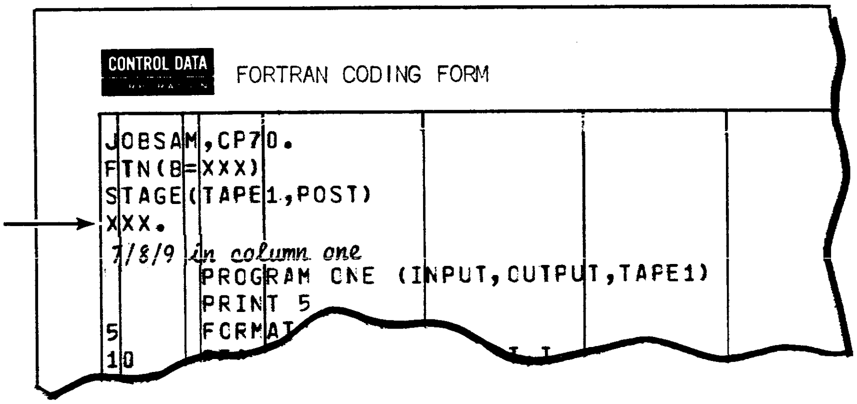
```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,POST)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,EASE,HEIGHT,I
```

Example 3-3. LGO File Name Statement

The loader always rewinds LGO before loading from it. If more than one compiler or assembler writes on LGO before execution is called for, the output from both the language processors is loaded and executed as a single program.

The compiler options permit you to rename the load-and-go file. In this case the statement that calls for loading and execution must use the file name you designate.

In Example 3-4, the load-and-go file is renamed XXX on the FTN statement. XXX calls for load-and-go of the compiled program.


CONTROL DATA
FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN(B=XXX)
STAGE(TAPE1,POST)
XXX.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT
10
```

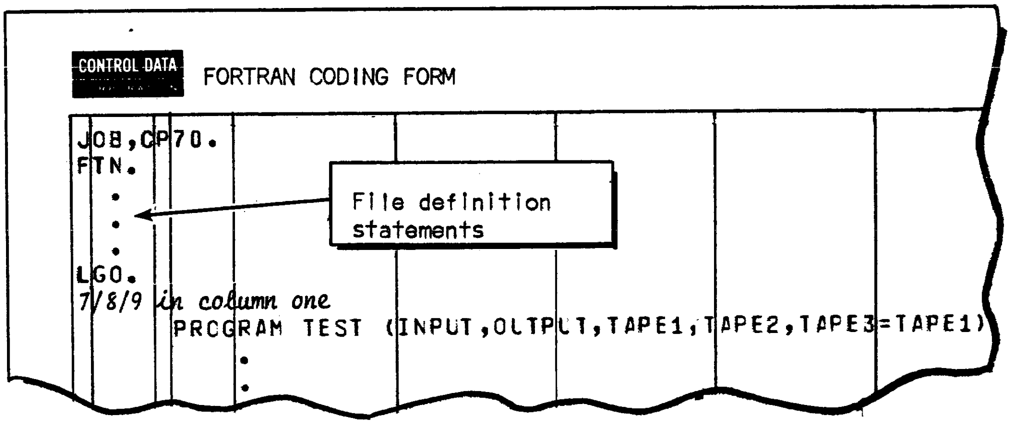
Example 3-4. Renaming the Load-And-Go File

Substituting File Names at Execution Time

Usually, the file names you supply on the PROGRAM statement are the names used for the files at execution time. You can, however, change these names when you execute the program by supplying parameters on the LGO statement.

If no parameter is specified on the load-and-go statement, the file names are the same as those in the PROGRAM statement.

In Example 3-5, the program uses files INPUT, OUTPUT, TAPE1, and TAPE2.

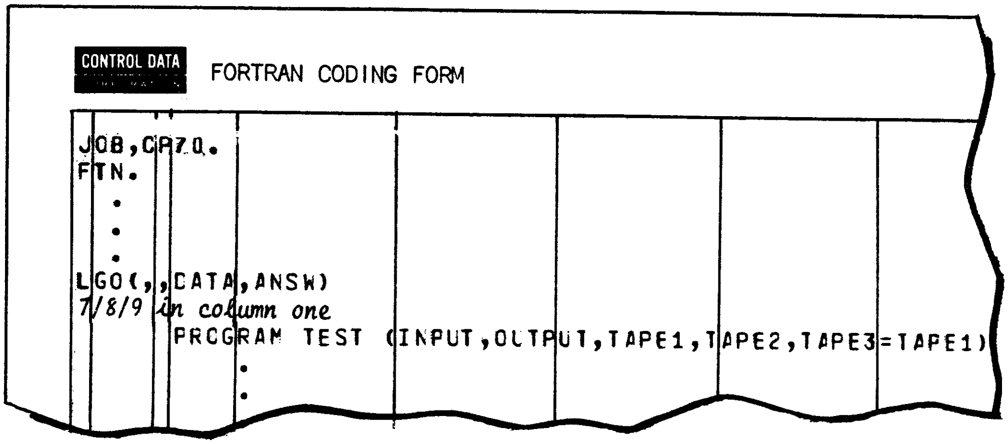

CONTROL DATA
FORTRAN CODING FORM
File definition statements

```
JOB,CP70.
FTN.
 •
 •
 •
LGO.
7/8/9 in column one
      PROGRAM TEST (INPUT,OUTPUT,TAPE1,TAPE2,TAPE3=TAPE1)
      •
      •
```

Example 3-5. No Substitution of File Names on LGO

File names specified on the load-and-go statement replace the names on the PROGRAM statement in a one-to-one relationship.

In Example 3-6 the program still uses files INPUT and OUTPUT. However, files TAPE1 and TAPE2 have been replaced by the files named DATA and ANSW, respectively. TAPE3 which was equated to TAPE1 is also replaced by DATA. Any reference to TAPE1 or TAPE3 in the program is actually a reference to DATA. Any reference to TAPE2 is actually a reference to ANSW.

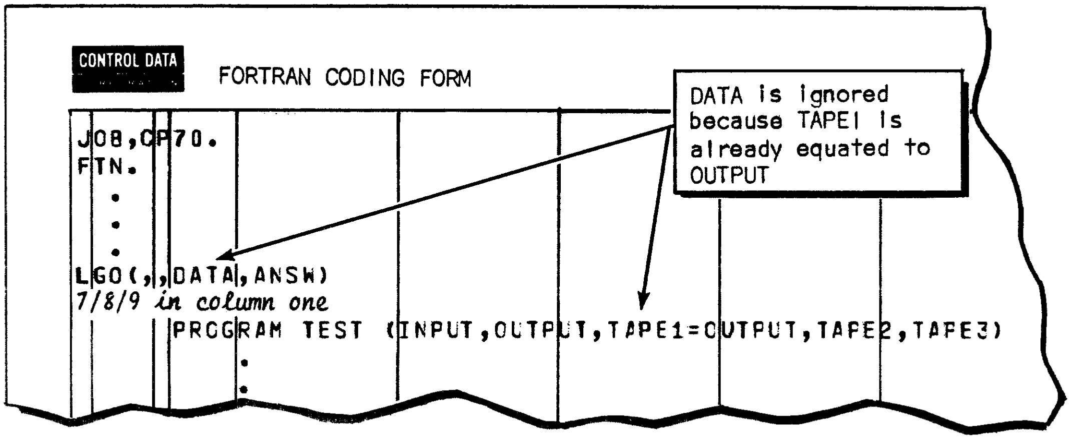


```
JOB,CP70.
FTN.
 •
 •
 •
LGO(,,DATA,ANSW)
7/8/9 in column one
      PROGRAM TEST (INPUT,OUTPUT,TAPE1,TAPE2,TAPE3=TAPE1)
        •
        •
```

Example 3-6. Substitution of File Names on LGO

If the file name on the load-and-go statement specifies a file that has been equated on your PROGRAM statement, the equate on the PROGRAM statement takes precedence. That is, the redefinition is ignored.

Note that in Example 3-7, DATA refers to the PROGRAM parameter TAPE1=OUTPUT. Because TAPE1 has already been equated to OUTPUT, the redefinition to DATA is ignored. Any reference to TAPE1 in the program is actually a reference to OUTPUT.



DATA is ignored because TAPE1 is already equated to OUTPUT

```
JOB,CP70.
FTN.
 •
 •
 •
LGO(,,DATA,ANSW)
7/8/9 in column one
      PROGRAM TEST (INPUT,OUTPUT,TAPE1=OUTPUT,TAPE2,TAPE3)
        •
        •
```

Example 3-7. Precedence of Equating File Names

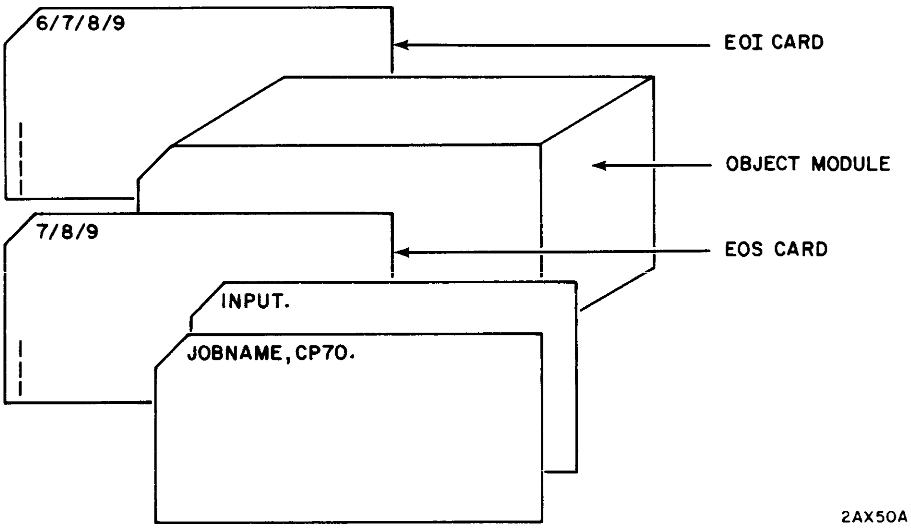
Loading from INPUT

If you have on hand the punched deck of an object program, you can load and execute the program from the job deck. This requires an INPUT statement, as follows:

```
INPUT.
```

When load is from INPUT, the loader does not rewind the file before loading from it. That is, the job deck pointer is not set to the beginning of the deck. Terminate the punched deck with an EOP card or two EOS cards. If the deck is at the end of the job deck, however, the EOI is sufficient.

Example 3-8 illustrates placement of a punched deck (sometimes referred to as a SCOPE binary deck, an object module, or a relocatable binary deck) in the job deck.



Example 3-8. Loading From INPUT

Name Call Statement

The LGO and INPUT statements, as well as the compiler and assembler request statements, are examples of a special type of loader statement called the name call statement. Indeed, many of the statements you will be using in the control statement section are actually name call statements. That is, they are statements that summon the loader to load a program in your SCM and LCM fields and then execute it. Keyword statements, by contrast, request the system to perform some specific action without requiring a program to be loaded into the user SCM or LCM field. The system recognizes the SCOPE and loader keyword statements given in Table 3-3.

TABLE 3-3. KEYWORD STATEMENTS

SCOPE Keyword Statements		Loader Keyword Statements
ACCOUNT	MAP	EXECUTE
ALTER	MODE	LDSET
ATTACH	PASSWRD	LIBLOAD
AUDIT	PAUSE	LOAD
CATALOG	PURGE	NOGO
COMMENT	REDUCE	SLOAD
DISPOSE	REQUEST	
DMP	RFL	
DMPJSL	RTRVSIF	
DMPJT	SATISFY	
DMPLC	SEGLOAD	
DUMPF	STAGE	
EXIT	SUMMARY	
EXTEND	SWITCH	
FILE	SYSLIBE	
LABEL	TRANSF	
LIBRARY	VSN	
LIMIT		
LOADPF		

Any other control statement is a name call statement. The distinction is made between these statements and name call statements because no name call statement can be the same as one of these keywords. In interpreting a control statement, the system first determines whether the control statement is one of the keywords. If it is, it performs the requested action. If the statement is not a keyword statement, the system checks to see whether the statement is a name call statement. If the statement is not recognized as either a keyword statement or a name call statement, a control statement error occurs.

Name call statements divide into two classes: file name calls, of which LGO and INPUT are examples; and entry point calls, of which FTN, RUN, COBOL, and COMPASS are examples.

File Name Statement

This statement consists of the name of a file that contains an object program. The file name is optionally followed by parameters used by the program to be loaded.

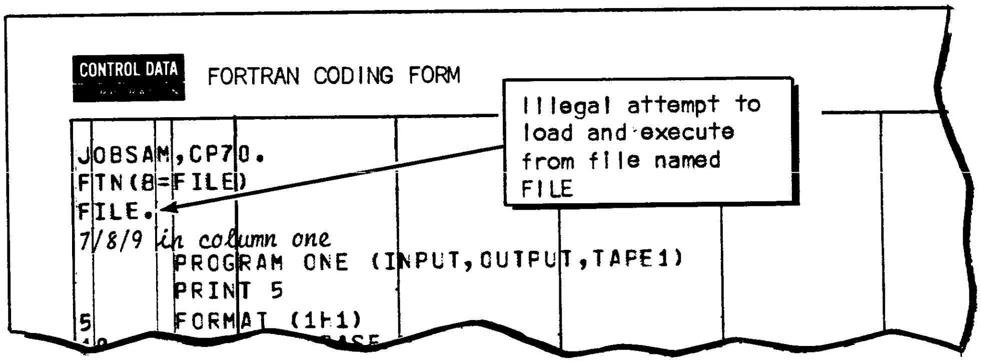
$\text{lfn}(p_1, p_2, p_3, \ldots, p_n)$

Thus, any program that resides on a file used by your job can be loaded into SCM and LCM and executed simply by referring to the name of the file.

The loader rewinds the file before loading from it. An exception to this rule is INPUT, which the loader does not rewind. Loading continues until the loader encounters EOI, EOP, or double EOS. A single EOS separates modules (that is, groups of loader tables that form a program) to be loaded.

In assigning a name to a file, there is nothing to prevent you from naming a file the same as one of the keyword statements. However, if you attempt to load and execute from the file by using a file name call, you will find that the statement is always interpreted as a keyword statement.

Example 3-9 illustrates a futile attempt to load from a file that has the same name as a keyword statement. In this case, the system interprets the FILE statement as a keyword statement, not as a name call statement. A file named FILE cannot be loaded in this way. The user receives an error message because the FILE statement is missing some required parameters. If the file had been given a name such as EXIT there would be no error indication.



```
JOBSAM,CP70.
FTN(B=FILE)
FILE.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
```

Illegal attempt to load and execute from file named FILE

Example 3-9. Illegal Keyword/Name Call Statement

Entry Point Name Statement

The locations at which execution can begin in a program are known as entry points. Compilers and assemblers form lists of available entry points when programs are compiled or assembled. These lists become significant when the programs are placed on system and user libraries because the user can name an entry point on a control statement and cause the program containing the entry point to be loaded and executed.

The entry point name statement consists of the name of the entry point (eptname) optionally followed by parameters to be passed to the loaded program.

eptname(p_1, p_2, p_3, . . . , p_n)

In interpreting the statement, SCOPE 2 first determines that it is not a keyword statement by checking eptname against the list of keywords, and then that it is not a file name by checking it against the list of files known to the job. Failing to find it in these two lists, the system assumes that the name is an entry point and searches for it on the libraries known to the job. To be recognized as an entry point, the name must have been declared as an entry point in some program on a library. If no library lists the name as an entry point, the system issues a control statement error.

All of the standard product set members (COMPASS, FTN, RUN, COBOL, etc.) and many of the SCOPE control statements (COPY, REWIND CONTENT, etc.) fall in the classification of entry point name statements. These programs are listed as entry points on the system nucleus library. They were placed there during installation of the SCOPE 2 system.

Remember that file name calls take precedence over entry-point name statements.

Thus, if you name a file FTN and then attempt to call the FTN compiler, the loader will attempt to load from a file named FTN instead of loading the FTN compiler from the library.

For entry-point name calls, loading and execution can never be separate options and cannot involve other loader statements.

LOADING OF OBJECT MODULES

The basic program unit produced by a compiler or assembler is an object module. It consists of several loader tables that define blocks, their contents, and address relocation information. An object module is sometimes referred to as a relocatable subprogram. When object modules are on a file, they can be called for loading and execution through a file name call. When they are on libraries, they can be called through entry-point names.

In either case, loading continues until an end-of-partition or double end-of-section is encountered. The load may consist of several modules separated by end-of-section delimiters. Additional loading of modules is required if the loaded modules contain references to entry points in other subprograms. These are known as external references. The process of locating and loading of modules containing the entry points is called satisfying of externals. The loader does not allow program execution to begin until all externals are satisfied.

Figure 3-1 illustrates a program consisting of the four object modules PROGA, PROGB, PROGC, and PROGD. PROGA is on the LGO file produced as a result of a compilation and assembly. The other modules are on libraries. (Loading from libraries is described in greater detail later in this section.) An LGO statement initiates the load sequence followed by program execution after all the object modules have been loaded.

A labeled common block is loaded below the program block of the first module that defines it. The first module is loaded immediately above the job communication area. This 100_8-word area is shown in Appendix B.

CORE IMAGE MODULES AND HOW THEY ARE LOADED

The core image module, also referred to as the loaded program or an absolute program, is the contents of the SCM field and the LCM field produced by the load operation.

When the core image module is copied onto a file, it is sometimes referred to as an overlay. A core image module on a file can be reloaded and executed at any time through a file name call. If the core image module is placed on a library, it can be called through an entry point name.

Because a core image module is the product of an object module load sequence, all of its external references have been satisfied. No manipulation of data is necessary. Loading is very swift and can consist of the one module only.

A core image module is also known as a "binary machine language program" because it requires no processing whatsoever before execution.

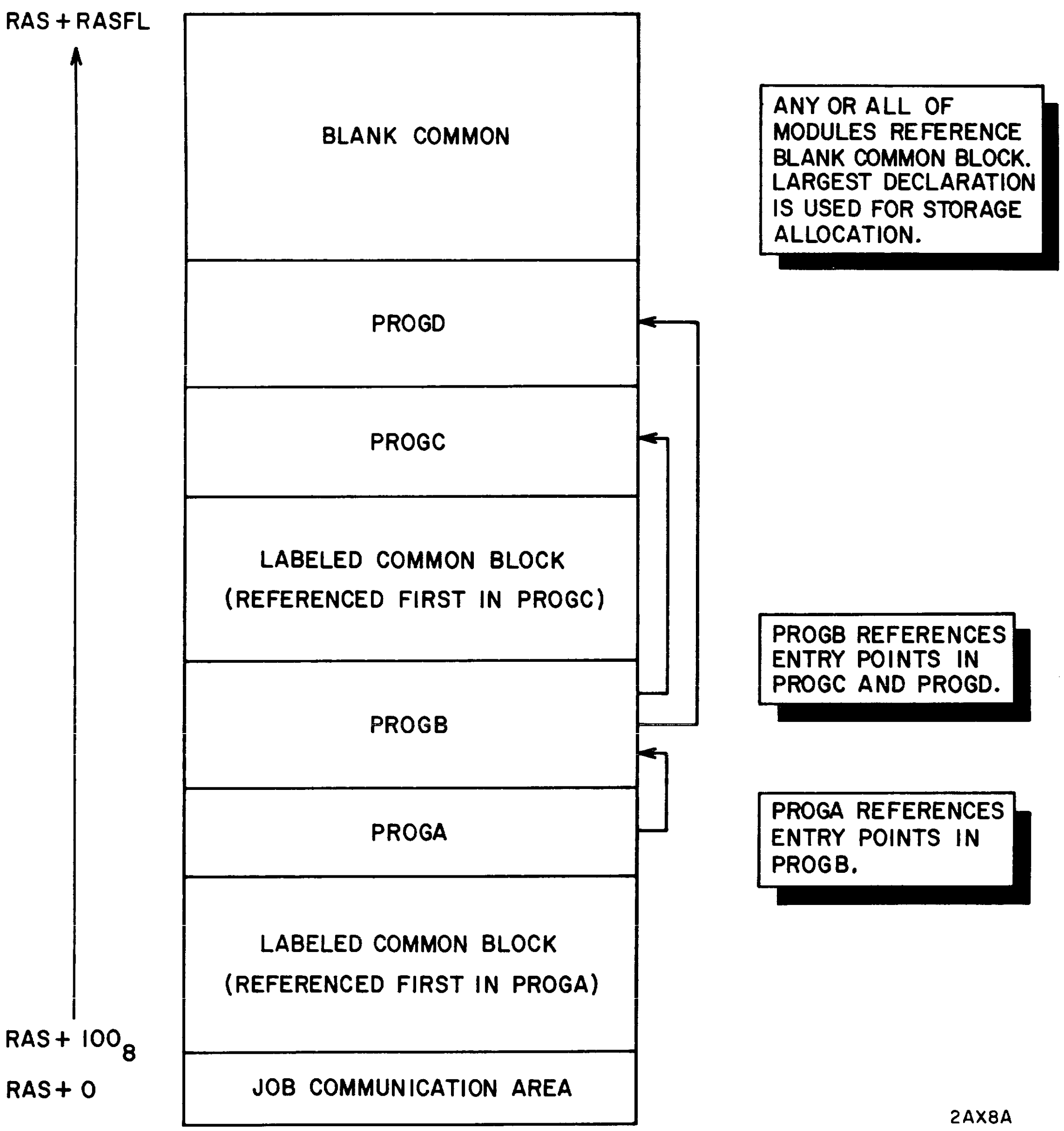

RAS + RASFL
BLANK COMMON
ANY OR ALL OF MODULES REFERENCE BLANK COMMON BLOCK. LARGEST DECLARATION IS USED FOR STORAGE ALLOCATION.
PROGD
PROGC
LABELED COMMON BLOCK (REFERENCED FIRST IN PROGC)
PROGB REFERENCES ENTRY POINTS IN PROGC AND PROGD.
PROGB
PROGA
PROGA REFERENCES ENTRY POINTS IN PROGB.
LABELED COMMON BLOCK (REFERENCED FIRST IN PROGA)
RAS + 100_8
RAS + 0
JOB COMMUNICATION AREA
2AX8A

Figure 3-1. Structure of Loaded Program

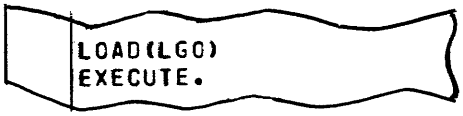
LOADING AND EXECUTION AS SEPARATE OPERATIONS

Suppose you want to load object modules from two different files and execute them as a single program. This is not possible using just the file name call. It becomes necessary to separate the load operation from the execute operation.

Another reason for separating the load operation from execution is if you wish to obtain a load map, but do not wish to execute the program.

Load and Execute

To illustrate how the load operation can be separated from execution, replace an LGO statement with the following two statements.



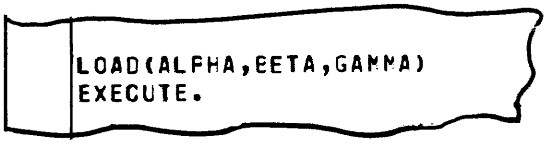
```
LOAD(LGO)
EXECUTE.
```

These two statements combined perform the same action as the LGO statement alone.

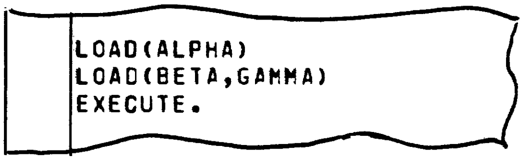
Load From Multiple Files and Then Execute

You can tell the loader to load from more than one file either by specifying all the files to be loaded on one LOAD statement or by using several LOAD statements prior to the EXECUTE statement.

The following are equivalent: either sequence causes files ALPHA, BETA, and GAMMA to be loaded and executed as a single program. Remember that the files must be local to the job.



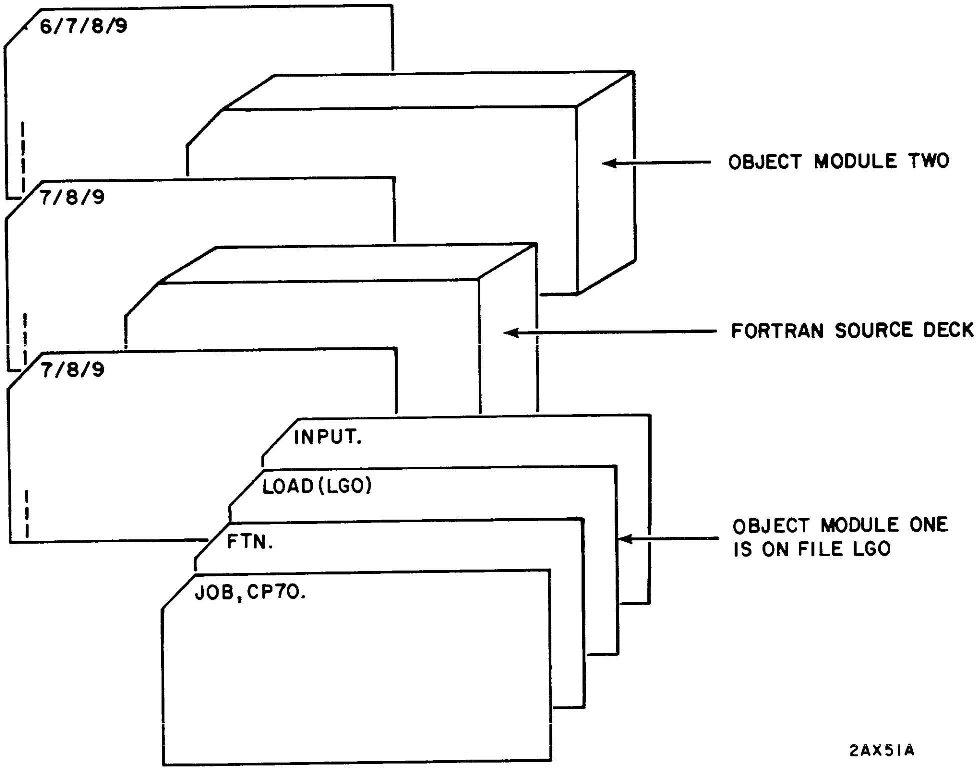
```
LOAD(ALPHA,EETA,GAMMA)
EXECUTE.
```



```
LOAD(ALPHA)
LOAD(BETA,GAMMA)
EXECUTE.
```

Another alternative is to use LOAD to load from one or more files and then call for the final load and execution through a file name call.

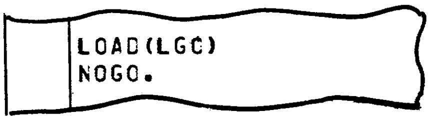
Example 3-10 illustrates this technique.



Example 3-10. Load From LGO and INPUT; Then Execute

Load With No Execution

If you want to load from a file but do not wish to immediately execute the program, replace the EXECUTE statement with a NOGO statement.



```
LOAD(LGC)
NOGO.
```

The NOGO tells the loader that no more loads are to take place and that execution is not to occur. If a map is requested, the loader generates the map but does not execute the loaded program.

Using NOGO To Generate Core Image Modules

In addition to using NOGO to inhibit program execution, NOGO can be used to write the loaded program onto a file as a single core image module. To do this requires the following NOGO control statement.

```
NOGO(lfn)
```

The file name is required for this application. One application of this technique permits execution of programs that would otherwise exceed available LCM. The following sequence illustrates this use.

```
LOAD(LGO)
NOGO(X)          Generate core image module
X.               Load core image and execute
```

LOAD SEQUENCES

The LOAD(LGO) statement followed by EXECUTE or NOGO is an example of a series of loader control statements known as a load sequence. Usually, a load sequence consists of a series of loader control statements terminated by an EXECUTE, a NOGO, or a file name call. The entry point name call is a special case because the load sequence consists of the entry point name call only. Other loader control statements are LDSET, LOAD, LIBLOAD, and SLOAD.

As illustrated in Figure 3-2, load sequences are not interpreted in the same way as SCOPE keyword statements. The set of statements making up the loader sequence resembles a single job step. The system accumulates all of the statements in a load sequence.

When the system encounters a terminating statement, the loader processes the entire sequence and satisfies any unsatisfied external references.

Note that for compatibility with previous systems, three SCOPE control statements, DMP, MAP, and REDUCE are recognized within a load sequence. Any other SCOPE control statement or entry point statement such as RUN(S) or COMPASS is illegal inside a load sequence and causes job termination with the message lfn FILE UNKNOWN.

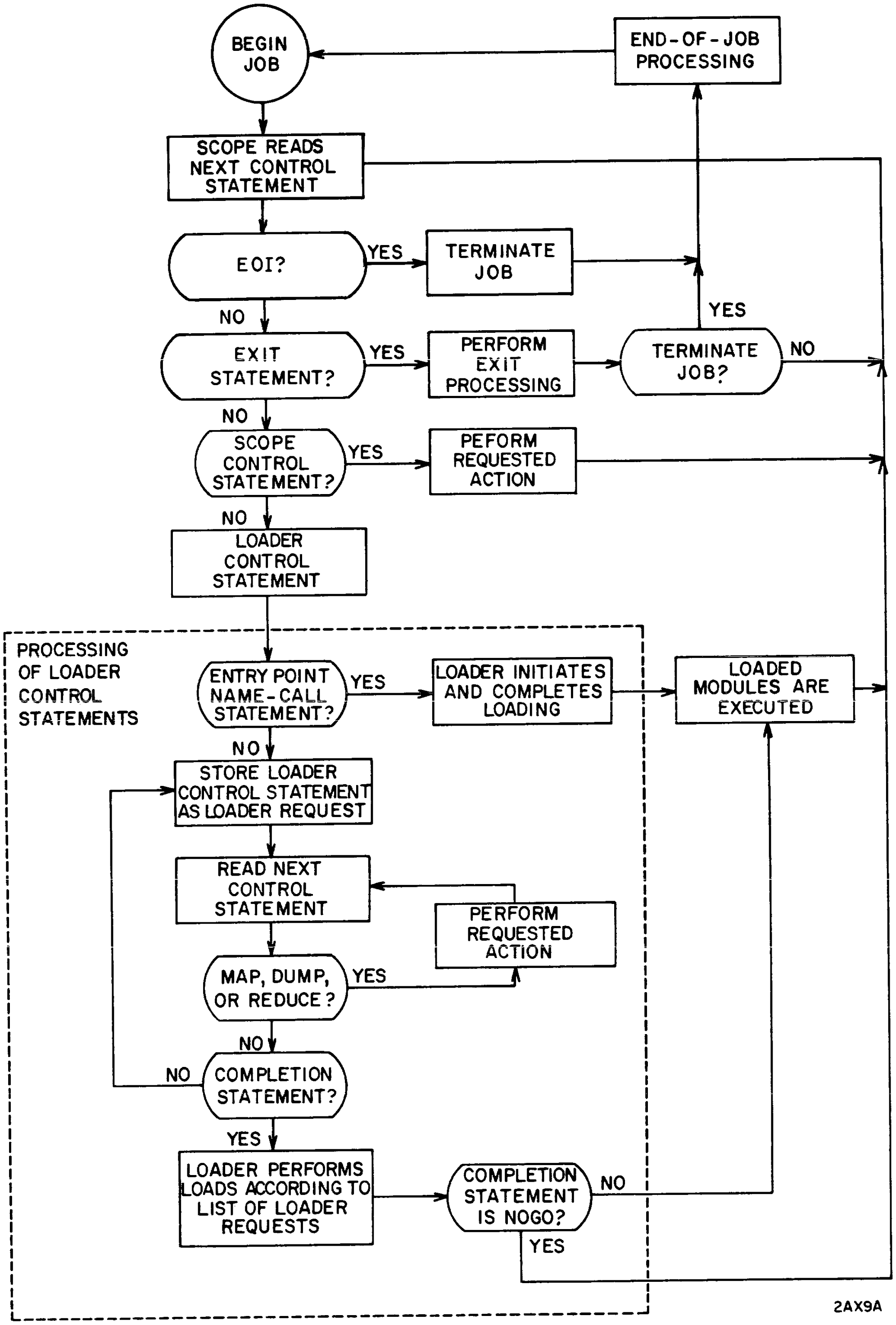
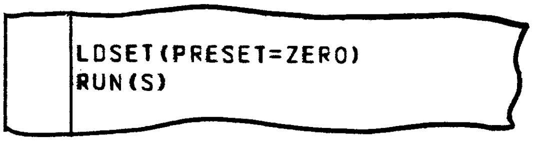

BEGIN JOB
END-OF-JOB PROCESSING
SCOPE READS NEXT CONTROL STATEMENT
EOI?
YES
TERMINATE JOB
NO
YES
EXIT STATEMENT?
YES
PERFORM EXIT PROCESSING
TERMINATE JOB?
NO
NO
SCOPE CONTROL STATEMENT?
YES
PEFORM REQUESTED ACTION
NO
LOADER CONTROL STATEMENT
PROCESSING OF LOADER CONTROL STATEMENTS
ENTRY POINT NAME-CALL STATEMENT?
YES
LOADER INITIATES AND COMPLETES LOADING
LOADED MODULES ARE EXECUTED
NO
STORE LOADER CONTROL STATEMENT AS LOADER REQUEST
READ NEXT CONTROL STATEMENT
PERFORM REQUESTED ACTION
MAP, DUMP, OR REDUCE?
YES
NO
NO
COMPLETION STATEMENT?
YES
LOADER PERFORMS LOADS ACCORDING TO LIST OF LOADER REQUESTS
COMPLETION STATEMENT IS NOGO?
NO
YES
2AX9A

Figure 3-2. Processing of Control Statements

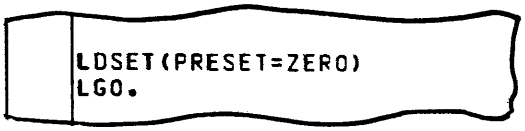
Thus, the following sequence is illegal:



```
LDSET(PRESET=ZERO)
RUN(S)
```

Begin load sequence
Entry-point name illegal
in loader sequence

However, the following is legal:



```
LDSET(PRESET=ZERO)
LGO.
```

Begin load sequence
File name completes
loader sequence

SELECTIVELY LOAD MODULES FROM FILES

Suppose a file has a number of object or core image modules on it and you wish to selectively load one or more modules. A file name call cannot be used because that would load all of the modules nor can an entry point name call be used because the file is not in library format and cannot be declared as a library. Similarly, the LOAD statement does not apply because all the modules are on the same file, not separate files. The statement that is needed is the SLOAD loader control statement.

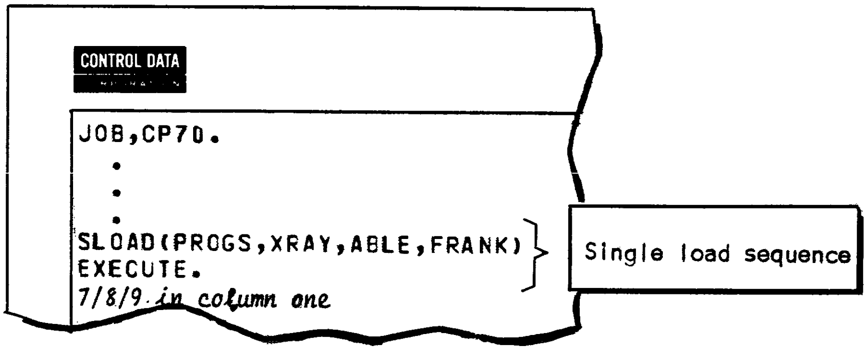
SLOAD(lfn, modname$_1$, modname$_2$, modname$_3$, . . . , modname$_n$)

SLOAD causes the loader to search the file for the modules named (modnames) by looking at the PRFX table that precedes each module. The PRFX table is a loader table created in all object modules and serves to identify the program to the loader. A module name is the program name assigned to the source program through the FORTRAN PROGRAM, SUBROUTINE, or FUNCTION statement; COBOL Identification Division; or COMPASS IDENT pseudo instruction. In addition to being the name of the program, this name is usually a primary entry point in the module.

Each module is a section on the file. Loading terminates upon encountering the end-of-section, end-of-partition, or end-of-information.

Remember that for execution to occur, you must complete the load sequence with EXECUTE or a file-name call. Also, since only selected modules are loaded from the file, references usually linked with the remaining modules will have to be satisfied from libraries.

Example 3-11 illustrates a load sequence that uses SLOAD to load modules ABLE, FRANK, and XRAY (in the sequence encountered on the file) from a file named PROGS. The file may contain many programs in addition to those to be loaded.



```
JOB,CP70.
  •
  •
  •
SLOAD(PROGS,XRAY,ABLE,FRANK)
EXECUTE.
7/8/9 in column one
```

Single load sequence

Example 3-11. Selective Load From a File by Program Name

SETTING LOAD SEQUENCE CHARACTERISTICS

The LDSET loader control statement is a general-purpose statement that allows you to set a number of characteristics for a specific load sequence. It has the following form:

LDSET(option$_1$, option$_2$, option$_3$, . . . , option$_n$)

Each option consists of a key which may or may not be equated to a parameter. You can use LDSET statements anywhere in the load sequence.

Options include the following:

ERR=level — Determines level (ALL, FATAL, or NONE) of error for which loader aborts load and does not initiate execution. The option is described under Setting Loader Error Conditions, page 12-8. The default is for FATAL errors to result in job termination.

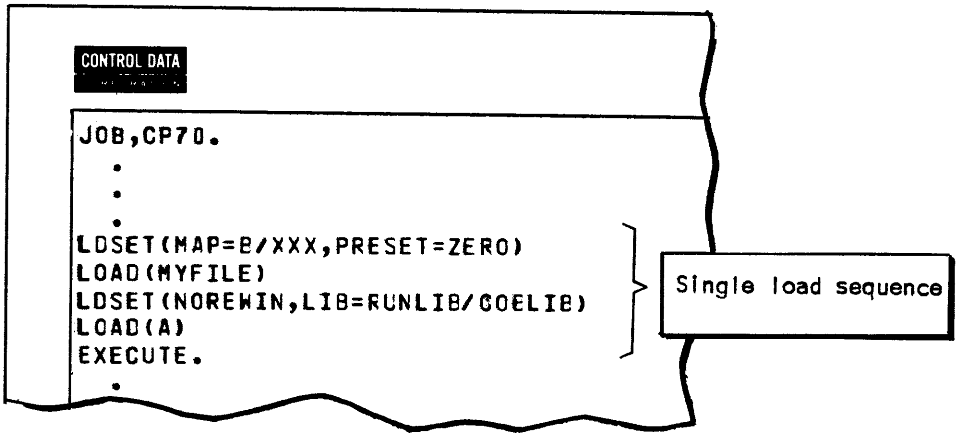
LIB=lfn$_1$/lfn$_2$/. . . /lfn$_n$ — Specifies list of library files to be searched when satisfying externals. This .is described under Using Libraries, page 3-21. The default is that the system library declared during compilation.

MAP=p/lfn — Specifies degree of load map produced and file on which map is to be written. This is described under Requesting Load Maps, page 12-17. The default map is determined by an installation or a MAP statement. The default file name is OUTPUT.

PRESET=value — Specifies that user SCM and LCM fields are to be preset to the indicated value. The default is for no presetting to occur. The option is described under Presetting Core Memory, page 4-8.

REWIND and NOREWIN — Specifies file positioning prior to load. Default is REWIND. This option is described under Rewinding of Load Files, page 4-15.

Example 3-12 illustrates a load sequence that contains two LDSET statements. The first statement requests core to be preset to zeros and a partial map on file XXX. The second requests NOREWIN, and for libraries RUNLIB and COBLIB to be used for satisfying externals.



CONTROL DATA

```
JOB,CP70.
   .
   .
   .
LDSET(MAP=B/XXX,PRESET=ZERO)
LOAD(MYFILE)
LDSET(NOREWIN,LIB=RUNLIB/COBLIB)
LOAD(A)
EXECUTE.
   .
```

Single load sequence

Example 3-12. Using LDSET Statements in Load Sequence

USING LIBRARIES

Several times we have alluded to the use of libraries as the source of object modules and core image modules to be loaded. Now let us consider what libraries are and how you determine which libraries are searched.

DEFINITION OF LIBRARY

A library is a collection of core image modules and/or object modules that can be efficiently accessed by the loader through a directory. Libraries are generated using the LIBEDT program. Libraries can be either system libraries or user libraries.

System Libraries

When the operating system was installed, several system libraries were placed on mass storage as permanent files. (A system library need not be attached to be used; it is not a permanent file in the usual sense.) The names of these libraries are maintained in a system library table. Often, the programmer will make use of these libraries without being aware of it. This is because the compilers all make external references to modules on system libraries. When the loader loads the object program, it knows to search the proper system library through a library declaration that the compiler inserted into the object program. As a result, object time programs for FORTRAN RUN declare libraries named RUNLIB and FORTRAN. Object programs for FORTRAN Extended declare the library named FORTRAN. COBOL object programs declare the COBLIB library. Generally a systems analyst can tell you the names of system libraries at your site.

The NUCLEUS Library

One system library file, the NUCLEUS, contains most of the operating system and product set members. The NUCLEUS library contains only core image modules. It contains no object modules and cannot be searched to satisfy externals. The contents of NUCLEUS are determined when the operating system is installed.

User Libraries

The user can create libraries, and direct the loader to satisfy externals from them instead of or in addition to the system libraries. A user library must be an input file for the job; it cannot be a magnetic tape file.

LIBRARY SETS

A library set is the list of libraries to be searched for entry point names and for satisfying externals. The user can declare that a library set be used for all subsequent loads in the job until further notice is given to the loader. This is called a global library set. The user can slso declare a second, temporary library set, that is, a list of libraries to be used for a single load sequence in addition to the global set. This is called the local library set. In either case, a library set can consist of both system and user libraries, but the number of user libraries is limited to five. The maximum size of a library set is ten libraries. NUCLEUS is not considered as part of a library set.

The Search For Entry Point Names

As previously note, the loader searches the library set for entry point names when the loader request consists of an entry point name. The search takes place after the system has eliminated the possibility that the name is either a keyword statement or a file name. The library sets are searched in the following order.

The global library set, if any

The local library set, if any

The NUCLEUS library

The Search For Externals

When the loader is attempting to satisfy external references encountered during an object module load, it searches library sets in the following order.

The global library set, if any

The local library set: The local library set automatically includes as a minimum the system library referenced by the compiler used. The RUN compiler references RUNLIB and FORTRAN, the FTN compiler references FORTRAN, the FTN compiler references FORTRAN, the COBOL compiler .references COBLIB.

NUCLEUS is not searched. It contains core image modules only and cannot be used for satisfying externals.

The loader does not attempt to satisfy externals until it encounters an EXECUTE, NOGO, or file name call in the load sequence. This is sometimes called load completion.

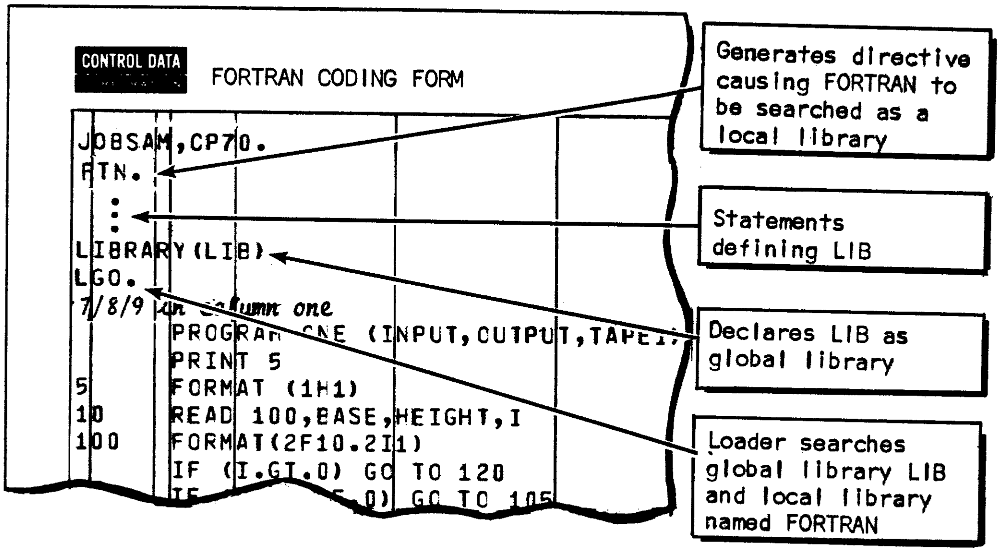
Defining the Global Library Set

With a LIBRARY statement, you can define your global library set, declare a new global set, or add to an existing global library set. Place the statement in your SCOPE control statements prior to the loader sequences in which you want to use the libraries. The LIBRARY statement is not a loader control statement and must not occur in a load sequence (for example, between LOAD and EXECUTE). The LIBRARY statement has the following format:

LIBRARY(libname$_1$, libname$_2$, libname$_3$, . . . , libname$_n$)

The library files are searched in the order listed. If a user library (local file) and a system library have the same name, the system library takes precedence. Ten libraries can be specified with a maximum of five of them being user libraries. It is possible to completely nullify a previous set and declare an empty set by using the LIBRARY statement with no parameters.

Example 3-13 illustrates a job that creates user library, LIB. The LIBRARY statement declares this library to be a member of the global library set to be used for satisfying externals when LGO is loaded. It takes precedence over FORTRAN, the system library automatically entered in the local library set.



FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
  :
LIBRARY(LIB)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF [illegible]E.0) GO TO 105
```

Generates directive causing FORTRAN to be searched as a local library

Statements defining LIB

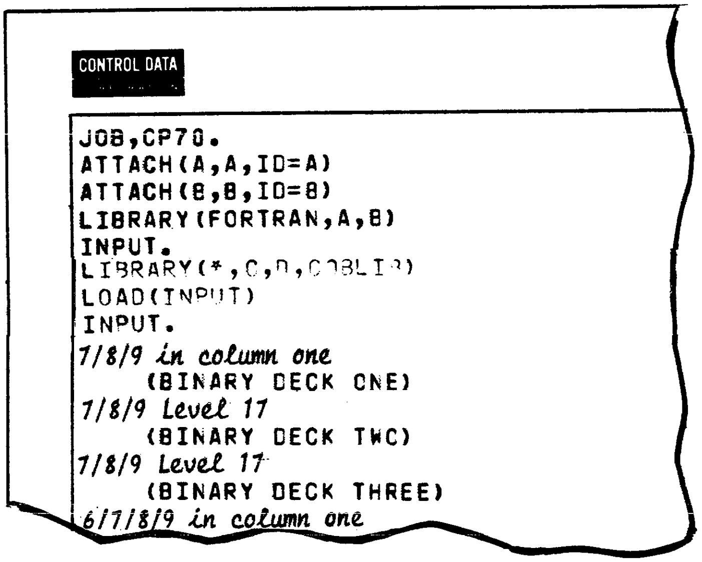
Declares LIB as global library

Loader searches global library LIB and local library named FORTRAN

Example 3-13. Defining Global Library

To retain a previous set as part of a new set, use an asterisk in place of a library name to indicate the point in the new list at which the previous global library set is to be inserted.

In Example 3-14, the first LIBRARY statement defines the global set as consisting of systém library FORTRAN and user libraries A and B. After execution of the deck on INPUT, the second LIBRARY statement defines the global set as consisting of system library FORTRAN; user libraries A, B, C, and D; and system library COBLIB.



CONTROL DATA

```
JOB,CP70.
ATTACH(A,A,ID=A)
ATTACH(B,B,ID=B)
LIBRARY(FORTRAN,A,B)
INPUT.
LIBRARY(*,C,D,COBLIB)
LOAD(INPUT)
INPUT.
7/8/9 in column one
     (BINARY DECK CNE)
7/8/9 Level 17
     (BINARY DECK TWC)
7/8/9 Level 17
     (BINARY DECK THREE)
6/7/8/9 in column one
```

Example 3-14. Combining New and Old Library Sets

Defining The Local Library Set

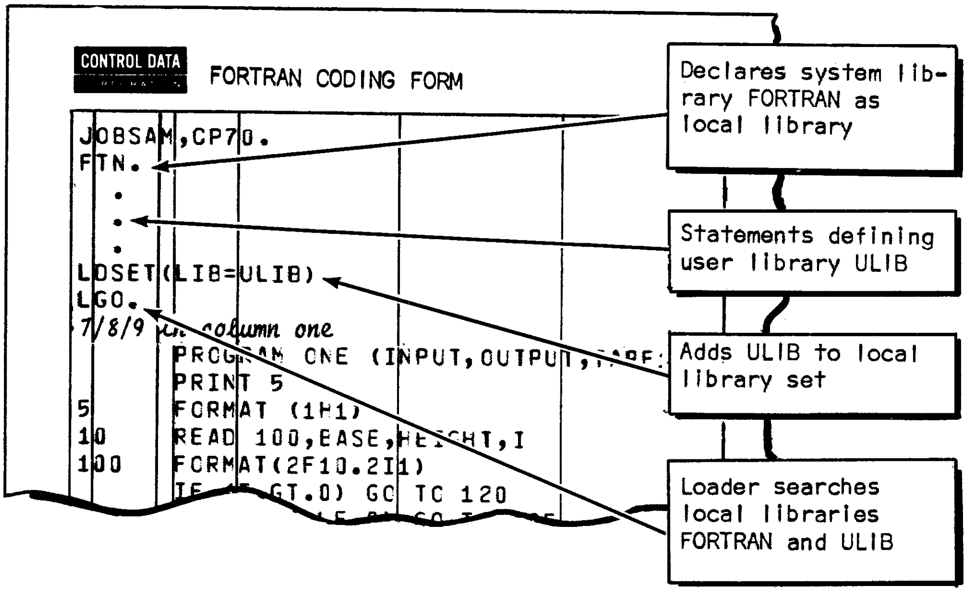
Use the LIB option on the LDSET loader control statement to declare a library local to the load sequence. Place the LDSET statement inside the load sequence for which the library is to be used. The LDSET statement is a loader control statement and either initiates or continues a load sequence.

```
LDSET(LIB=libname1/libname2/libname3/libnamen)
```

Library files can be either system or user libraries.

Any library declared by a compiler or by a previous LDSET statement in the load sequence is added to the list of local files. Clear the local set by omitting any parameters. This also clears any compiler library declarations (for example, it clears FORTRAN which is declared by the FTN compiler). If no global library set has been declared and the user clears the local set, there is no way for externals to be satisfied. No libraries are available to be searched. As previously noted, LDSET has many optional parameters of which LIB is only one. The LIB=parameter can occur in any order in the LDSET statement.

In Example 3-15, there is no global library set; the local library set consists of FORTRAN and ULIB.



```
JOBSAM,CP70.
FTN.
  .
  .
  .
LDSET(LIB=ULIB)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
```

Declares system library FORTRAN as local library

Statements defining user library ULIB

Adds ULIB to local library set

Loader searches local libraries FORTRAN and ULIB

Example 3-15. Defining a Local Library

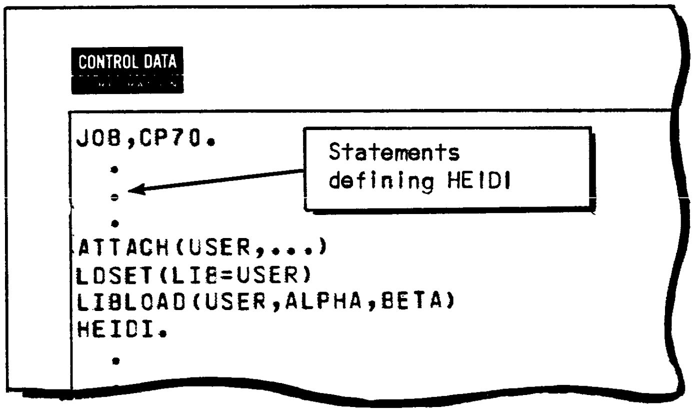
LOADING DIRECTLY FROM LIBRARIES

Now that you know how to declare libraries in global and local library sets, you can tell the loader to load from them. One loader control statement that can be used is the LIBLOAD statement. LIBLOAD loads one or more modules from a library in a global or local library set. Modules are specified through entry-point names.

LIBLOAD(libname, eptname_1, eptname_2, eptname_3, . . . , eptname_n)

The first parameter must name the library containing the entry points. If one module contains more than one of the entry points, fewer modules are loaded than entry points.

Example 3-16 illustrates a load sequence containing a library load request. Library USER contains entry point names ALPHA and BETA. Notice that a statement that terminates the load sequence must follow the LIBLOAD statement before execution can occur. In this case, the file name statement for HEIDI completes loading and begins execution.


CONTROL DATA

```
JOB,CP70.
  .
  .
  .
ATTACH(USER,...)
LDSET(LIB=USER)
LIBLOAD(USER,ALPHA,BETA)
HEIDI.
  .
```

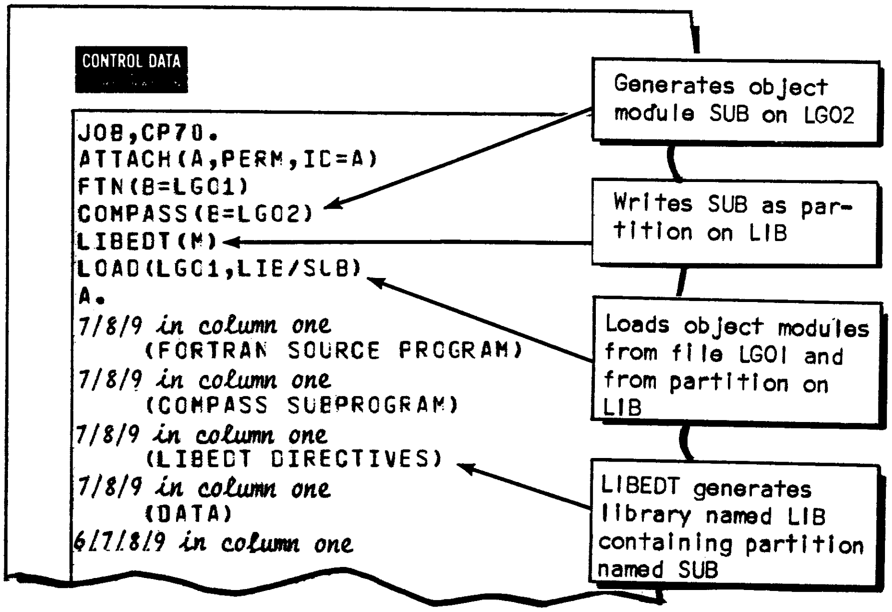
Statements defining HEIDI

Example 3-16. Direct Load From Library Using LIBLOAD

LOADING PARTITIONS FROM LIBRARIES

LIBEDT allows each partition on a library to be named. The name is either the program name for the object module or core image module in the partition, or is a name assigned by the creator of the library. The LOAD and SLOAD statements both provide options for loading a partition by name from a given library. Instead of entering a file name on the LOAD or SLOAD statement, enter a library name and a partition name in the form libname/pname.

LOAD allows loading of modules from both libraries and files, as shown in Example 3-17.



```
JOB,CP70.
ATTACH(A,PERM,IC=A)
FTN(B=LGO1)
COMPASS(B=LGO2)
LIBEDT(M)
LOAD(LGO1,LIB/SUB)
A.
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
7/8/9 in column one
     (COMPASS SUBPROGRAM)
7/8/9 in column one
     (LIBEDT DIRECTIVES)
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

Generates object module SUB on LG02

Writes SUB as partition on LIB

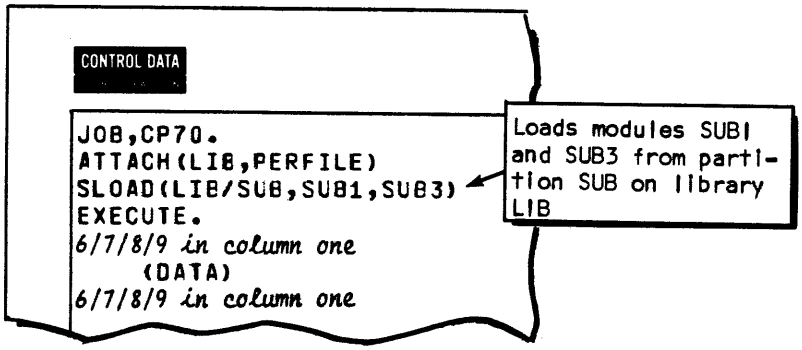
Loads object modules from file LGO1 and from partition on LIB

LIBEDT generates library named LIB containing partition named SUB

Example 3-17. Load Partition From Library Using LOAD

With SLOAD it is possible to load one or more modules from the partition indicated (see Example 3-18).

Remember that LOAD and SLOAD are loader control statements and can be used only in load sequences. Note that SLOAD allows loading of several modules from a partition. For LOAD, if the partition contains more than one module, only the first module is loaded.



```
JOB,CP70.
ATTACH(LIB,PERFILE)
SLOAD(LIB/SUB,SUB1,SUB3)
EXECUTE.
6/7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

Loads modules SUB1 and SUB3 from partition SUB on library LIB

Example 3-18. Load Partition From Library Using SLOAD

PROGRAM AND JOB OPTIONS 4

SCOPE 2 provides several options for overriding system defaults that normally affect every load during processing of the job. For example, a user can specifically designate the size of the SCM and LCM fields instead of having the loader assign a field length or he can tell the loader to always set the SCM field or the LCM field to a predetermined value before loading. This section describes how these options interrelate with the system-defined default values and tells why a user may want to use them.

Because none of these options is required for normal job processing, the user may wish to omit this section and continue with Section 5.

USING CORE MEMORY

User SCM and LCM field sizes either are automatically determined by the loader, or are specifically defined by the user.

AUTOMATIC CORE MEMORY MANAGEMENT

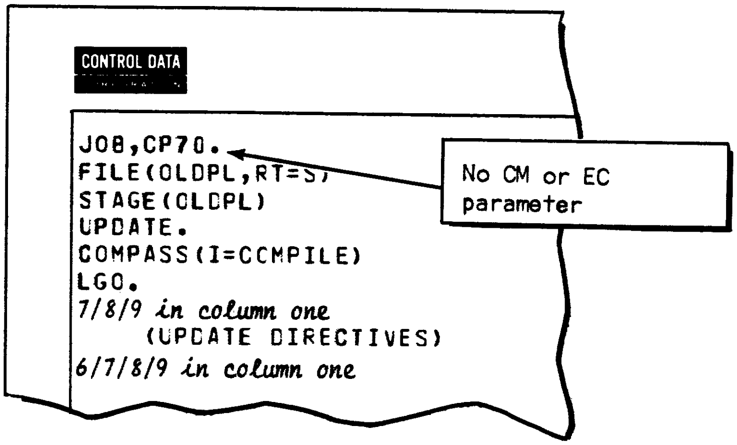
The most common, most efficient, and easiest way of managing SCM and LCM field lengths is by using the automatic mode. This mode (also called dynamic field assignment and system controlled mode) is the system default. It is initially in effect for small core memory if the CM parameter is omitted from the job identification statement and is in effect for large core memory if the EC parameter is omitted. Automatic mode is overridden for the applicable core type if CM or EC is specified on the job statement, or if an RFL control statement is used.

Automatic mode applies separately to SCM field size and LCM field size. One can be user controlled while the other can be in automatic mode.

Example 4-1 illustrates a job that consists of five job steps. COMPASS is chosen for the illustration because of its LCM requirements. The FILE and STAGE statements involve the control statement processor only, which uses about 1000_8 words of SCM and does not need any LCM. UPDATE executes in the user field length in two passes, each of which has different SCM requirements. COMPASS Version 2 executes in the user field length. It has low SCM requirements and high LCM requirements. These requirements vary with each of the three passes. The final step is object program execution (LGO). In this example, the object program requires both SCM and LCM. The SCM requirements are increased when an overlay load occurs. Requirements for object programs depend entirely on the source program.

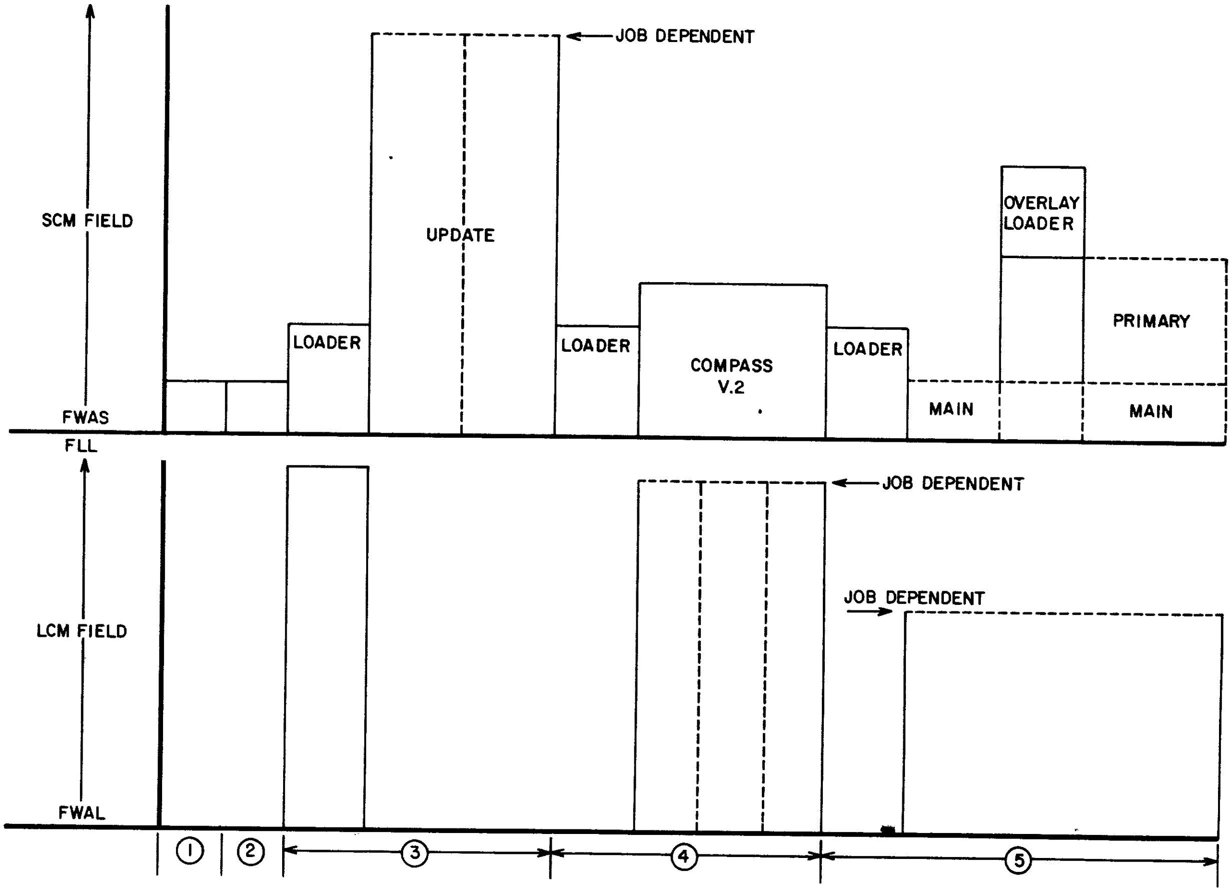
When automatic core memory management is in effect, the only limitation on the size of the SCM field length for a job step is the amount of SCM available to all users in the system. This is 60000_8 for 32K systems and 160000_8 for 65K systems. Any time the loader is called, the amount of core assigned to the job is re-evaluated.

LCM limits are slightly more complex because system I/O buffers for a job are in LCM but are not in the user LCM field length. The sum of the memory used for buffers and for LCM field length (FLL) cannot exceed an installation parameter that is normally set at 400000_8 for 256K systems and at 1400000_8 for 512K systems.



```
JOB,CP70.
FILE(OLDPL,RT=S)
STAGE(OLDPL)
UPDATE.
COMPASS(I=COMPILE)
LGO.
7/8/9 in column one
     (UPDATE DIRECTIVES)
6/7/8/9 in column one
```

No CM or EC parameter



Example 4-1. Job Using Automatic Core Memory Management

After each load, the system increases or decreases the field lengths to meet the changed requirements. Each call for the loader results in loader execution in SCM (about 3000_8 words). The loader uses tables in system buffers in LCM, but these buffers are not in the user LCM field length and do not affect the amount of LCM required for field length.

USER CONTROLLED CORE MEMORY MODE

Although user controlled memory management is provided primarily for compatibility with other systems, certain types of programs cannot execute under automatic mode. For these programs, the user must explicitly specify memory requirements.

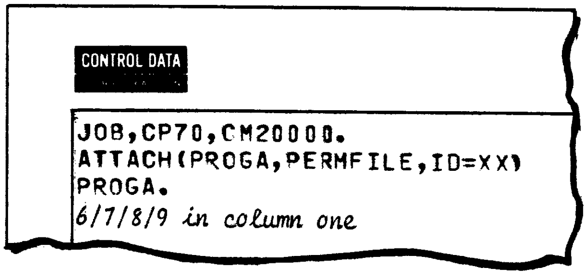
Automatic mode cannot be used in special cases such as when the program legally contains a reference to a core location that exceeds the highest address loaded with the core module (for example, it references a blank common block that is not known to the loader). Only COMPASS allows this type of condition to occur.

Controlling SCM

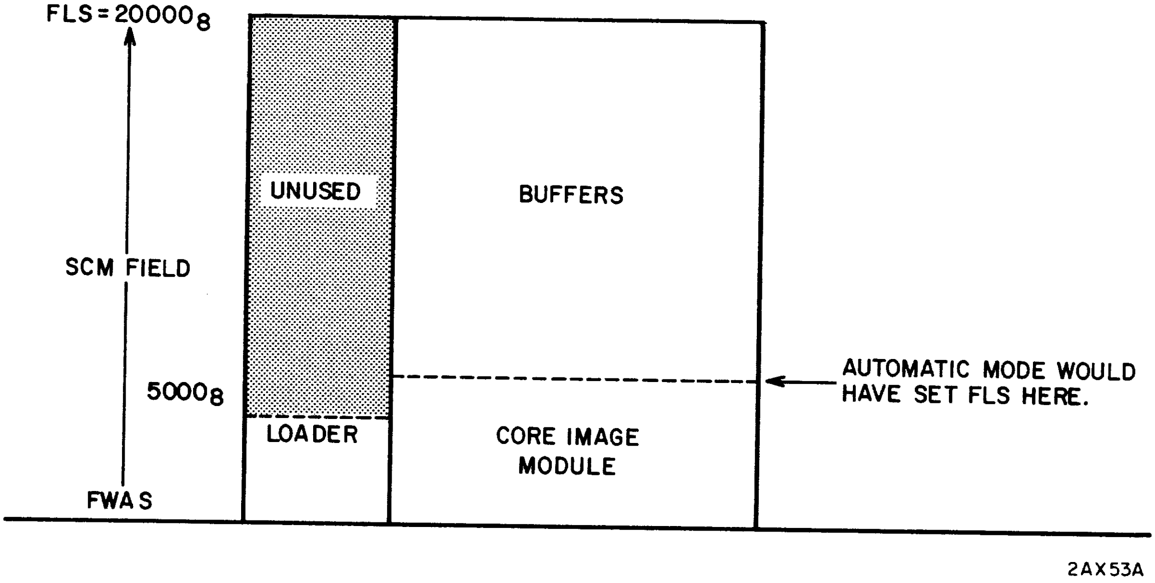
The amount of SCM assigned for the job can be determined either by the CM parameter on the job identification statement or by a parameter on the RFL control statement.

CM parameter: On the job identification statement, enter the amount of SCM assigned to the job as an octal value prefixed by the letters CM. The amount of SCM assigned the job is the exact amount specified by the CM parameter. Unlike SCOPE 3.4, no roundup of the value occurs. Any attempt to load a program beyond this fixed amount of SCM causes job termination. SCM cannot be set below 1000_8, the amount required for interpreting control statements nor can it be set above an installation defined value.

Example 4-2 illustrates a job using the CM parameter. In this example, the program being loaded occupies 5000_8 words of SCM and no LCM. However, the program references addresses in SCM above 5000_8 that the loader is unaware of because they are initially empty and were not generated as part of the core image module to speed up loading. This technique is described in the COMPASS Reference Manual.



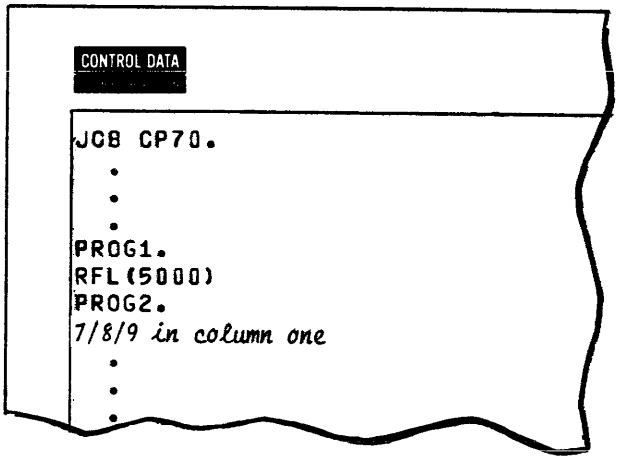
```
JOB,CP70,CM20000.
ATTACH(PROGA,PERMFILE,ID=XX)
PROGA.
6/7/8/9 in column one
```



Example 4-2. Using the CM Parameter to Control SCM

RFL statement: Use the RFL statement to change from automatic control of SCM to user control or to change the amount of SCM assigned to your job. Enter the amount of memory needed as an octal value. Place the statement before the load sequence to be affected. The RFL statement cannot occur within a load sequence.

Example 4-3 illustrates a job that initially acquires 30000 words of SCM through dynamic allocation and then reduces the requirement to 5000 words before loading the second program.



```
JOB CP70.
  •
  •
  •
PROG1.
RFL(5000)
PROG2.
7/8/9 in column one
  •
  •
  •
```

Example 4-3. Using the RFL Statement to Control SCM

Controlling LCM

Although some programs do not have any LCM requirements, others make heavy use of LCM. The FORTRAN Extended, RUN, COBOL, and COMPASS languages all provide for using LCM.

Compiler or Assembler	Language Element That Uses LCM
FORTRAN Extended and RUN	LEVEL statement where level is 2 or 3
COBOL	SECONDARY STORAGE section
COMPASS	USELCM pseudo instruction

If LCM is under user control, remember to schedule LCM for object programs that use these statements.

Set the amount of LCM assigned to your job (excluding LCM buffers) through the EC parameter on the job identification statement or by an RFL parameter.

EC parameter: On the job identification statement, enter the amount of LCM needed in octal thousands prefixed by the characters EC. The COBOL compiler requires at least 40000_8 words of LCM; the COMPASS assembler requires at least 26000_8 words.

```
JOBNAME,CP70,EC16.
```

The preceding job identification statement sets LCM to a fixed value of 16000_8 words.

RFL statement: Use the RFL statement to declare user control of LCM or to change a previous field length assignment. This statement cannot occur within a load sequence (for example between a LOAD statement and an EXECUTE statement).

Enter the amount of LCM in octal thousands prefixed by the characters L=. There is no minimum for LCM; it can be 0.

```
RFL(L=16)
```

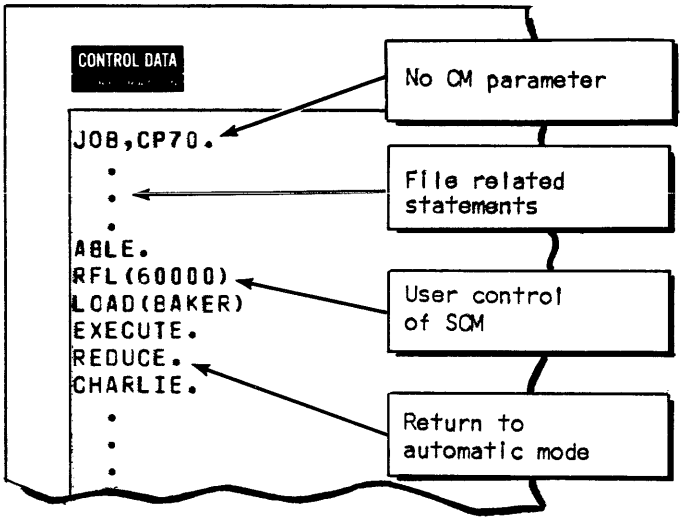
The preceding statement sets LCM to a fixed value of 16000_8.

RETURNING TO AUTOMATIC MODE

Return the job to automatic mode by placing a REDUCE statement in the control statements as soon as possible.

To return both fields to automatic mode, use REDUCE with no parameters. Otherwise, use an S to indicate SCM or an L to indicate LCM.

In Example 4-4, automatic memory management is in effect for the loading and execution of program ABLE. The RFL statement sets SCM field length to 60000$_8$ for the loading and execution of BAKER. When BAKER has terminated, the user specifies return to automatic memory management through a REDUCE statement. CHARLIE is then loaded and executed under memory management.



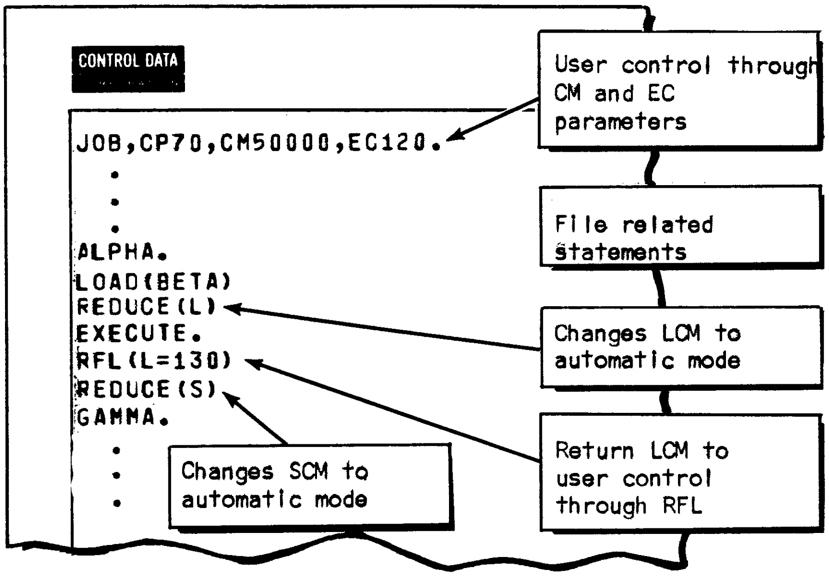
```
JOB,CP70.
 .
 .
 .
ABLE.
RFL(60000)
LOAD(BAKER)
EXECUTE.
REDUCE.
CHARLIE.
 .
 .
 .
```

Example 4-4. Mixed Mode Control of SCM

In Example 4-5, SCM and LCM management are initially under user control through the CM and EC parameters. Program ALPHA can reference the assigned field lengths but must not initiate any loads that would exceed these limits. In the load sequence for BETA, however, the management of LCM is returned to automatic through a REDUCE(L) statement. Note that this statement is a SCOPE control statement. It is allowed to occur within a load sequence for compatibility with previous operating systems. The preferred location for the REDUCE statement is before the LOAD(BETA) statement which begins the load sequence.

Following execution of BETA, the user again assumes control of LCM through an RFL statement requesting 130000$_8$ words of LCM. Finally, for the loading of GAMMA, the user returns SCM control to automatic with the REDUCE(S) statement.

Unlike SCOPE 3.4, SCOPE 2 allows the value specified on the RFL statement to exceed the value specified by the corresponding EC or CM parameter on the job identification statement.



```
JOB,CP70,CM50000,EC120.
 •
 •
 •
ALPHA.
LOAD(BETA)
REDUCE(L)
EXECUTE.
RFL(L=130)
REDUCE(S)
GAMMA.
 •
 •
 •
```

Example 4-5. Mixed Mode Control of Both SCM and LCM

PRESETTING CORE MEMORY

When the loader determines the SCM and LCM field lengths (either dynamically or under user-control), it has the option of setting the fields to an installation specified value (initializing core) or of not setting the fields. For this discussion, let us assume that the installation default specifies no presetting of core.

One option available is telling the loader to preset the field length for each load sequence through the use of the LDSET PRESET option. Enter one of the parameters given in Table 4-1 prefixed by the characters PRESET=. One reason to preset core is to ensure that any read reference preceding a store into blank common will return zeros. In this case, use LDSET(PRESET=ZERO).

For NGINF, each location contains its address in the lower bits. For example, if locations RAS + 1000_8 and RAS + 1001_8 are unused, they are set to

```
          4000  0000  0000  0000  1000
and
          4000  0000  0000  0000  1001
```

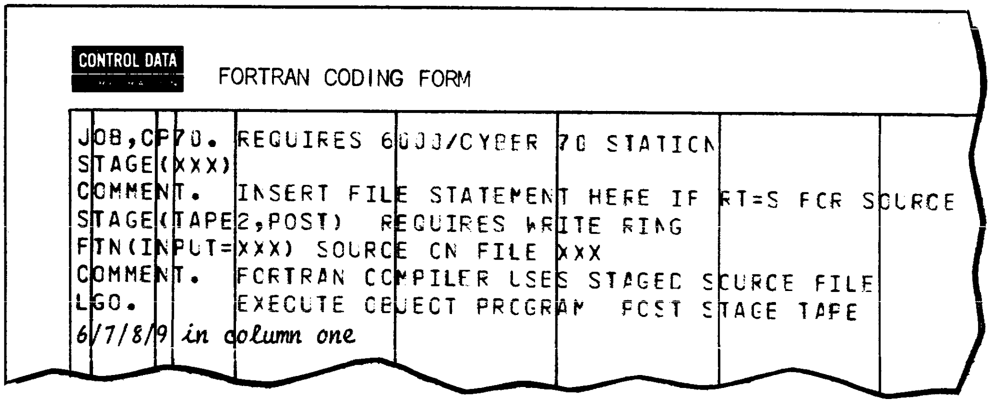
For SCM, addr is a maximum of 18 bits. For LCM, it is a maximum of 21 bits.

TABLE 4-1. PRESET OPTIONS

Option	Octal Preset Value
NONE	No presetting
ZERO	0000 0000 0000 0000 0000
ONES	7777 7777 7777 7777 7777
INDEF	1777 0000 0000 0000 0000
INF	3777 0000 0000 0000 0000
NGINDEF	6000 0000 0000 0000 0000
NGINF	4000 0000 0000 addr
ALTZERO	2525 2525 2525 2525 2525
ALTONES	5252 5252 5252 5252 5252

INSERTING COMMENTS IN THE PROGRAM LISTING

Comments help provide a history of a job. Insert special comments or remarks after the terminator on any control statement. Such remarks are useful in interpreting program listings, or in providing general information. Example 4-6 illustrates some control statements that include comments.



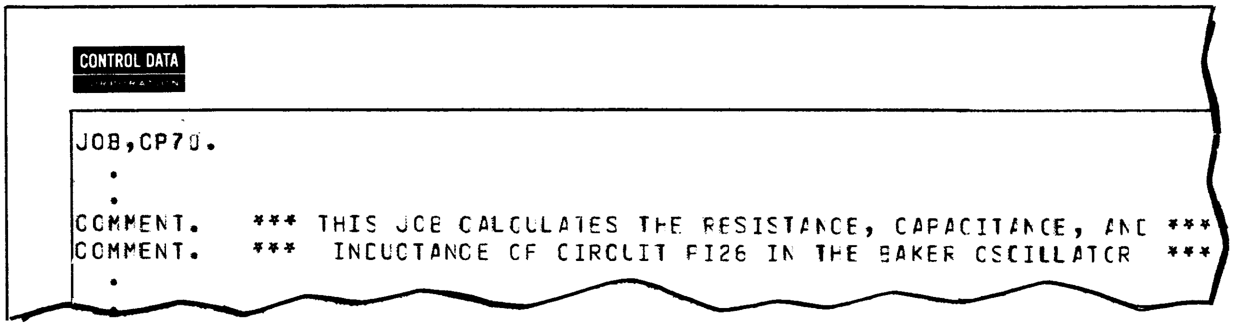
CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70. REQUIRES 6000/CYBER 70 STATION
STAGE(XXX)
COMMENT.  INSERT FILE STATEMENT HERE IF RT=S FOR SOURCE
STAGE(TAPE2,POST)  REQUIRES WRITE RING
FTN(INPUT=XXX) SOURCE ON FILE XXX
COMMENT.  FORTRAN COMPILER USES STAGED SOURCE FILE
LGO.      EXECUTE OBJECT PROGRAM  POST STAGE TAPE
6/7/8/9 in column one
```

Example 4-6. Comments in Dayfile Listing

Another way to introduce comments is through a COMMENT control statement. Any remarks can occupy columns 9 through 80 following the period after COMMENT. Comments can include any characters except the double colon which has special significance because it may be interpreted as a 12-bit zero byte. Blanks, periods, and other punctuation are allowed.

Remarks following COMMENT. are printed in the dayfile and the first 50 characters of the statement including COMMENT. are displayed to the operator on the console screen at the originating station. If you have a message to the operator, however, use PAUSE rather than COMMENT because comments from the COMMENT statement may not be displayed at the console long enough for the operator to see them. If a comment is too lengthy to fit on the line of coding, it can be continued on a second and subsequent COMMENT statements (see Example 4-7).

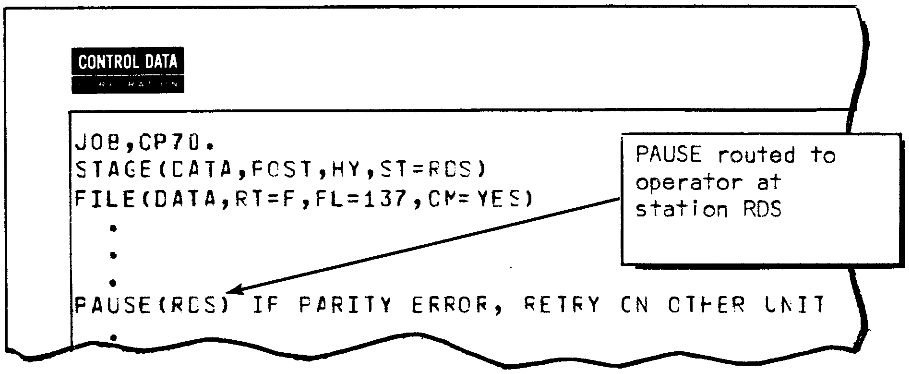


```
JOB,CP70.
  .
  .
COMMENT.    *** THIS JOB CALCULATES THE RESISTANCE, CAPACITANCE, AND ***
COMMENT.    ***  INDUCTANCE OF CIRCUIT FI26 IN THE BAKER OSCILLATOR  ***
  .
```

Example 4-7. Comment Two Lines Long

PAUSE FOR OPERATOR ACTION

The PAUSE statement allows the user to give an operator at one of the stations specific directions regarding the processing of your job. A PAUSE statement causes a message to remain on the console screen until it is acknowledged by the operator. Meanwhile, the job has halted processing. The operator acknowledges the message and restarts the job by typing GO, unless comments on the PAUSE statement direct him to DROP or KILL the job. Example 4-8 shows a PAUSE statement that tells the operator at station RDS to change tape units and rerun the job if any tape parity errors are encountered. If the user omits the station/terminal identifier, the station that originated the job is assumed. In the example, (RDS) would be replaced by a period. The message on the PAUSE statement has a maximum length of 50 characters and cannot be continued on a second statement.



```
JOB,CP70.
STAGE(DATA,POST,HY,ST=RDS)
FILE(DATA,RT=F,FL=137,CM=YES)
 .
 .
 .
PAUSE(RDS) IF PARITY ERROR, RETRY ON OTHER UNIT
 .
```

PAUSE routed to operator at station RDS

Example 4-8. Directing the Operator Through a PAUSE Statement

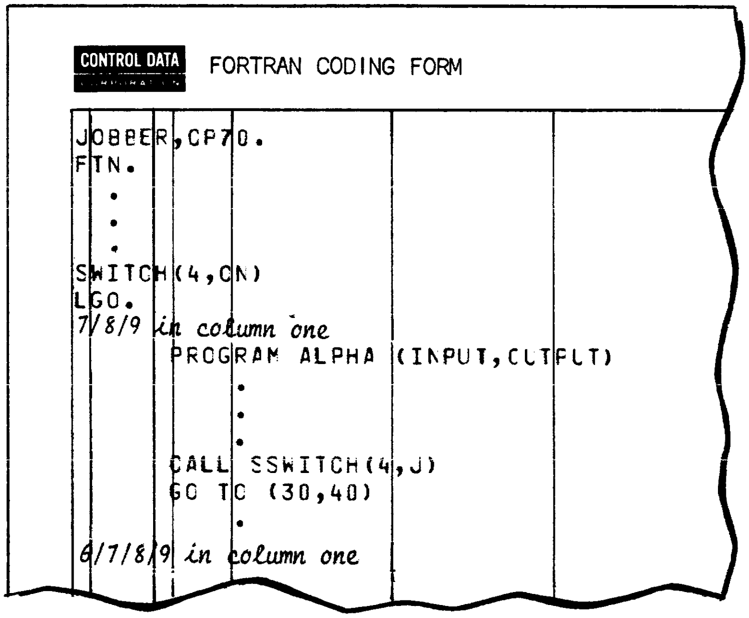
SETTING PROGRAM SWITCHES

The Job Communication Area (Appendix B) contains, in the lower part of word 0, six bits that are accessible through control statements and through the FORTRAN language. These are the pseudo sense switches. They logically simulate manual switches that were physically present on very early computer models. On early models the operator had to manually set or clear the switches. With the pseudo switches, however, the programmer sets or clears them through SWITCH statements.

SWITCH(n,$\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}$)

Switches are numbered 1 through 6. All of the switches are initially off. To turn a switch on or off, specify the switch number followed by ON or OFF, respectively. Otherwise, omit the setting (ON or OFF) and by doing so specify that the switch position is to be alternated. That is, if it is off, it is turned on and vice versa.

Example 4-9 illustrates a FORTRAN job that tests the status of sense switch 4.

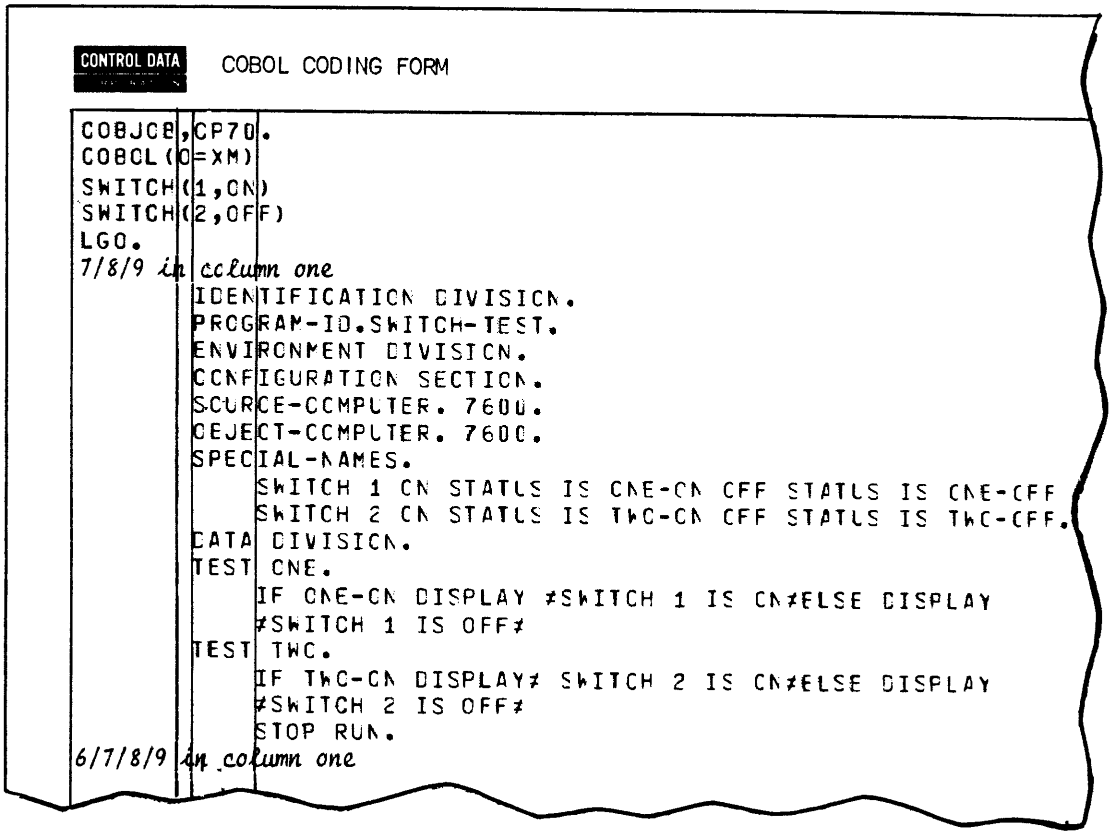


CONTROL DATA FORTRAN CODING FORM

```
JOBBEER,CP70.
FTN.
 .
 .
 .
SWITCH(4,ON)
LGO.
7/8/9 in column one
      PROGRAM ALPHA (INPUT,OUTPUT)
        .
        .
        .
      CALL SSWITCH(4,J)
      GO TO (30,40)
        .
6/7/8/9 in column one
```

Example 4-9. Using the SWITCH Statement

Example 4-10 illustrates a COBOL job that tests the status of sense switches 1 and 2.



CONTROL DATA
COBOL CODING FORM

```
COBJCB,CP70.
COBOL(O=XM)
SWITCH(1,ON)
SWITCH(2,OFF)
LGO.
7/8/9 in column one
       IDENTIFICATION DIVISION.
       PROGRAM-ID.SWITCH-TEST.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. 7600.
       OBJECT-COMPUTER. 7600.
       SPECIAL-NAMES.
           SWITCH 1 ON STATUS IS ONE-ON OFF STATUS IS ONE-OFF
           SWITCH 2 ON STATUS IS TWO-ON OFF STATUS IS TWO-OFF.
       DATA DIVISION.
       TEST ONE.
           IF ONE-ON DISPLAY ≠SWITCH 1 IS ON≠ELSE DISPLAY
           ≠SWITCH 1 IS OFF≠
       TEST TWO.
           IF TWO-ON DISPLAY≠ SWITCH 2 IS ON≠ELSE DISPLAY
           ≠SWITCH 2 IS OFF≠
           STOP RUN.
6/7/8/9 in column one
```

Example 4-10. COBOL Test of Sense Switches

PROCESSING INTERDEPENDENT JOBS

Sometimes the user is faced with the problem that he must have the output from one job or the job must satisfy some condition before he can run another job, but would like to submit both jobs at the same time. SCOPE allows several related jobs to be submitted and delays the processing of a job until one or more criteria are met. A user controls the progress of the related jobs through the combined use of the TRANSF control statement and the dependency parameter on the job identification statement.

JOB DEPENDENCY PARAMETER

For each job in the dependency string, whether it supplies a requirement of a waiting job or whether it is a job waiting for the requirement, enter the Dym parameter on the job identification statement.

y is two alphabetic characters (A through Z) that identify the jobs in the string. That is, it

1. Provides uniqueness for the events in the system. The event name is formed by taking the first five characters of the job name and then appending the string identifier.

2. Allows the operator using a console command to drop all of the dependent jobs in a string.

m is a 1 or 2 octal digit count (0-77) of the dependencies the job must have fulfilled before it can begin processing. For the first job to be processed in the string, m must be 0, that is, Dy0.

TRANSF CONTROL STATEMENT

Enter a TRANSF statement in the control statement sequence each time a job satisfies a criterion needed by another job.

TRANSF(job_1, job_2, . . . , job_n)

A job can contain several TRANSF statements. Also, with one TRANSF the user can signal several jobs concurrently. Remember, the last job in the string cannot contain any TRANSF statements.

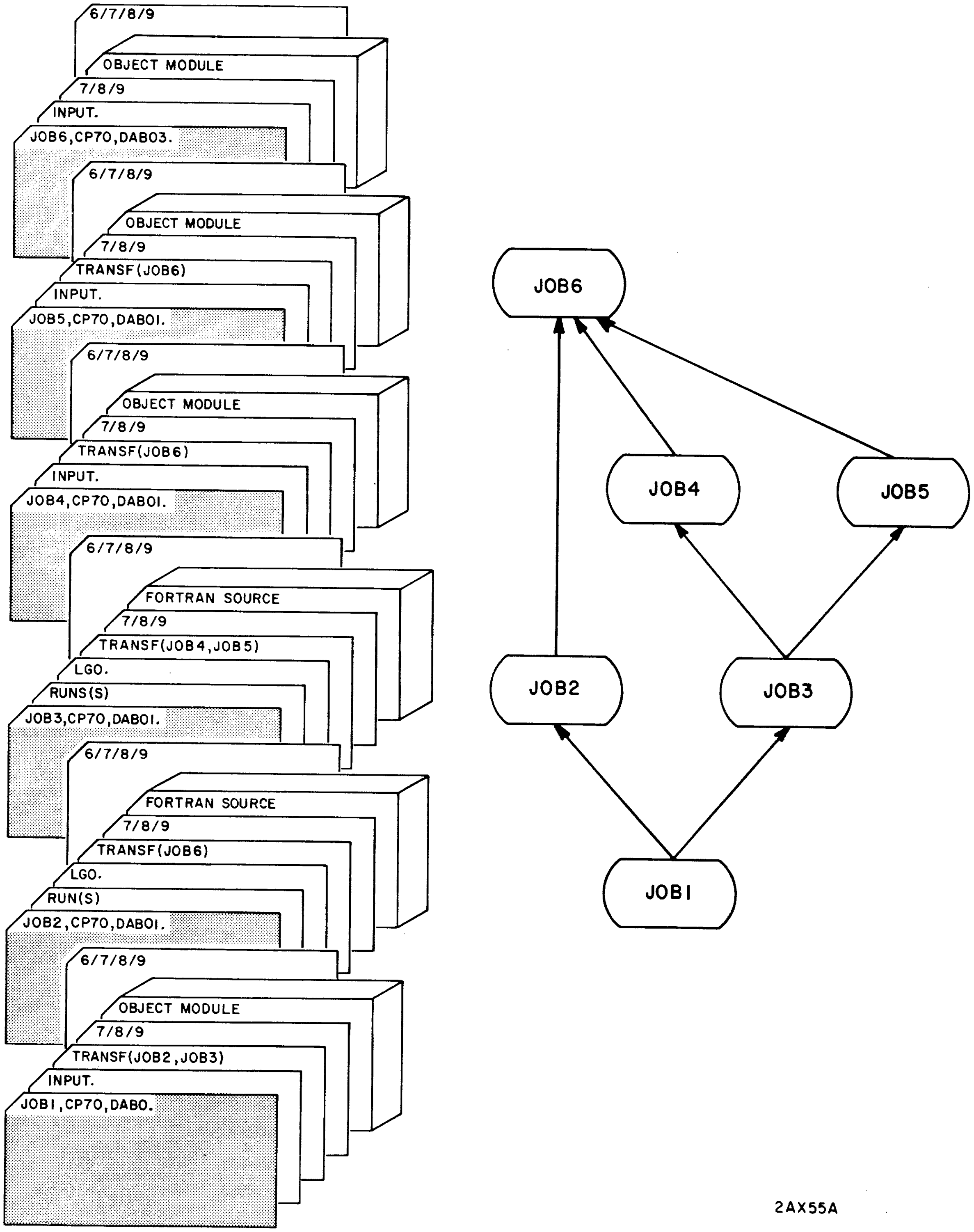
Parameters of the TRANSF statements consist of the names of jobs in the dependency string for which the job meets some need. Only the first five characters of the job name are relevant.

A job will wait indefinitely for its dependencies. Also, if a job posts a dependency for a job not yet in the system, SCOPE maintains a record of the position so that when the job is submitted, it can begin processing.

In Example 4-11, the dependency string consists of six jobs all submitted at the same time and each identified with the identifier AB. JOB1 has no dependencies so it can immediately begin processing. Before its completion, JOB1 posts dependencies for JOB2 and JOB3. Each of these two jobs has one dependency and can now begin processing. Each job in turn signals jobs waiting. JOB3 posts dependencies for JOB4 and JOB5; JOB2, JOB4, and JOB5 each posts a dependency for JOB6. When the dependency count for a job is reached, it can begin processing.

JOB RERUN LIMIT

The operator may terminate a job and resubmit it (that is, rerun it) at any time during processing. Circumstances that might prompt such action are hardware problems at the station, operator errors (for example, mounting the wrong tape), etc. The operating system itself may also rerun a job upon encountering some system error when processing the job (for example, if an SCM or LCM parity error occurred within the user's field length). Following a deadstart recovery, any abnormally terminated jobs are automatically rerun. The user is notified that his job is rerun through a special listing of the control statements in the dayfile. This listing is terminated by the message JOB RERUN.



Example 4-11. Job Dependency String

Normally, a job cannot be rerun if one of the following conditions has occurred. Placing the Rr parameter on the job statement permits the operator to rerun the job despite the occurrence of one of these error conditions.

1. The job has attached a SCOPE 2 permanent file with extend or modify permission.

2. The job has cataloged a permanent file under either SCOPE 2 or 6000 SCOPE 3.x.

3. The job is a member of a dependency string, that is, it has a D parameter on its job identification statement.

If none of the preceding conditions has occurred, there is no limit on the number of times the job can be rerun.

If the user wants the job to be rerun regardless of the occurrence of any of the stated conditions, he enters the letter R followed by one or two octal digits. The value specifies the number of times that the job may be rerun unconditionally.

A job named JOB is to be unconditionally rerun a maximum of five times.

```
JOB,CP70,R5.
```

To specify that the job is not to be rerun under any conditions, enter the parameter R0 on the job identification statement.

A job named ONCE is not to be rerun under any circumstances.

```
ONCE,CP70,R0.
```

REWINDING OF LOAD FILES

Generally, positioning files before loading is not of concern. Rewinding of files is usually assured through an installation rewind option. Refer to the following rules to ensure that the file is rewound or not rewound before loading from it.

Rules for rewinding:

1. Before loading from the file, the loader always rewinds a file name call (for example, the LGO file).

2. The REWIND and NOREWIN options of the LDSET statement are available for specifying that files in the load sequence be rewound or not. This statement should precede the LOAD or SLOAD statements that refer to the affected files. After the load sequence is completed, the installation option again takes effect.

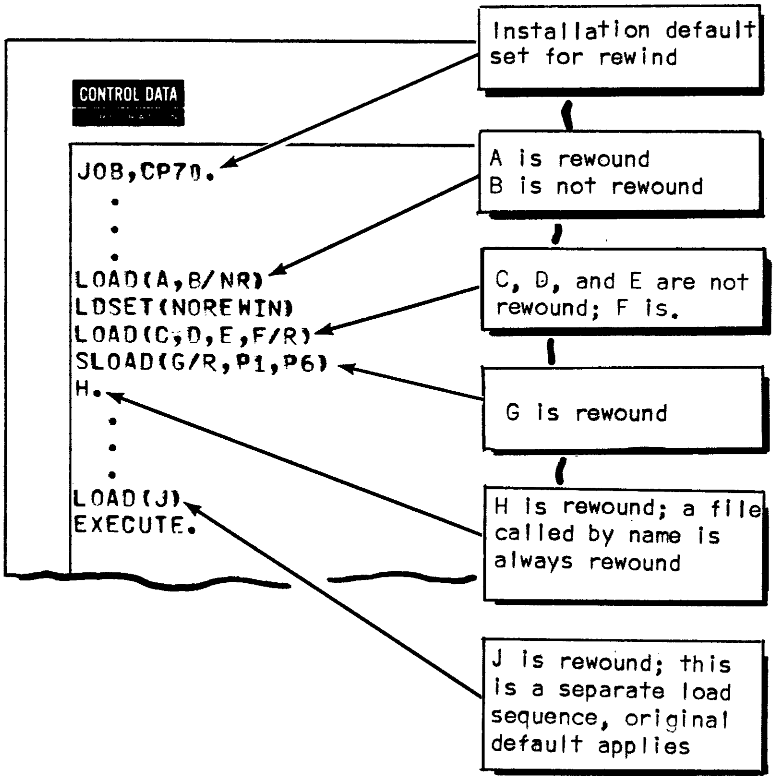
3. Use both LOAD and SLOAD loader statements to specify whether a file is to be rewound or not before loading from it. Do this by entering a file name as lfn/R to indicate rewind or as lfn/NR to indicate no rewind. The parameters on the LOAD and SLOAD statements take precedence over any LDSET statement in the sequence.

4. The only exception to the preceding rules is the INPUT file which is never rewound (set to beginning-of-information) by the loader. The loader ignores any request to rewind INPUT.

5. Use the REWIND statement (Section 10) to explicitly request a file to be rewound before loading from it. This statement is illegal in the load sequence. It can be used for repositioning the INPUT file, however. A rewind of INPUT positions the file to the section following the control statement in the job deck.

6. There is no need to rewind libraries.

Example 4-12 illustrates load sequences that use a combination of the above options.



```
JOB,CP70.
  .
  .
  .
LOAD(A,B/NR)
LDSET(NOREWIN)
LOAD(C,D,E,F/R)
SLOAD(G/R,P1,P6)
H.
  .
  .
  .
LOAD(J)
EXECUTE.
```

Installation default set for rewind

A is rewound
B is not rewound

C, D, and E are not rewound; F is.

G is rewound

H is rewound; a file called by name is always rewound

J is rewound; this is a separate load sequence, original default applies

Example 4-12. Rewind or No Rewind of Load Files

FILE STRUCTURES 5

SCOPE 2 is a file-oriented system. All information, data, and programs known to the system are maintained as logical files. The characteristics of files, for example, record type, are determined by SCOPE 2 defaults, source language programs, or through control statements. Familiarity with the File Information Table, the mechanism through which the system and user communicate information about a file, is helpful to obtain an understanding of how file characteristics are determined.

FILE INFORMATION TABLE

Each logical file used by a job has associated with it a File Information Table (FIT) through which the system and the user communicate information about the file.

SCOPE 2, and the record manager, in particular, expect to find the following information about a file in the FIT.

1. Logical file name
2. File organization (sequential, word addressable, or library)
3. Record type and specifications relevant to record type
4. Blocking type, if any
5. Processing direction (input, output, or input/output)
6. Labeling requirements (blocked files only)
7. End-of-data exit options
8. Error exit options
9. Disposition (processing code), if any
10. Other, optional information

Generally, the programmer provides the file name as a minimum and can rely entirely on the compilers and assemblers to generate a FIT in the SCM field using system default values to fill in the file description. For example, the FORTRAN compilers generate a FIT for each file noted in the PROGRAM statement. The COBOL compilers generate a FIT for each file assigned, using the File Description (FD) entry for the file.

INTRODUCTION TO FILE STATEMENT

Sometimes, the generated FIT does not automatically exactly match the requirements of the file to be generated or the description of an existing file. When this happens, the user has the option of overriding the information in the FIT supplied in the object program by using a FILE control statement. This statement gives the user considerable freedom to control the format of data to be read or written.

Place a FILE statement in the control statement section anywhere before the job step that requires the file specification. The first parameter must be the logical file name (lfn).

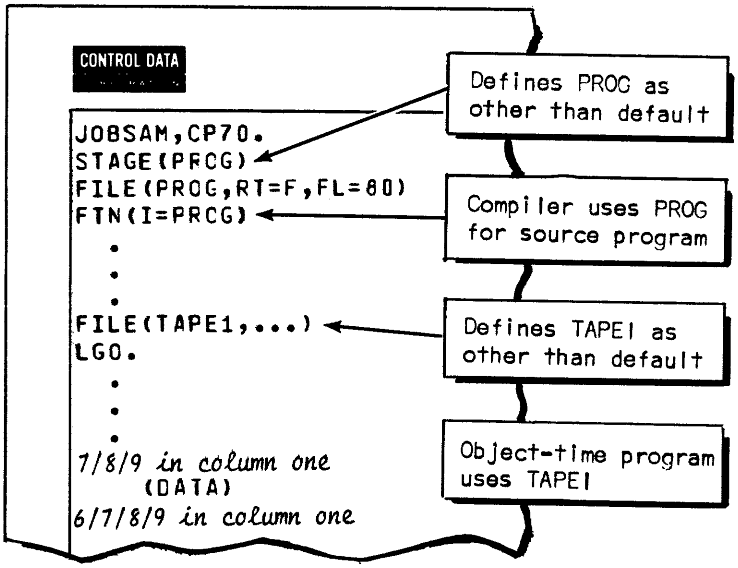
```
FILE(lfn,...)
```

All other parameters can be in any order separated by commas. These parameters are described with related features. For example, the record type parameter (RT) is described under Specifying Record Type.

MULTIPLE FILE STATEMENTS

Information from multiple FILE statements that refers to the same file is merged into the FIT. If a specification is repeated, the most recently encountered specification takes precedence over earlier specifications.

In Example 5-1, the FILE statement to redefine list output file LO precedes the compilation; the FILE statement that redefines TAPE1 precedes the load-and-go statement.



```
JOBSAM,CP70.
STAGE(PROG)
FILE(PROG,RT=F,FL=80)
FTN(I=PROG)
   •
   •
   •
FILE(TAPE1,...)
LGO.
   •
   •
   •
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

CONTROL DATA
Defines PROG as other than default
Compiler uses PROG for source program
Defines TAPE1 as other than default
Object-time program uses TAPE1

Example 5-1. Placement of FILE Statement

CARRYING FILE DEFINITIONS ACROSS JOB STEPS

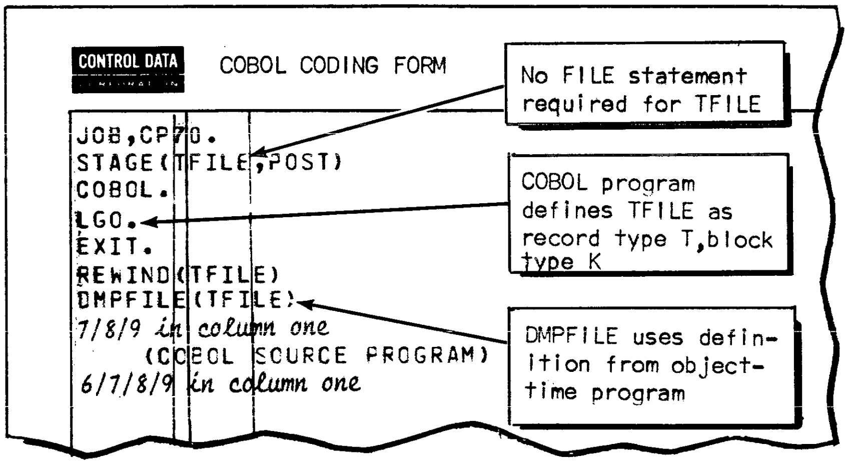
Remember that the File Information Table (FIT) is inside the object program. Let us examine what happens to the file when the object program terminates to see how subsequent job steps are able to use the file. For example, suppose that following object program execution you wish to copy the data file. What file description will the copy routine use - the description used by your object program or the system default for the file? To use the object program description, the answer to both of the following questions must be "yes".

1. Does the object program close the file?

2. Does the routine carry a description from the previous job step to the current routine? This description is maintained in a File Description Table (FDT) within the SCOPE 2 Job Supervisor. Information differs somewhat from that in the FIT.

Neither factor is easily discernible to the user. As a general rule, files are closed between job steps. Most members of the SCOPE 2 product set, that is, compilers, assemblers, object-time routines, and library maintenance routines close files and have SETFIT macros in their file initialization procedures. Copy routines are an exception; they do not close files.

What this means is that the file definition used by the first step that accesses the file applies to the entire job if there are no subsequent FILE statements. Example 5-2 illustrates this point.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70.
STAGE(TFILE,POST)
COBOL.
LGO.
EXIT.
REWIND(TFILE)
DMPFILE(TFILE)
7/8/9 in column one
    (COBOL SOURCE PROGRAM)
6/7/8/9 in column one
```

No FILE statement required for TFILE

COBOL program defines TFILE as record type T,block type K

DMPFILE uses definition from object-time program

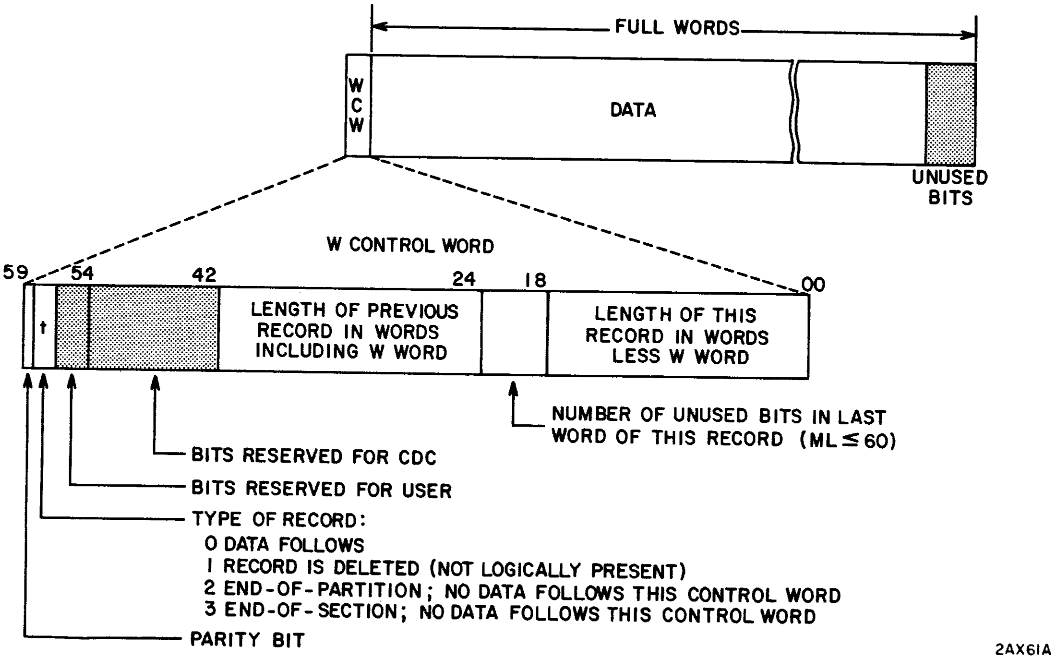
Example 5-2. Carrying File Definition Across Job Steps

SPECIFYING RECORD TYPE

The logical record is the basic unit of data handled by the record manager. Its definition varies according to record type. That is, the end-of-record is defined separately for each record type other than U, for which it is undefined. Record lengths are defined in units of 6-bit characters.

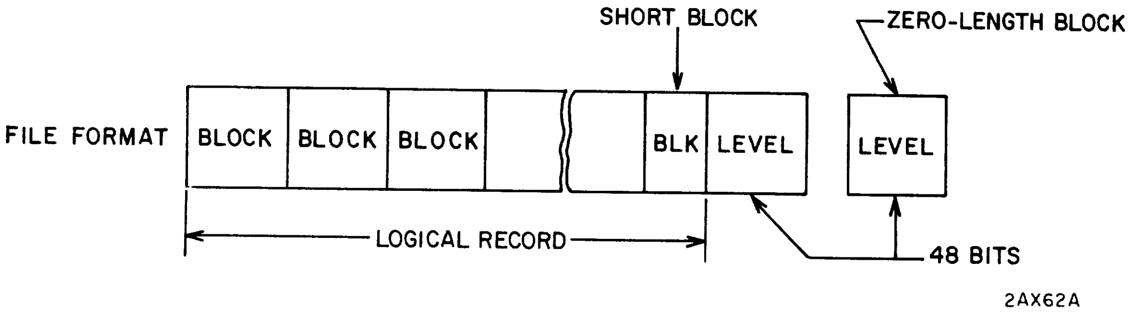
The nine record types and their associated FILE statement parameters are briefly summarized in the following text. Additional detail is provided in Appendix D.

W Control Word: Each record is characterized by a control word header containing sizes of current and previous records. This is the system and FORTRAN (RUN and FTN) default. To specify, place RT=W on the FILE statement.

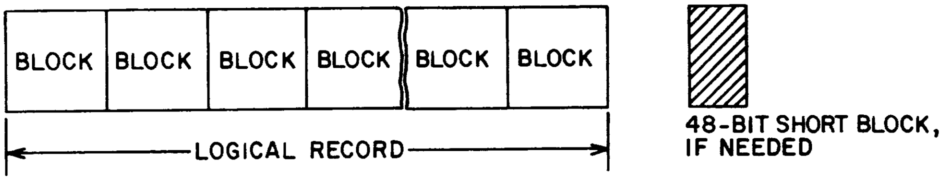


FULL WORDS
W
C
W
DATA
UNUSED BITS
W CONTROL WORD
59
54
42
24
18
00
t
LENGTH OF PREVIOUS RECORD IN WORDS INCLUDING W WORD
LENGTH OF THIS RECORD IN WORDS LESS W WORD
NUMBER OF UNUSED BITS IN LAST WORD OF THIS RECORD (ML≤60)
BITS RESERVED FOR CDC
BITS RESERVED FOR USER
TYPE OF RECORD:
0 DATA FOLLOWS
1 RECORD IS DELETED (NOT LOGICALLY PRESENT)
2 END-OF-PARTITION; NO DATA FOLLOWS THIS CONTROL WORD
3 END-OF-SECTION; NO DATA FOLLOWS THIS CONTROL WORD
PARITY BIT
2AX61A

S SCOPE Logical: Each record consists of blocks of data terminated by a short block to which is appended a 48-bit level number or terminated simply by the level number (called a zero length block). A block is the data between two interrecord gaps on magnetic tape. This record type is also known as 6000 SCOPE Standard and 7600 1.1 I-Mode. To specify, place RT=S on the FILE statement.

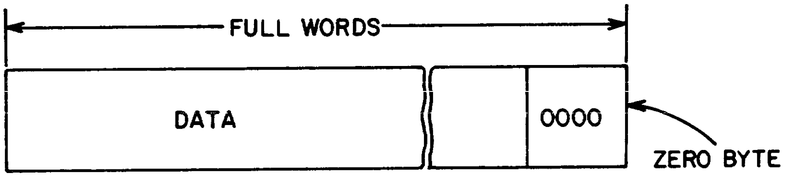


X X-Mode: Each record consists of blocks of data terminated by a short block. If the data ends on a block boundary, the short block consists of a 48-bit short (zero-length) block. This record type is read only. It is specified by RT=X on the FILE statement.



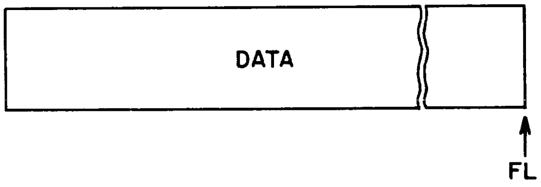
2AX63A

Z Zero Byte: Each record consists of an integral number of 60-bit words in which the last word has the low-order 12 bits set to zero, that is, contains a zero byte. The user must specify RT=Z and FL=n on the FILE statement, where n is a decimal count of characters. It must be large enough to accommodate the largest record on the file. The default for n is 0.



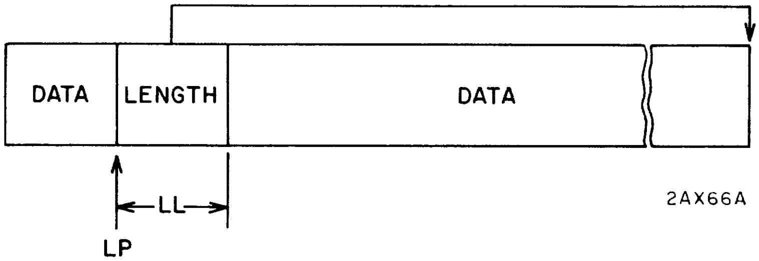
2AX64A

F Fixed Length: Each record consists of a fixed number of characters. This record type is specified by RT=F, FL=n on the FILE statement, where n is the decimal count of characters in each record. The default for n is 0.

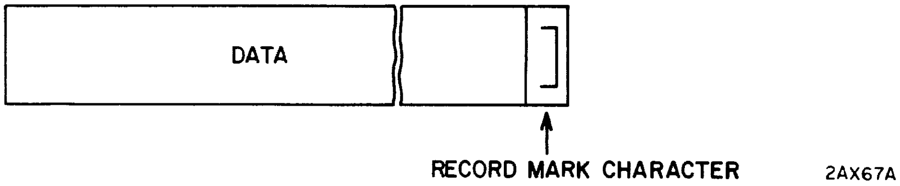


2AX65A

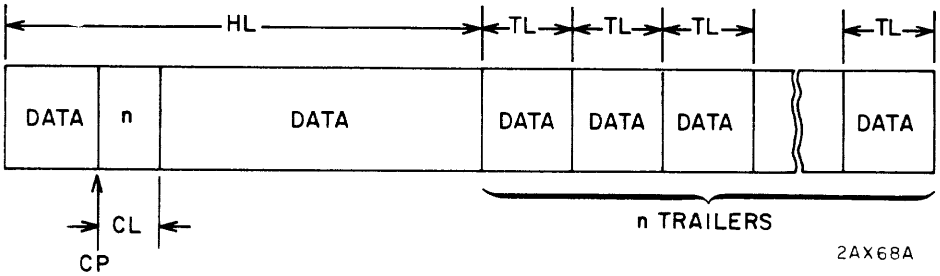
D Decimal Count: Each record consists of a number of characters specified in a decimal count field within the record. The record type is specified as RT=D. The position of the length field is specified in characters as LP=n; the length of the length field is specified as LL=m, where m is 1 to 6. The defaults for n and m are 0.



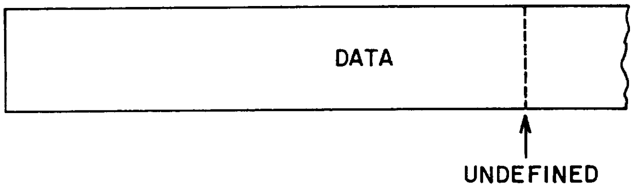
R Record Mark: Each record consists of a series of characters terminated by a character designated as the record mark character, conventionally] . The default for 7600 record manager, however, is a colon. Use RMK=n, where n is the decimal or octal equivalent (octal equivalent suffixed by B), of the character in display code to specify a record mark character. In addition, it is possible to specify the minimum record length before which the record mark is not sought by the record manager. For this, use MNR=m, where m is a decimal character count. Use RT=R to specify R records.



T Trailer Count: Each record contains a fixed length header followed by a variable number of fixed length trailers. The header length is specified as HL=hl on the FILE statement. The trailer length is specified as TL=tl. In addition, the trailer count field which contains a decimal count of the number of trailers in a record is defined through the count position (CP=cp) and count length (CL=cl) parameters, where cp indicates the beginning character position of the count field and cl indicates the length (1 to 6). The default for all of the parameters is 0. To specify T records, use RT=T.



U Undefined: The size of each record is literally undefined. By using K blocking with one record per block, the record manager uses block delimiters as end-of-record delimiters. Use RT=U to specify U record type.



2AX69A

The user can either use the default type specified by the FIT assembled or compiled for the file or can use the RT and associated parameters on a FILE control statement to specify some other record type. This must be done with great care since some programs are not designed to handle all record types. For example, the SCOPE 2 system default record type is W. No other record type can be used for input or output from the loader or for the standard files, INPUT, OUTPUT, PUNCH, and PUNCHB.

The FORTRAN Extended and RUN compilers always generate a FIT with the record type set to W. This is the easiest record type to use. Other record types are subject to the constraints listed in Table 5-1.

The COBOL compiler generates a FIT for each of the system files INPUT, OUTPUT, PUNCH, and PUNCHB whether they are assigned in the program or not. If they are assigned, they have record type W.

For any other file,' COBOL sets the record type according to the file description (FD) entries in the COBOL source program, as shown in Table 5-2.

As a general rule for output, regardless of the file description, a file can be redefined as W record type. If all of the records are the same length, the file can be defined as F or Z record type. If records are all multiples of 10 characters (full words), the file can be redefined as S record type.

TABLE 5-1. FORTRAN RECORD TYPE CONSTRAINTS

Record Type	Constraint When Writing
W	Recording mode must be binary for magnetic tape. Each write creates a W record.
S	Each write creates an S record. Recording mode must be binary.
X	Read only; no writing allowed.
Z	Recording mode must be binary for magnetic tape. Each write creates a Z record.
F	User must ensure that all records are fixed length.
D	User must insert record length in the decimal count field. Decimal count field must be within FORTRAN object-time buffer limits.
R	User must supply record mark character that terminates data.
T	User must insert trailer count in the count field in the header. The count field must be within the FORTRAN object-time buffer limits.
U	Only block type K with one record per block is allowed. Each write creates a block containing one record.

On a read, with the exception of Z records, if the record does not completely fill the buffer, the remainder of the area is unchanged from the last read; it is not blank filled. For Z records, the area is blank filled up to FL.

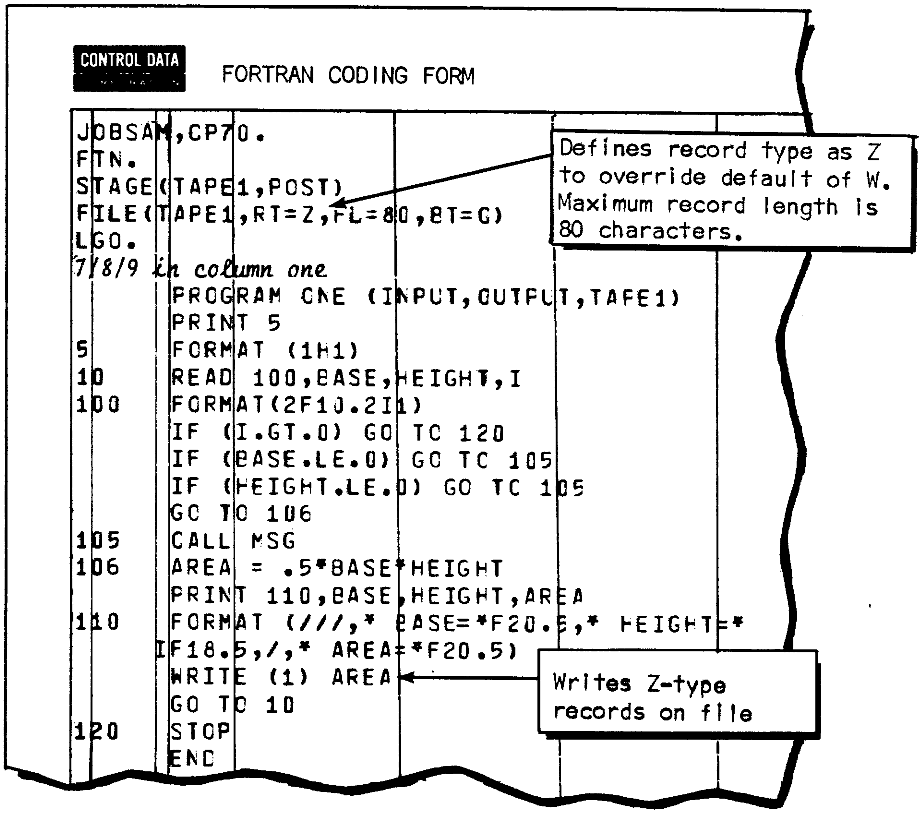
TABLE 5-2. COBOL SPECIFIED RECORD TYPES

FD Entries	01 Entries of Same Length	01 Entries of Different Lengths	01 Entry with OCCURS... DEPENDING ON data name
RECORD CONTAINS integer_1 TO integer_2 CHARACTERS BLOCK CONTAINS 1 RECORD or BLOCK CONTAINS clause omitted	U	U	T
RECORD CONTAINS integer_1 TO integer_2 CHARACTERS BLOCK CONTAINS integer_2 RECORDS or BLOCK CONTAINS integer_2 CHARACTERS	W	W	T
RECORD CONTAINS clause omitted BLOCK CONTAINS 1 RECORD	F	U	T
RECORD CONTAINS clause omitted BLOCK CONTAINS integer_2 RECORDS	F	W	T
RECORD CONTAINS integer CHARACTERS	F	illegal	illegal
RECORD CONTAINS integer TO integer CHARACTERS DEPENDING ON data-name	D	D	illegal
RECORD CONTAINS integer TO integer CHARACTERS DEPENDING ON RECORD MARK	R	R	illegal

CAUTION

When using COBOL, it is possible for a file to contain records of more than one type.

In Example 5-3, the FIT generated for TAPE1 by the RUN compiler defines record type as W. The STAGE statement, as will be described later, causes the file to be blocked. To change the description, the programmer inserts a FILE statement before execution causing record type to be Z with a record length of 80. In this example, block type (BT) is also specified to change the block type to C from the default for Z which would have been K.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,POST)
FILE(TAPE1,RT=Z,FL=80,BT=C)
LGO.
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TC 120
      IF (BASE.LE.0) GC TC 105
      IF (HEIGHT.LE.0) GO TC 105
      GC TO 106
105   CALL MSG
106   AREA = .5*BASE*HEIGHT
      PRINT 110,BASE,HEIGHT,AREA
110   FORMAT (///,* BASE=*F20.5,* HEIGHT=*
     IF18.5,/,* AREA=*F20.5)
      WRITE (1) AREA
      GO TO 10
120   STOP
      END
```

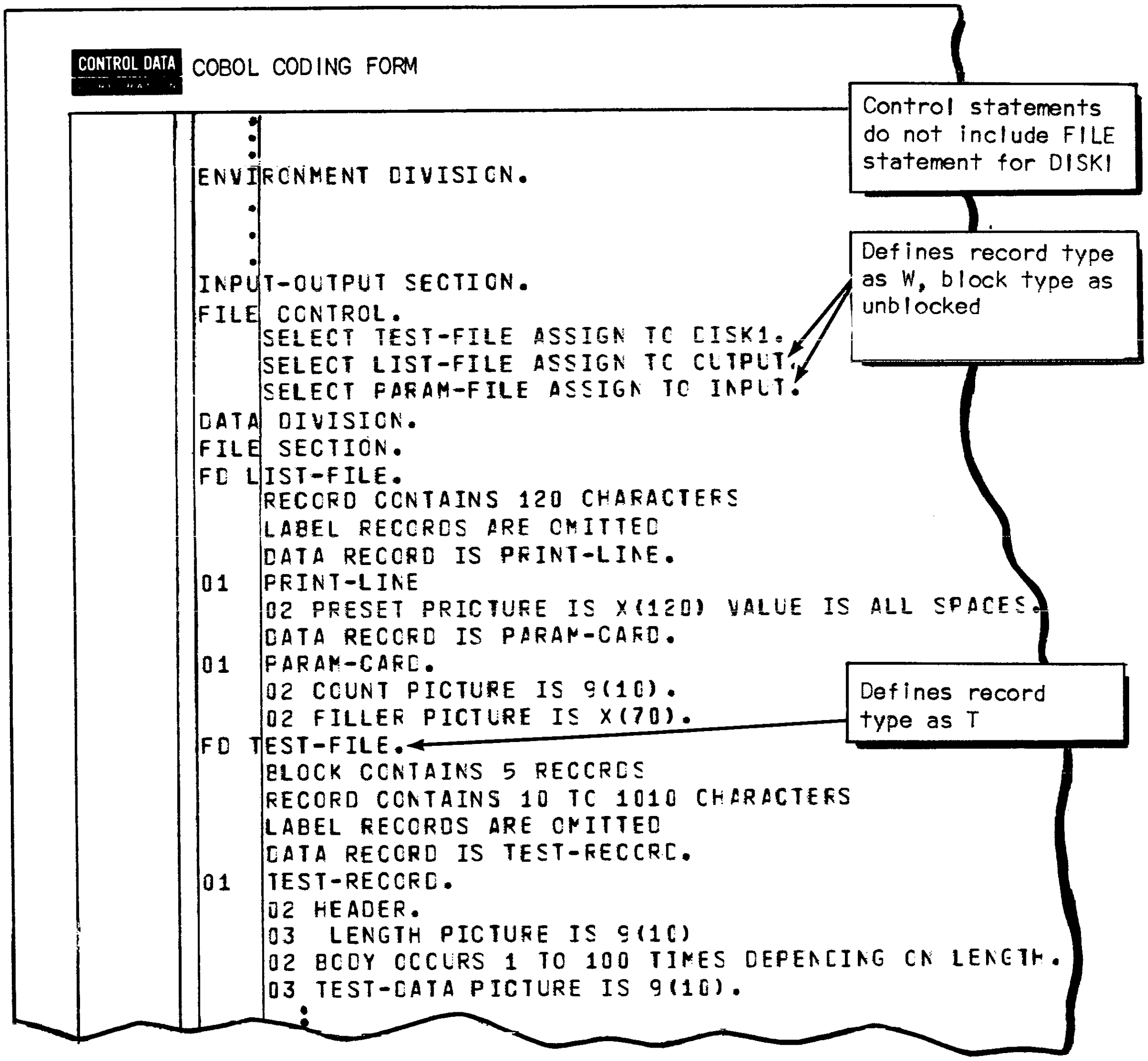
Defines record type as Z to override default of W. Maximum record length is 80 characters.

Writes Z-type records on file

Example 5-3. Overriding Default of W Record Type for FORTRAN Program

Example 5-4 illustrates file description entries for COBOL implementor names LIST-FILE, PARAM-FILE, and TEST-FILE. From the entries, COBOL determines that LIST-FILE and PARAM-FILE are record type F. These descriptions are overridden, however, by the ASSIGN clause which assigns these files to system files OUTPUT and INPUT, making them W unblocked.

The FD entry for TEST-FILE describes trailer (T) records. Thus, the record type for DISK1, to which TEST-FILE is assigned, is set to T.



```
       .
       .
ENVIRONMENT DIVISION.
       .
       .
       .
INPUT-OUTPUT SECTION.
FILE CONTROL.
       SELECT TEST-FILE ASSIGN TO DISK1.
       SELECT LIST-FILE ASSIGN TO OUTPUT.
       SELECT PARAM-FILE ASSIGN TO INPUT.
DATA DIVISION.
FILE SECTION.
FD LIST-FILE.
       RECORD CONTAINS 120 CHARACTERS
       LABEL RECORDS ARE OMITTED
       DATA RECORD IS PRINT-LINE.
01     PRINT-LINE
       02 PRESET PRICTURE IS X(120) VALUE IS ALL SPACES.
       DATA RECORD IS PARAM-CARD.
01     PARAM-CARD.
       02 COUNT PICTURE IS 9(10).
       02 FILLER PICTURE IS X(70).
FD TEST-FILE.
       BLOCK CONTAINS 5 RECORDS
       RECORD CONTAINS 10 TO 1010 CHARACTERS
       LABEL RECORDS ARE OMITTED
       DATA RECORD IS TEST-RECORD.
01     TEST-RECORD.
       02 HEADER.
       03  LENGTH PICTURE IS 9(10)
       02 BODY OCCURS 1 TO 100 TIMES DEPENDING ON LENGTH.
       03 TEST-DATA PICTURE IS 9(10).
       .
```

Control statements do not include FILE statement for DISK1

Defines record type as W, block type as unblocked

Defines record type as T

Example 5-4. COBOL Assignment of Record Types Through File Description

SPECIFYING THE MAXIMUM RECORD LENGTH

The MRL parameter on the FILE statement permits the user to specify the maximum record size for the file. MRL does not apply for F and Z records; for these two record types, FL serves the same purpose. MRL is significant only for input; it is ignored on output.

The setting of MRL depends on whether the program I/O routines manipulate full or partial records. The FORTRAN and COBOL object-time routines always manipulate full records. Do not use a FILE statement to set MRL because the compiler sets the value for you. FORTRAN sets the MRL to 150 for coded files and to 327,680 for binary files.

In COBOL, the size of MRL is determined by the sum of all the fixed length elementary items plus the sum of the maximum number of variable length items in the record. Either the RECORD CONTAINS clause or the OCCURS clause is used in determining the maximum size for variable length records.

The MRL set by the compiler takes precedence over the system default for MRL, which is 0. MRL of 0 specifies unlimited record size.

To specify MRL, use the MRL parameter on the FILE statement where n is the number of characters in decimal.

```
FILE(lfn, MRL=n, . . . )
```

Rules:

Do not set MRL for the INPUT file to less than 88.

For block types K and E, MRL cannot exceed block size (MBL). (See Blocked File Format page 5-14.)

UNBLOCKED FILE FORMAT

> NOTE
>
> The term "unblocked" used in the SCOPE 2 sense, is essentially a gapless format supported only on a mass storage. It is a continuous stream of data. In comparison the industry accepted meaning for unblocked describes the situation where the information between two interrecord gaps on magnetic tape comprises one record. In this case, a block and a record are synonymous. To SCOPE 2, this is a blocked format with one record per block.

The unblocked file can exist on mass storage only. Just one record type, the W record type, permits delimiters of a higher order than records, that is, permits sections and partitions on mass storage. This is because the unblocked file normally has no vehicle for maintaining section and partition delimiters. When W control words are present, they act as such a vehicle (Figure 5-1). Thus, for record types other than W, an unblocked file is simply a collection of records. S and X records cannot be unblocked.

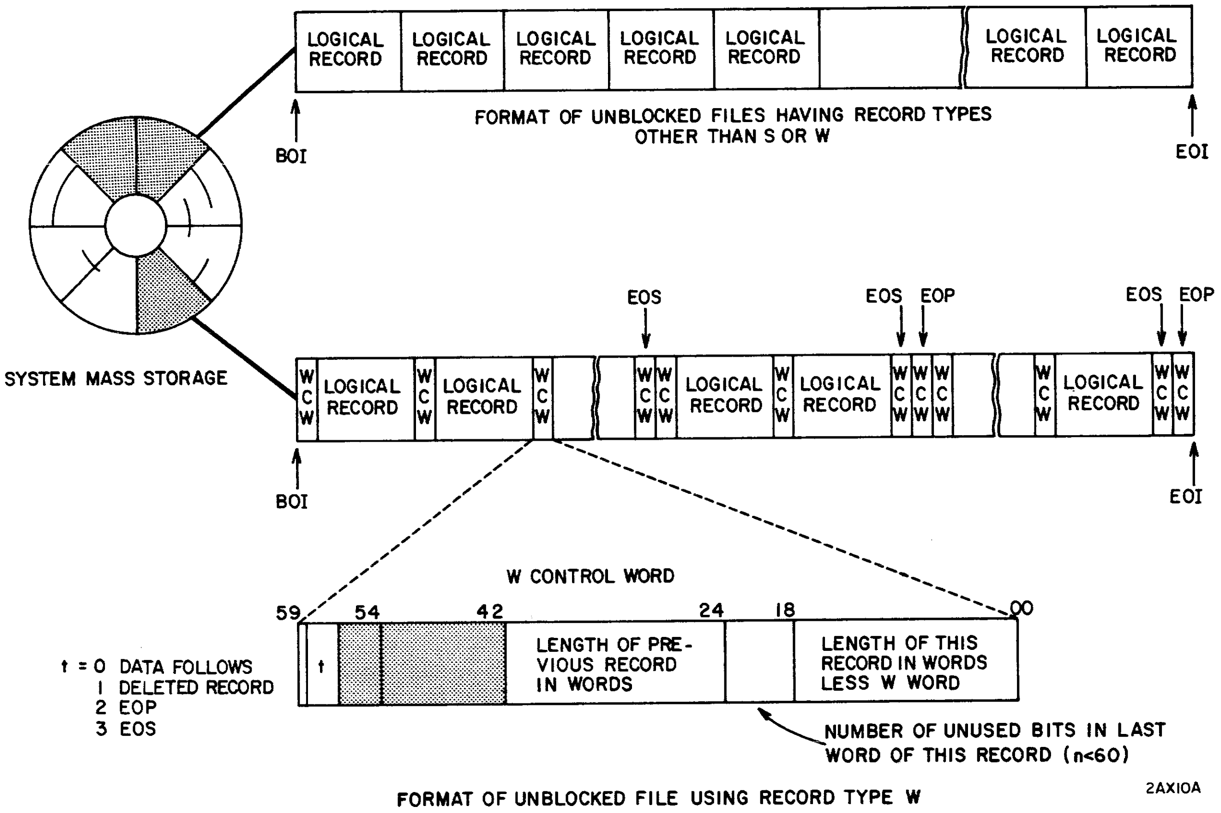


Figure 5-1. Unblocked File Format

Rules for Accessing Unblocked Files

1. Unblocked files can be accessed using any of the file organizations. Remember, however, that 6000 SCOPE 3.4 does not allow sequential (SQ) files to be unblocked.

2. Only unblocked files can be accessed as word addressable (WA) files.

3. Only unblocked files using W record type can be accessed as library (LB) files.

Thus, the LB file organization can be considered as a special case of the word addressable file.

How to Specify Unblocked

1. The system default for blocking type (BT) in the FIT is normally unblocked for mass storage files. The default for COBOL is blocked (Table 5-3).

2. The user can specify unblocked for a mass storage file by providing a FILE statement with a null BT parameter (simply BT).

```
FILE(lfn,BT)
```

Specifying unblocked for a magnetic tape file is illegal.

BLOCKED FILE FORMAT

Blocking of records is the process of grouping a number of logical records before writing them on a magnetic tape file. This grouping is called a block. Grouping two or more records per block improves data transfer rates by reducing the number of interrecord gaps in the file. Blocking usually increases processing efficiency by reducing the number of physical input/output operations required to process the file.

Blocked files can be magnetic tape files (Figure 5-2) or mass storage files (Figure 5-3). When on magnetic tape, a file must be blocked. Even when it is on mass storage, a blocked file is basically an image of a magnetic tape file. Therefore, delimiters possible on magnetic tape file such as interrecord gaps and tapemarks must be simulated on blocked mass storage files. The vehicle used to simulate these delimiters is the recovery control word (RCW) (Figure 5-3).

THE BLOCK

On magnetic tape, a block is the information contained between two interrecord gaps. On mass storage a block is the information between two recovery control words. Except for the fact that the record manager must be informed that a file is blocked (see rules for specifying blocking), blocking has little impact on the user. Blocks are invisible to FORTRAN and COBOL object-time routines because the record manager deblocks the records on input and blocks them on output. The various blocking types (Figure 5-2) are designed to meet ANSI standards and/or to provide compatibility with formats used on other computer systems.

For S, X, and Z records with C blocking, a short block has special meaning. For S records, it signals either end-of-record (level 0, 48-bit appendage) or end-of-partition (level 17_8, 48-bit appendage). For X records it signals end-of-record. For Z records, it signals either end-of-section (level 0, 48-bit appendage) or end-of-partition (level 17_8, 48-bit appendage).

Generally, the default block type is the most efficient for the record type. The block size depends on several factors; in general, it should be around 5000 characters. Larger sizes tend to increase the chance of parity errors, which in turn reduces throughput. For on-line tapes, if the application program processes data fast enough to achieve nonstop I/O, it may be advisable to use shorter blocks to sustain the I/O. Also, by selecting a small block size, the user could keep core use to a minimum. This technique is advisable in a heavily loaded multiprogramming environment.

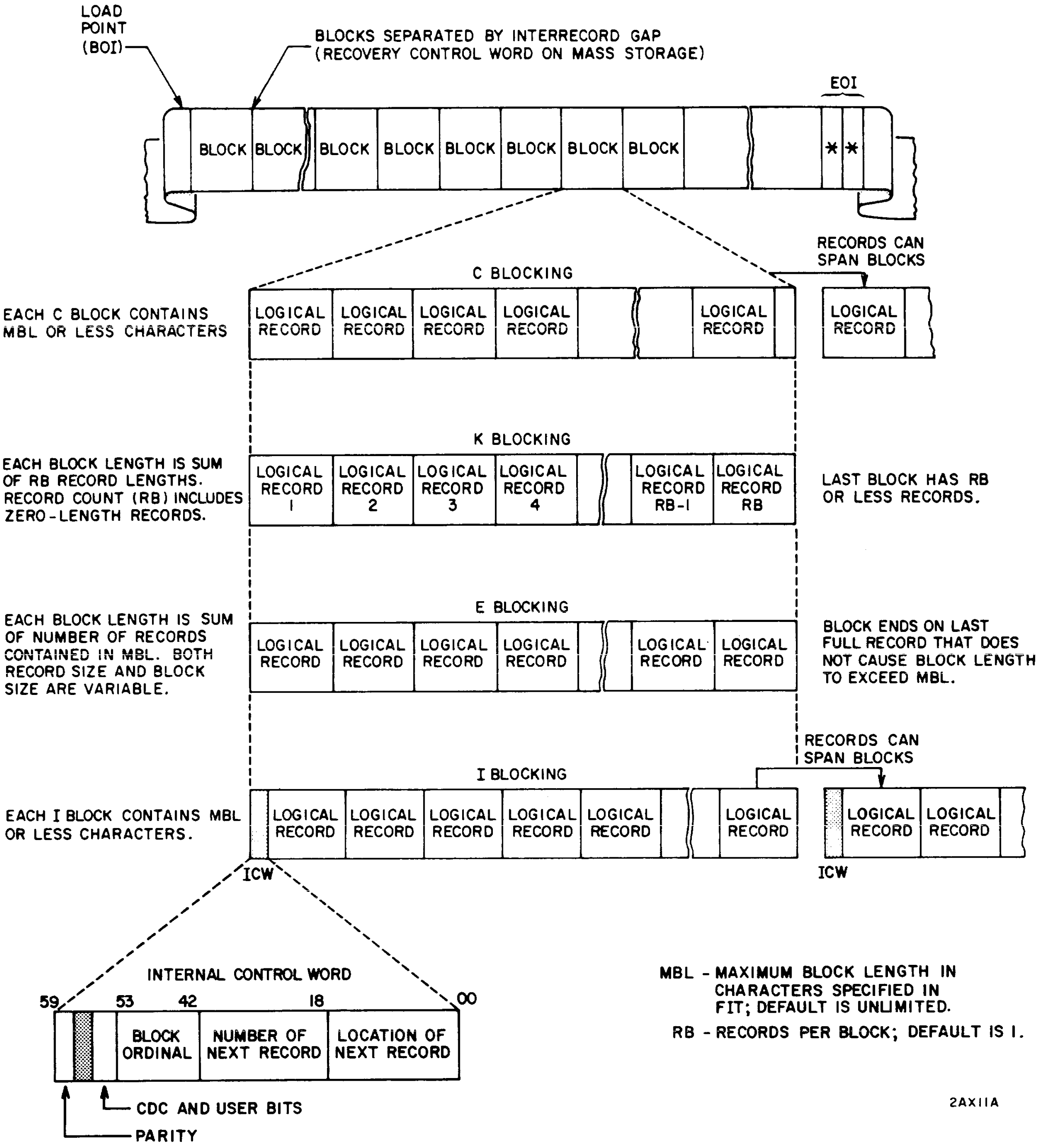

LOAD POINT (BOI)
BLOCKS SEPARATED BY INTERRECORD GAP (RECOVERY CONTROL WORD ON MASS STORAGE)
EOI
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
* *
RECORDS CAN SPAN BLOCKS
C BLOCKING
EACH C BLOCK CONTAINS MBL OR LESS CHARACTERS
LOGICAL RECORD
K BLOCKING
EACH BLOCK LENGTH IS SUM OF RB RECORD LENGTHS. RECORD COUNT (RB) INCLUDES ZERO-LENGTH RECORDS.
LOGICAL RECORD 1
LOGICAL RECORD 2
LOGICAL RECORD 3
LOGICAL RECORD 4
LOGICAL RECORD RB-1
LOGICAL RECORD RB
LAST BLOCK HAS RB OR LESS RECORDS.
E BLOCKING
EACH BLOCK LENGTH IS SUM OF NUMBER OF RECORDS CONTAINED IN MBL. BOTH RECORD SIZE AND BLOCK SIZE ARE VARIABLE.
BLOCK ENDS ON LAST FULL RECORD THAT DOES NOT CAUSE BLOCK LENGTH TO EXCEED MBL.
RECORDS CAN SPAN BLOCKS
I BLOCKING
EACH I BLOCK CONTAINS MBL OR LESS CHARACTERS.
ICW
ICW
INTERNAL CONTROL WORD
59
53
42
18
00
BLOCK ORDINAL
NUMBER OF NEXT RECORD
LOCATION OF NEXT RECORD
CDC AND USER BITS
PARITY
MBL - MAXIMUM BLOCK LENGTH IN CHARACTERS SPECIFIED IN FIT; DEFAULT IS UNLIMITED.
RB - RECORDS PER BLOCK; DEFAULT IS 1.
2AX11A

Figure 5-2. Blocking Types

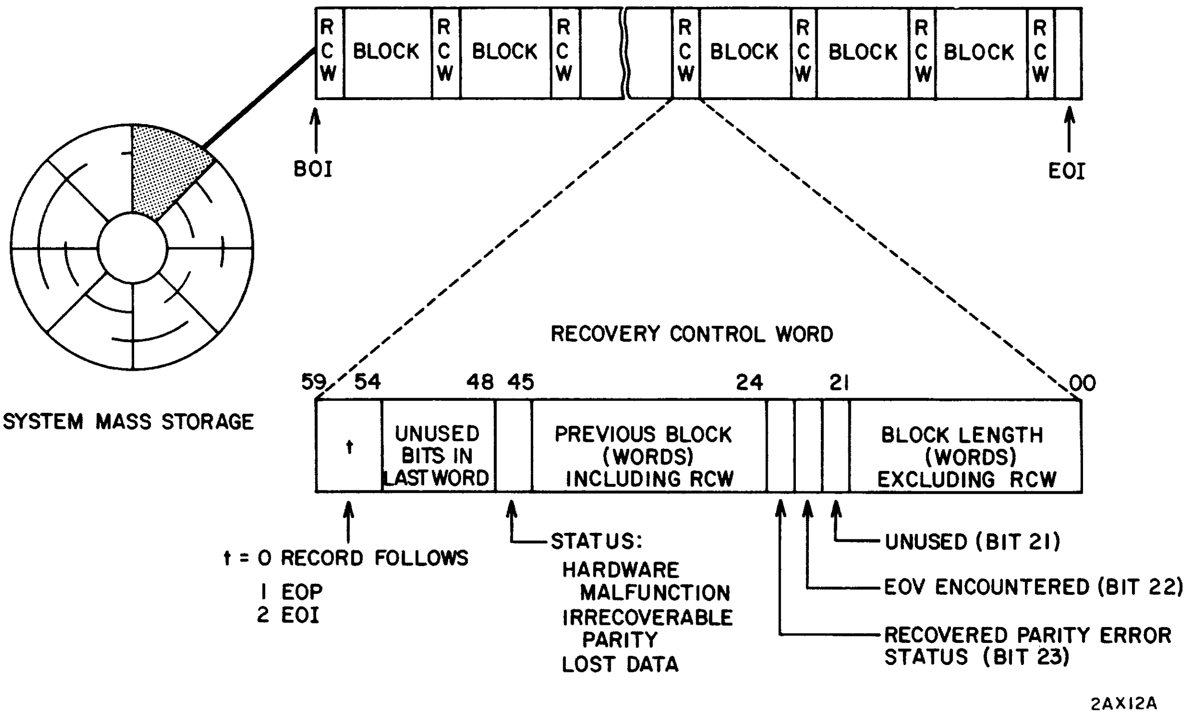


Figure 5-3. Blocked File Format on Mass Storage

E type blocking can be used, but is generally not as efficient as K type blocking. However, for particular applications, E blocking is more suitable than K blocking. For example, E blocking is preferable when a file contains records that vary widely in size.

I blocking and C blocking allow records to span blocks. For K blocking and E blocking records cannot span blocks, thus, maximum record length (MRL) cannot exceed maximum block length (MBL).

Accessing Blocked Files

Blocked files can be accessed as sequential (SQ) files only. Remember that blocked W records cannot be printed, punched, loaded by the loader, or used as input to LIBEDT.

How to Specify Blocking

1. Blocking is automatically specified by the record manager when the user defines a file as being on magnetic tape through use of a REQUEST MT control statement or through use of a STAGE control statement. The default block types are then determined according to record type as follows:

Record Type	Block Type
W	I
S or X	C
other	K

2. Blocking can be specified in the COBOL source language through the BLOCK CONTAINS clause, as shown in Table 5-3. The BLOCK CONTAINS clause takes precedence over system defaults described in Rule 1.

TABLE 5-3. COBOL SPECIFICATION OF BLOCKING

BLOCK CONTAINS Clause	Block Type
BLOCK CONTAINS integer-2 RECORDS BLOCK CONTAINS integer-1 TO integer-2 RECORDS Clause is omitted	K Record Count
BLOCK CONTAINS integer-1 TO integer-2 CHARACTERS	E Exact Records
BLOCK CONTAINS integer-2 CHARACTERS	C Character Count

3. To override the default or COBOL defined block type, or to specify blocking for a mass storage file, use the following parameters on the FILE control statement.

Block Type	File Statement Parameters	Notes
Internal	BT=I, MBL=x	BT=I is allowed for W records only. If the block size (MBL) is unspecified, the default for x is 5120.
Record count	BT=K, RB=x	BT=K is not allowed for S or X records. If the number of records per block (RB) is unspecified, the default for x is 1. K blocking is conventional for U records. RB must be 1 for U records. Maximum block length is computed from maximum record length (MRL) multiplied by records per block (RB). If MRL has not been previously specified it is set to 5120 by default.
Character count	BT=C, MBL=x	If the number of characters per block is unspecified, the default for x is 5120. BT=C is the only block type allowed for S and X records. It is conventional for Z records.
Exact records	BT=E, MBL=x	If the maximum number of characters per block is unspecified, the default for x is 5120. BT=E is not allowed for S or X records.

Rules for Specifying MBL

1. For I blocking, MBL must be 5120 to be compatible with SCOPE 3.4.

2. For C blocking, MBL must be 5120 to be compatible with SCOPE 3.4 S/L devices.

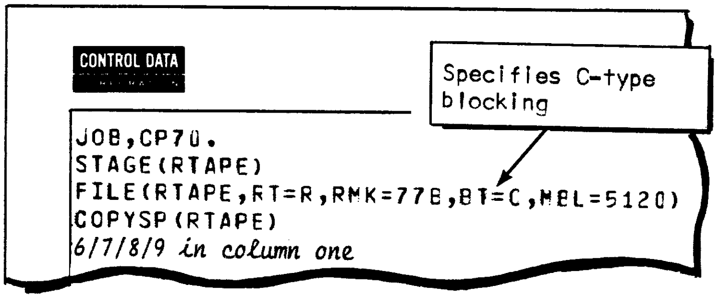
3. For S, X, Z, and W records, MBL must be a multiple of 10 characters (full words).

4. MBL (for BT=K, MRL x RB) is limited by location of the magnetic tape unit as shown in Table 5-4. The 6000/CYBER 70 station restrictions apply also to SCOPE 3.4 blocked permanent files.

TABLE 5-4. MAXIMUM BLOCK SIZES ALLOWED FOR STAGED AND ON-LINE TAPES

Location of Tape Unit	Block Size for 7-Track Tape		Block Size for 9-Track Tape		
	6-Bit Char.	Words	Binary	Coded	Words
On-Line tape	25,590	2559	25,590	19,162	2559
7611-11, 6000, or CYBER 70 Station	25,590	2559	5120	3,840	512
7611-2 Magnetic Tape Station	5120	512	5120	3,840	512
7611-1 I/O Station	5120	512	--	--	--

Example 5-5 illustrates a job that lists a tape created on another computer system. The tape contains records terminated by a 77_8 (; in display code). Records are blocked 5120 characters per block and can span blocks. Blocks are even multiples of 6-bit characters. In this example, C blocking must be specified on the FILE statement to override the system default of K blocking for R records.



CONTROL DATA

Specifies C-type blocking

```
JOB,CP70.
STAGE(RTAPE)
FILE(RTAPE,RT=R,RMK=77B,BT=C,MBL=5120)
COPYSP(RTAPE)
6/7/8/9 in column one
```

Example 5-5. Using a FILE Statement to Specify Blocking

PARTITIONS

End-of-partition is synonymous with the term end-of-file as commonly used for FORTRAN and COBOL languages and previous operating systems. See Figure 5-4, which illustrates end-of-partition on magnetic tape. Note that the representation of end-of-partition is different for some of the record types. A single tapemark is equivalent to an end-of-partition (EOP) for the following:

1. Blocked files with record types F, D, R, T, U, X, and W
2. Z records if they are K or E blocked

For record types S and Z with C blocking, a short block consisting of only 48 bits and having a level number of 17_8 signals an end-of-partition. This is known as a zero-length block because it contains no data. The contents of this block are not passed to the user buffer. Instead, EOP status is returned. A single tapemark is unlikely to occur on S and Z records (C blocked).

For W records, an end-of-partition is signaled through a tapemark or an EOP W control word, depending on how the EOP is generated. The control word is conventional.

Figure 5-4 illustrates conventional usage. That is, a single tapemark could be used to indicate end-of-partition on S record tapes or W record tapes by using the WTMK macro in the COMPASS language instead of the ENDFILE macro, but doing so is not customary.

The following language functions produce an end-of-partition.

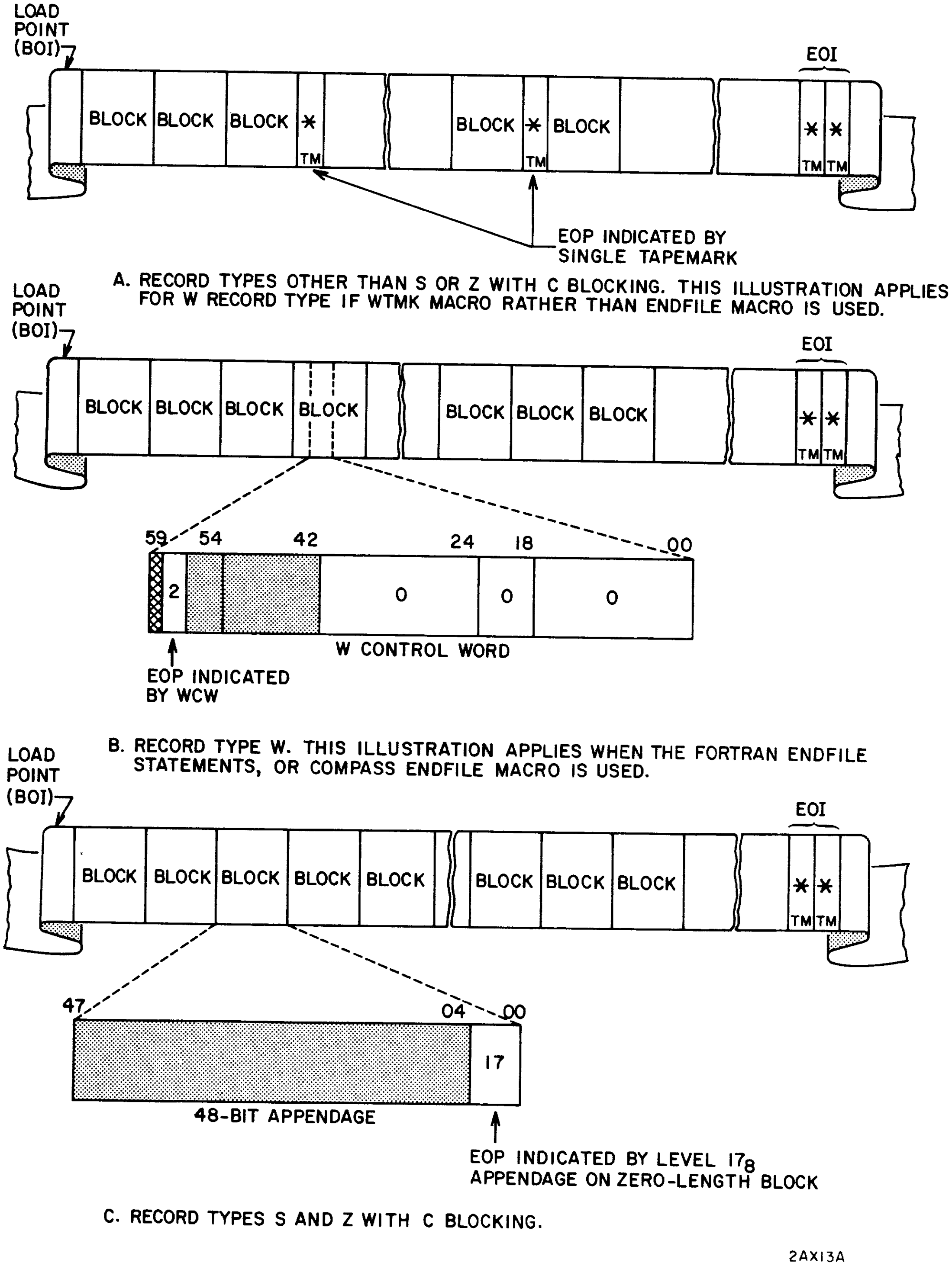
Language	Function
FORTRAN Extended	ENDFILE statement
RUN	ENDFILE statement
COMPASS	WTMK macro and ENDFILE macro

For all record types, the WTMK macro generates a tapemark when it is used on a blocked file. The WTMK macro is ignored if it is used on an unblocked file.

The ENDFILE macro (which is used by copy routines and the compiler object time routines, that is, the ENDFILE statement) does not always generate a tapemark. In particular, it generates an EOP control word for record type W, whether blocked or unblocked. This W control word may occur inside of a block. It also generates the short block when used on S records and Z records with C blocking. For all other record types, it generates a tapemark.

On the INPUT file, the end-of-partition is represented by a 7/8/9 card with a level of 17_8.

An end-of-partition is equivalent to the 7600 SCOPE 1 end-of-file card (6/7/9 multipunch in column 1) or a SCOPE 1 type 2 boundary control word.



A. RECORD TYPES OTHER THAN S OR Z WITH C BLOCKING. THIS ILLUSTRATION APPLIES FOR W RECORD TYPE IF WTMK MACRO RATHER THAN ENDFILE MACRO IS USED.

B. RECORD TYPE W. THIS ILLUSTRATION APPLIES WHEN THE FORTRAN ENDFILE STATEMENTS, OR COMPASS ENDFILE MACRO IS USED.

C. RECORD TYPES S AND Z WITH C BLOCKING.

Figure 5-4. End-Of-Partition on Blocked Magnetic Tape

SECTIONS

In file hierarchy (Figure 5-5), the section lies between the record and the partition. Thus, for the record types that support sections, records can be grouped into sections and sections into partitions. However, only the following file types can be divided into sections.

W records whether blocked or unblocked

Z records with C blocking

On W records, an end-of-section is indicated in a W control word. On Z records with C blocking, an end-of-section is indicated by a short or zero-length block with a level 0, 48-bit appendage.

There is no vehicle for indicating end-of-section using any other record type. On S records, the level 0 appendage indicates end-of-record.

Figure 5-6 illustrates the relationship of S and Z records with C blocking. Note that the same file containing zero-byte delimiters can be defined as either S or Z records. The Z definition describes records, sections, and partitions; the S definition describes records and partitions. The zero bytes are not significant when the file is defined as S records.

With respect to S records, an end-of-section is referred to as an end-of-record by earlier 6000 Series operating systems. Note, however, that although other operating systems may allow several levels of records (levels 0 through 16_8), SCOPE 2 allows only one level (level 0). Levels 0 through 16_8 as they are used by 6000 SCOPE 3.x are converted to level 0 by SCOPE 2.

The FORTRAN and COBOL compiler languages have no counterpart to sections. There is no way of generating a section delimiter when using these languages. The situation could arise, however, where the user desires to have the FORTRAN or COBOL object-time program read a file that contains section delimiters. In particular, the user may want to read from the INPUT file, which conventionally uses end-of-section cards as separators in the deck.

For this reason, the 7600 FORTRAN Extended and RUN object-time routines consider an EOS on INPUT as an EOP. If the routines encounter an EOS on any file other than INPUT, they do not upgrade the EOS to EOP; instead, it is ignored. This implementation is consistent with object-time routines for 6000 SCOPE 3.x. 7600 SCOPE 1.1, however, always ignores the EOS for BCD files and considers it an EOP for binary files. Example 5-6 shows a FORTRAN program that illustrates how FORTRAN object-time routines handle an EOS encountered on a file other than input (in this case, TAPE1). To change the job so that it reads from the INPUT file, remove the COPY statement and change the PROGRAM statement to:

```
PROGRAM      EOS (INPUT,OUTPUT,TAPE1=INPUT)
```

When the EOS is recognized as an EOP, each section begins a new page of printout.

The COBOL object-time routines make no distinction between the INPUT file and other files. If an EOS is encountered, it is always treated as an EOP. This implementation is consistent with 6000 SCOPE 3.4, but differs from 6000 SCOPE 3.3 and 7600 SCOPE 1 implementation, which always ignores the level 0 EOS.

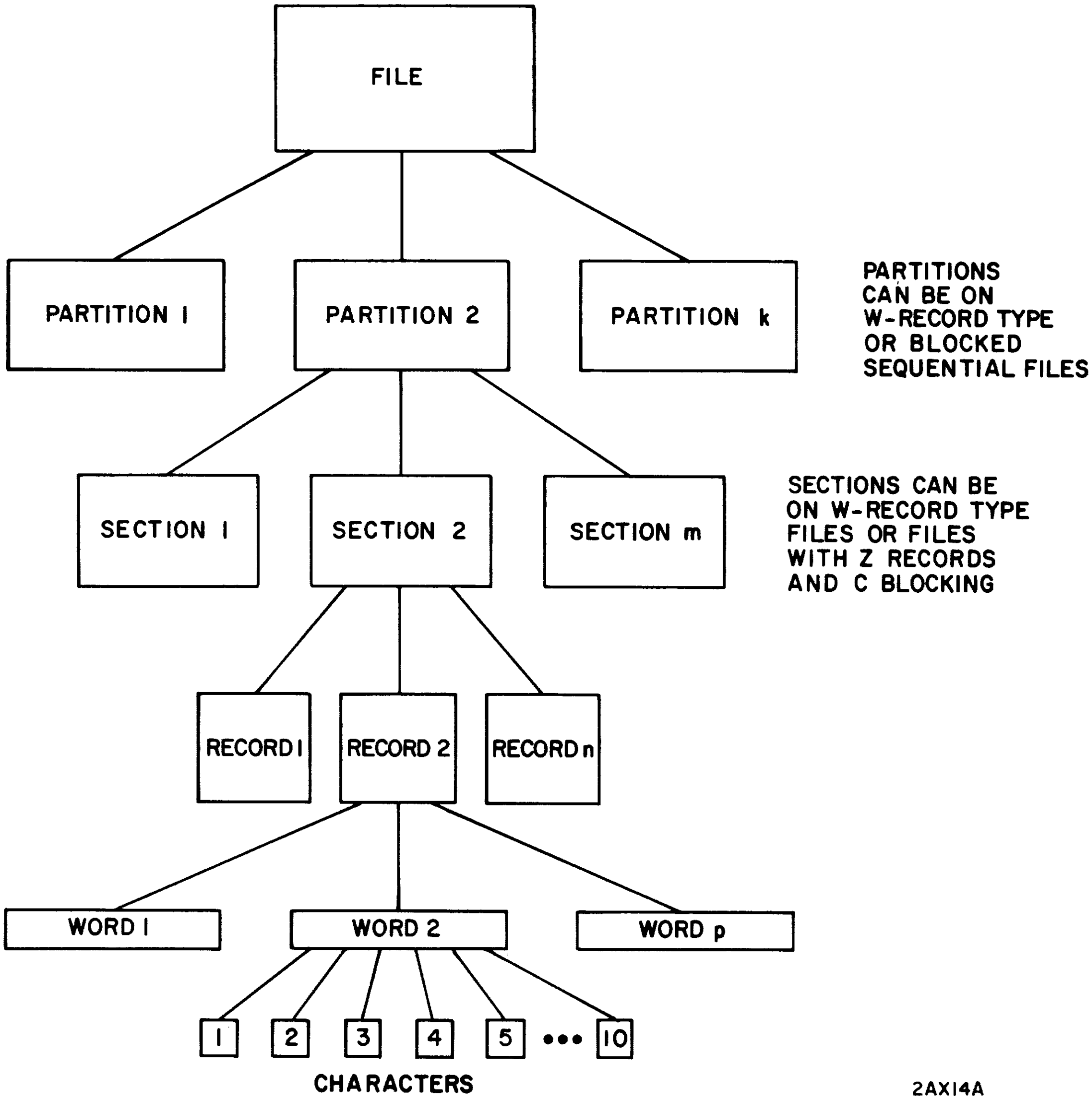

FILE
PARTITION 1
PARTITION 2
PARTITION k
PARTITIONS CAN BE ON W-RECORD TYPE OR BLOCKED SEQUENTIAL FILES
SECTION 1
SECTION 2
SECTION m
SECTIONS CAN BE ON W-RECORD TYPE FILES OR FILES WITH Z RECORDS AND C BLOCKING
RECORD 1
RECORD 2
RECORD n
WORD 1
WORD 2
WORD p
1
2
3
4
5
•••
10
CHARACTERS
2AX14A

Figure 5-5. File Hierarchy

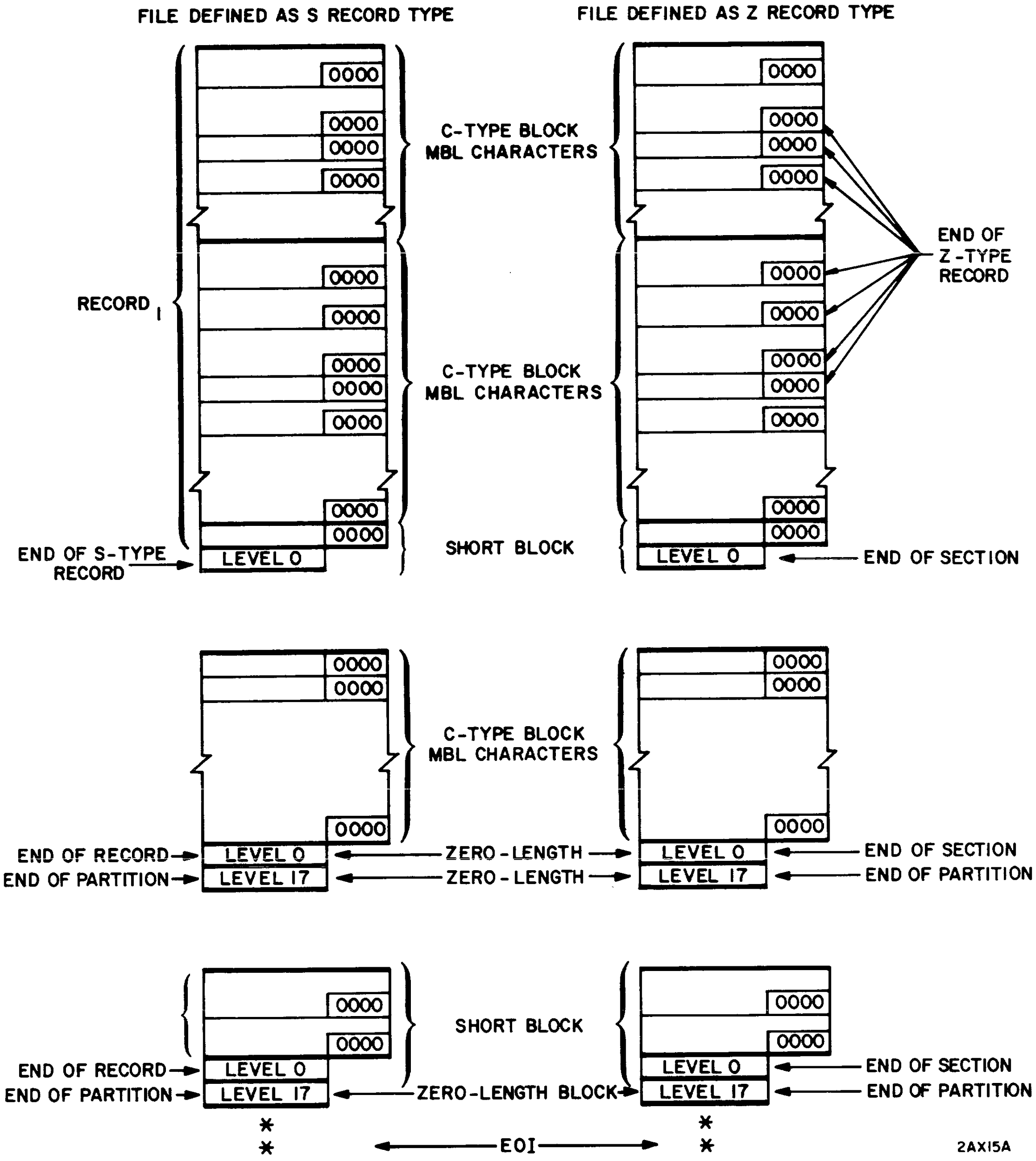
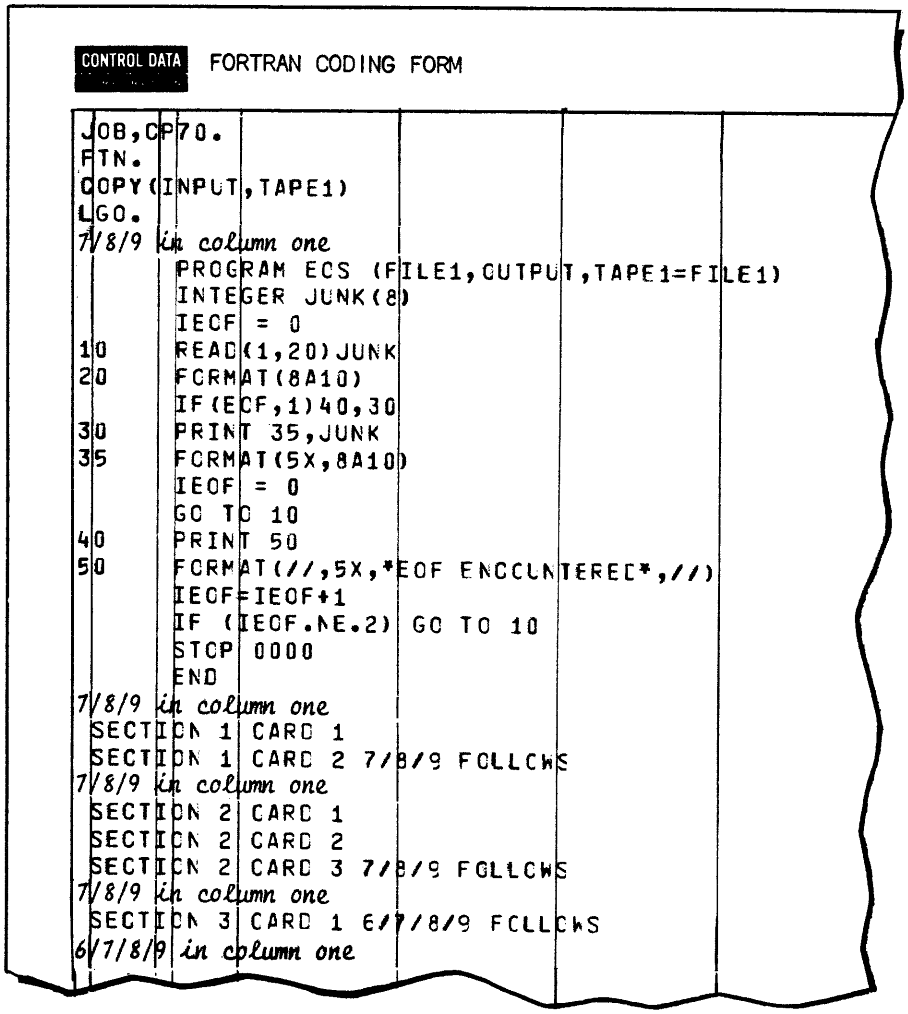


Figure 5-6. Relationship of S and Z Records (C Blocking)



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN.
COPY(INPUT,TAPE1)
LGO.
7/8/9 in column one
      PROGRAM EOS (FILE1,OUTPUT,TAPE1=FILE1)
      INTEGER JUNK(8)
      IEOF = 0
10    READ(1,20)JUNK
20    FORMAT(8A10)
      IF(EOF,1)40,30
30    PRINT 35,JUNK
35    FORMAT(5X,8A10)
      IEOF = 0
      GO TO 10
40    PRINT 50
50    FORMAT(//,5X,*EOF ENCOUNTERED*,//)
      IEOF=IEOF+1
      IF (IEOF.NE.2) GO TO 10
      STOP 0000
      END
7/8/9 in column one
 SECTION 1 CARD 1
 SECTION 1 CARD 2 7/8/9 FOLLOWS
7/8/9 in column one
 SECTION 2 CARD 1
 SECTION 2 CARD 2
 SECTION 2 CARD 3 7/8/9 FOLLOWS
7/8/9 in column one
 SECTION 3 CARD 1 6/7/8/9 FOLLOWS
6/7/8/9 in column one
```

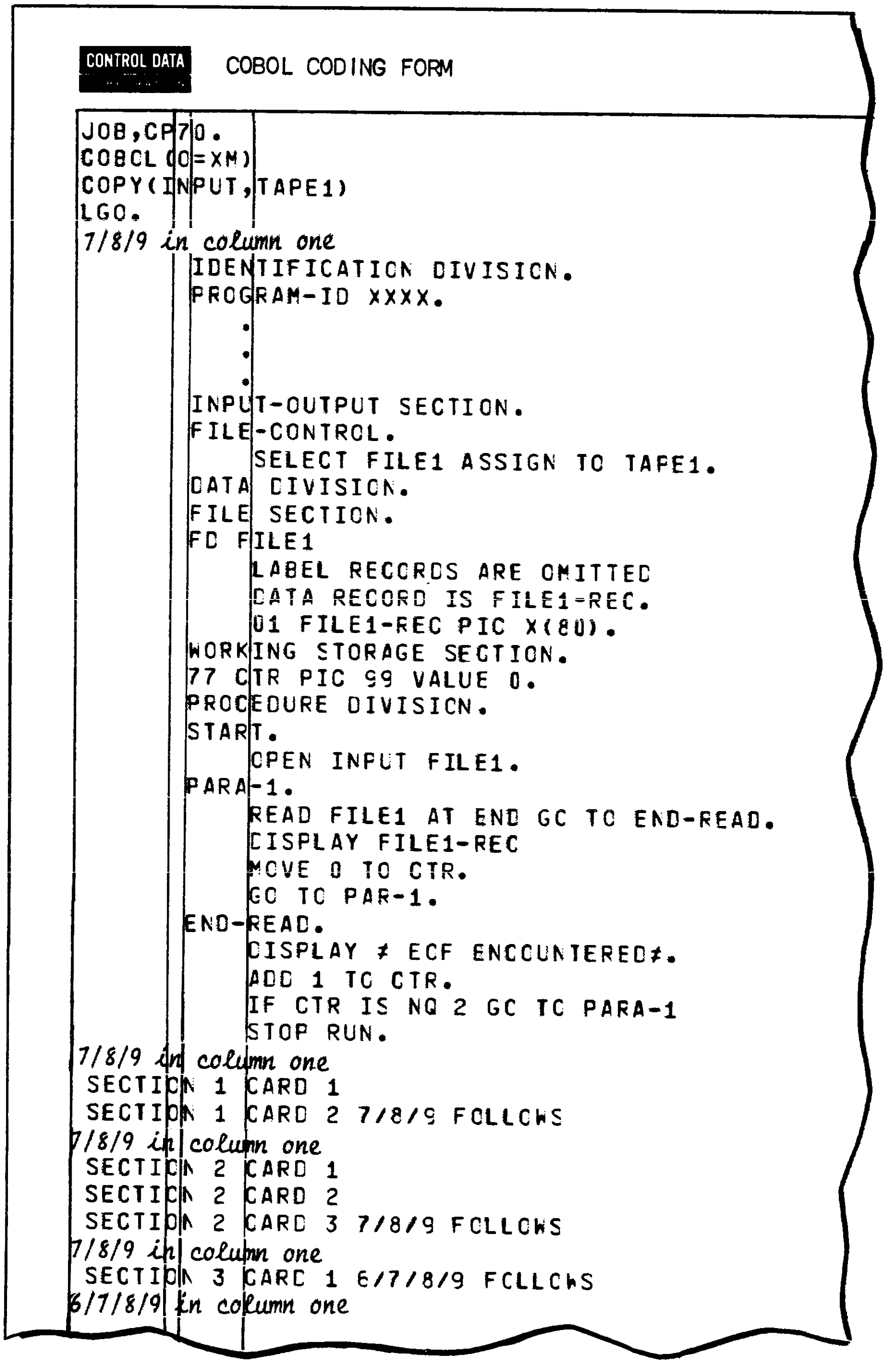
Example 5-6. FORTRAN Treatment of EOS on Input File

Example 5-7 illustrates how COBOL object-time routines handle an EOS encountered on a file other than input (in this case, TAPE1). To change the job so that it reads from the INPUT file, remove the COPY statement and change the file assignment from TAPE1 to INPUT. When the EOS is recognized as an EOP, each section begins a new page of printout.

Section delimiters can be generated through the COMPASS language WEOR macro. The macro is a no-op if it is used on a file type that does not support sections. There is no corresponding function in either the FORTRAN or COBOL languages.

NOTE

> A WEOR on an S-type file should follow a partial write only. If it follows a full write, it causes a superfluous zero-length record.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70.
COBCL(O=XM)
COPY(INPUT,TAPE1)
LGO.
7/8/9 in column one
       IDENTIFICATICN DIVISICN.
       PROGRAM-ID XXXX.
           •
           •
           •
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE1 ASSIGN TC TAPE1.
       DATA CIVISICN.
       FILE SECTICN.
       FD FILE1
           LABEL RECCRDS ARE OMITTEC
           CATA RECORD IS FILE1-REC.
           01 FILE1-REC PIC X(80).
       WORKING STORAGE SECTION.
       77 CTR PIC 99 VALUE 0.
       PROCEDURE DIVISICN.
       START.
           CPEN INPUT FILE1.
       PARA-1.
           READ FILE1 AT END GC TC END-READ.
           CISPLAY FILE1-REC
           MCVE 0 TO CTR.
           GC TC PAR-1.
       END-READ.
           CISPLAY ≠ ECF ENCCUNTERED≠.
           ADD 1 TC CTR.
           IF CTR IS NQ 2 GC TC PARA-1
           STOP RUN.
7/8/9 in column one
 SECTICN 1 CARD 1
 SECTION 1 CARD 2 7/8/9 FCLLCWS
7/8/9 in column one
 SECTICN 2 CARD 1
 SECTICN 2 CARD 2
 SECTION 2 CARC 3 7/8/9 FCLLCWS
7/8/9 in column one
 SECTION 3 CARC 1 6/7/8/9 FCLLCWS
6/7/8/9 in column one
```

Example 5-7. COBOL Treatment of EOS on Input File

ACCESS METHODS

Another factor that the user must consider when using files is the file organization, sometimes referred to as access method. The three file organizations are sequential (SQ), word addressable (WA), and library (LB). Of these, the sequential organization is by far the most commonly used file organization. Library organization is primarily used by the operating system, itself, and has little application for the FORTRAN or COBOL programmer.

The usual default for FORTRAN is sequential. However, a FORTRAN programmer can employ word addressable organization through use of the CALL READMS and CALL WRITMS statements. These are described in the FORTRAN RUN and FORTRAN Extended Reference Manuals. Any file accessed using READMS and WRITMS is automatically defined as word addressable. The user need not supply a FILE statement for the file. The index buffer is the last record in the file.

The COBOL programmer can specify the type or organization used through the ORGANIZATION clause, as follows.

ORGANIZATION IS SEQUENTIAL causes the file to be sequential (SQ).

ORGANIZATION IS STANDARD causes the file to be a word addressable (WA) file compatible with that created with the FORTRAN mass storage routines.

ORGANIZATION IS DIRECT also causes the file to be word addressable (WA) but the index is a different format.

The ORGANIZATION clause can be omitted. When it is omitted, the file organization depends on the factors specified in Table 5-5.

The selection of organization is made in the order shown.

TABLE 5-5. COBOL DETERMINED FILE ORGANIZATION

Order	Condition	Organization
1.	A suffix to implementor or name in ASSIGN clause	Direct (WA)
2.	ACTUAL/SYMBOLIC KEY clause specified	Standard (WA)
3.	ACTUAL KEY and/or FILE-LIMITS specified	Direct (WA)
4.	None of the above conditions exists	Sequential (SQ)

Do not change the file organization through the FO parameter on the FILE statement to conflict with the organization defined for the object-time program.

Only sequential files can be blocked. To save a file having some other file organization on magnetic tape, copy it to a tape. Then, when the tape is loaded, copy it to an unblocked file using the original file organization (see Section 10) before attempting to use the data.

FILE PROCESSING DIRECTION

The processing direction allowed on a file, that is, whether the file is an input file permitting reads only, an output file permitting writes only, or an I/O file permitting either reads or writes, is determined by open requests issued by object time routines. The direction is maintained in a field in the FIT for the file.

COBOL and FORTRAN object times routines and the SCOPE 2 copy routines usually open a file as input/output. A staged-in file is opened for input; a staged out file is opened for output.

A parameter of the FILE statement permits a user to specify processing direction, but this parameter is very seldom needed for mass storage files and may conflict with the processing directions specified on open requests. If a conflict occurs, the open request takes precedence over the direction specified on the FILE control statement.

```
FILE(lfn,...,PD=direction)
```

Direction can be I for input, O for output, or I-O for input/output.

FILE ERROR OPTIONS

When the record manager encounters an error condition, it either terminates the job or attempts to continue program execution despite the error. The action taken depends on the setting of the error option (EO) field in the FIT and on the type of error. As will be shown, processing of read parity errors represents a special case.

TERMINATION ON ANY ERROR

By system default, the error option field is set for termination to occur upon encountering any record manager error. The FORTRAN compiler, however, generates a FIT with this field set for accept and display. The COBOL compiler sets the field for termination if the COBOL program does not contain a USE AFTER STANDARD ERROR PROCEDURE. When the ERROR PROCEDURE is present, the COBOL program sets the field for accept.

To override the compiler setting or to explicitly set the error option for termination, use the EO=T parameter on the FILE statement.

```
FILE(lfn,...,EO=T)
```

ACCEPT ERROR

Instead of job termination, the user or the compiler can specify that if an error occurs control passes to a user error processing routine, that is, to an error exit. This option can be taken only when the program includes an error routine; the error exit cannot be specified on a control statement.

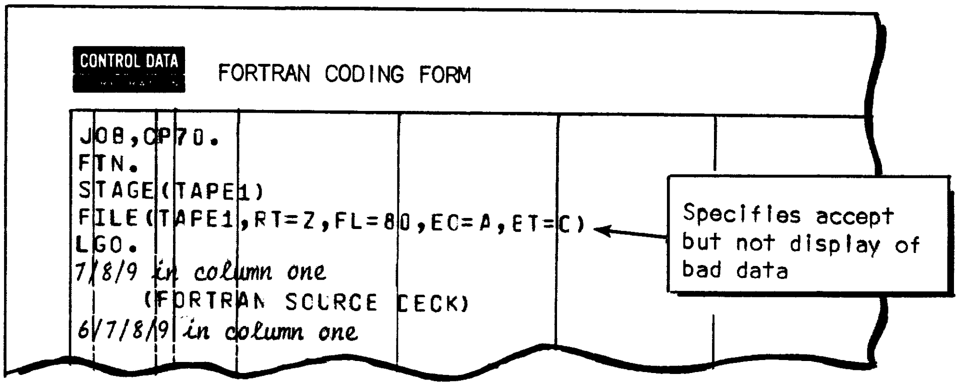
For errors other than read parity errors, the error status is indicated in the FIT and control passes to the user. For a read parity error, the record manager places the bad data in the user buffer and passes control to the user.

In addition to accepting the data, the user has the option of having it written on a file and automatically listed (displayed).

To explicitly specify accept on an error condition, specify EO=A on the FILE statement as follows:

```
FILE(lfn,...,EO=A)
```

In Example 5-8, EO=A is specified on the FILE statement to override the FORTRAN-specified accept-and-display.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN.
STAGE(TAPE1)
FILE(TAPE1,RT=Z,FL=80,EC=A,ET=C)
LGO.
7/8/9 in column one
    (FORTRAN SOURCE DECK)
6/7/8/9 in column one
```

Specifies accept but not display of bad data

Example 5-8. Specifying Error Option as Accept With No Display

DROP BAD DATA

For a read parity error, the user has the added option of having the record manager attempt to skip the bad data and resume file processing with the next data known to be good.

For a mass storage file, this means that all of the data in the sector on which a read parity error occurred is ignored.

For a magnetic tape file, and in this case a staged tape file is considered a magnetic tape file, the procedure of dropping the data is much more complex. Depending on the record and block type, the record manager attempts to save as much data in the bad block as possible. For W records with I or C blocking, the record manager attempts to locate the next W control word in the bad block and to resume processing at that point. For K and E blocking, it positions the file at the beginning of the next good block. For F records with C blocking, it calculates the next record boundary in the next block. In all cases, an error severity level is set in the FIT for use by the error routine.

To specify that data is to be dropped, enter the EO=D parameter on the FILE statement. This also causes error conditions other than read parity to be accepted.

```
FILE(lfn,...,EO=D)
```

This option requires the presence of an error routine.

DISPLAY BAD DATA

The programmer can specify when a read parity error occurs that the bad sector or block of data be written on a special list file which is automatically routed to the printer when the job ends.

Displaying is possible regardless of whether the programmer requested termination, accept, or drop.

To display bad data, append D to the EO parameters already described as follows:

EO=TD Terminate job; write data on special file.

EO=AD Accept data; write data on special file.

EO=DD Drop data; write data on special file.

MAGNETIC TAPE FILES 6

This section describes how the user can access magnetic tape files directly using on-line magnetic tape units or indirectly using staged magnetic tape units. Use of tapes, staged or on-line, requires explicit action on the part of the user. Files are never assigned to tapes by default. As noted in Section 5, tape files are always blocked sequential format. The record manager assures this by selecting a default block type whenever the file is defined as a staged or on-line file.

Tape labels, although mentioned in this section, are defined in Section 11.

STAGING TAPES

Staging is the practice of transferring all or part of a magnetic tape file to or from mass storage upon demand by the job. Staging isolates the CPU from the magnetic tape units so that more efficient CPU use can be achieved.

Tape staging requires that the user insert a STAGE control statement before the job step that first uses the file.

STAGE(lfn, p_1, p_2, . . . , p_n) and STAGE(lfn, POST, p_1, p_2, . . . , p_n)

Parameters include the following:

PRE or POST	Specifies input (PRE) or output (POST) file
MT or NT	Specifies 7-track (MT) or 9-track (NT) unit
PE, HY, HI, or LO	Specifies density (PE or HY for 9-track unit and HY, HI, and LO for 7-track unit)
VSN=vsn	Specifies volume serial number

In addition, the A, T, and U parameters described in Section 7 can appear on the STAGE statement to define mass storage characteristics such as maximum file size. If insufficient mass storage is allocated, the record manager issues the message MASS STORAGE EXCEEDED. If you receive this message, resubmit the job with An set to at least A2. The default is normally A1 (2 MAU).

For prestaging, the lfn parameter is required as a minimum. For post staging, the lfn and POST parameters are required. All optional parameters are order-independent.

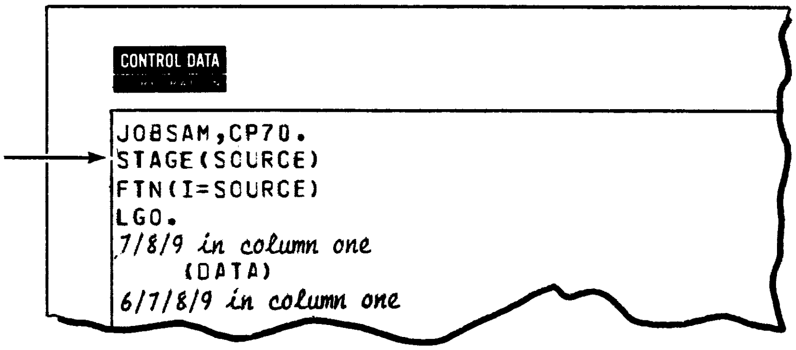
Staging does not occur at the time the STAGE statement is processed. The statement provides information to the record manager for future use. Multiple STAGE statements for a file are not allowed.

Tape staging makes the practice of extending a tape file by first skipping the information already recorded and then adding new information at the end both inconvenient and dangerous. If an error occurs during the process of rewriting the tape, information accumulated by a previous run of the job is likely to be irretrievably lost.

PRESTAGING

Prestaging is requested by default when neither PRE nor POST appears on the STAGE statement. Inclusion of the parameter PRE produces the same effect as when it is omitted.

In Example 6-1, SOURCE is prestaged for use by the compiler. It consists of a single reel (volume). The staging takes place when the compiler attempts to read from file SOURCE. The job waits for the operator to mount the tape. It resumes processing upon completion of the prestage operation.



CONTROL DATA

```
JOBSAM,CP70.
STAGE(SOURCE)
FTN(I=SOURCE)
LGO.
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

Example 6-1. Prestaging an Unlabeled Tape

POST-STAGING

NOTE

A file cannot be both prestaged and post-staged. Thus, a tape-to-tape copy of a staged file requires that the input file be prestaged and copied to another file which can then be post-staged.

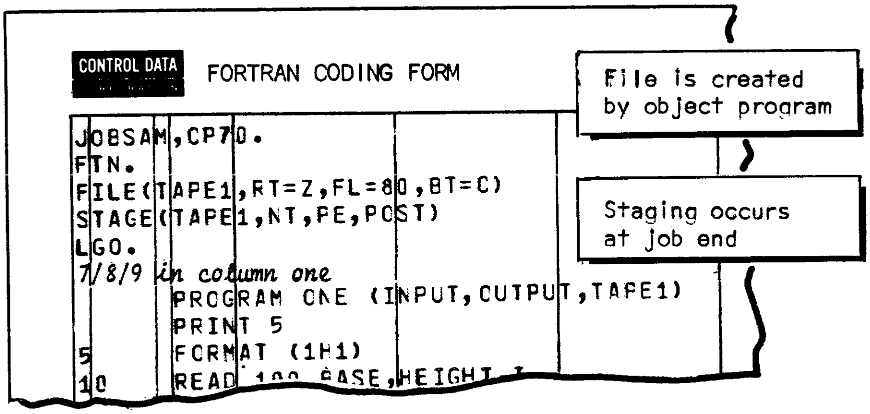
Post-staging is requested when POST appears on the STAGE statement. Upon job completion or return/unload of the file, the file is transferred to a station where it is written on one or more volumes of magnetic tape.

The operator is told to mount tapes as they are needed. For an unlabeled file, each volume is terminated by an end-of-information (double tapemark). A file cannot be both cataloged at the 6000/CYBER 70 Station and post-staged.

If abnormal job termination occurs, the file will not be staged out if it has not been created or if the fatal error causing abnormal termination involved the file.

A file is automatically rewound before it is post-staged. There is no way to post-stage part of a file.

Example 6-2 illustrates post-staging of TAPE1 to a 9-track tape unit (NT) at 1600 bpi (PE). The FILE statement describes TAPE1 as record type Z, block type C rather than the defaults of record type W, block type 1.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
FILE(TAPE1,RT=Z,FL=80,BT=C)
STAGE(TAPE1,NT,PE,POST)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100 CASE,HEIGH
```

File is created by object program

Staging occurs at job end

Example 6-2. Post-Staging an Unlabeled Tape

SPECIFYING TYPE OF TAPE UNIT

Staged magnetic tape units can be either 7-track units, 9-track units, or a combination of units.

> NOTE
>
> The 7611-11 Service Station does not support staged tape. The 7611-1 I/O Station supports 7-track units only. All other standard stations allow both types of units. Check with a systems analyst to determine what units are available.

Tapes written on 7-track units cannot be read on 9-track units and vice versa.

The presence of NT on a STAGE statement requests a 9-track unit. Otherwise, if the parameter is omitted or if MT is specified, a 7-track unit is used for staging. In Example 6-2, file TAPE1 is post-staged to a 9-track unit.

SPECIFYING TAPE DENSITY

When no density parameter is specified, the system assumes that the tape is to be read at 800 bits per inch or written at 800 bits per inch, regardless of whether the unit is 7-track or 9-track. Inserting the parameter HY on a STAGE statement has the same effect. This default can be changed by the installation.

For 7-track magnetic tape units, 800 bits per inch is usually preferred. If the tape is of questionable quality, however, you may wish to write at a lower density. Similarly, you may wish to read tapes recorded at lower densities.

For 9-track magnetic tape units, 1600 bits per inch is the preferred density. Table 6-1 summarizes magnetic tape density parameters. See Example 6-2 for an example of use.

TABLE 6-1. MAGNETIC TAPE DENSITY PARAMETERS

Parameter	7-Track	9-Track
LO	200 bpi	-
HI	556 bpi	-
HY	800 bpi	800 bpi
PE	-	1600 bpi

The density used for labels is not necessarily the same as that used for the data. On a 7-track input tape, when prestaging labeled magnetic tape files, if the system receives errors when reading the label at the density specified for labels (this density is set by the installation), it retries reading the label at other densities. It then uses the density specified in the label for reading the data.

No retries at other densities occur for unlabeled tapes.

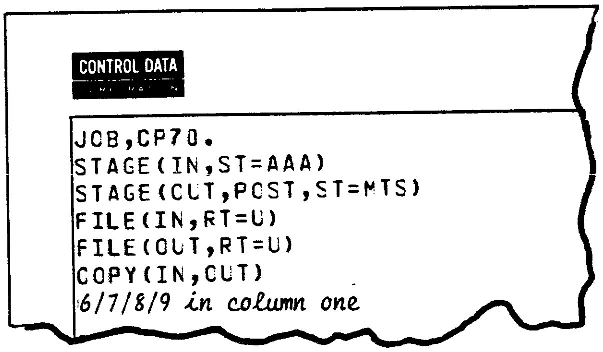
IDENTIFYING THE STATION FOR STAGING

By system default, the station that is used for staging is the station of job origin. If the job originates at a station that does not include magnetic tape units, or if the user desires to either read a tape or write a tape at some other station, he can include ST=gggttt on the STAGE statement. ggg identifies the station and ttt (optional) identifies a terminal of that station.

```
STAGE(lfn,...,ST=gggttt)
```

The system waits indefinitely for the station to be logged in if the identifier is not recognized as a currently logged in station.

In Example 6-3, the 7611-1 I/O Station has been logged in as AAA and the 7611-2 Magnetic Tape Station has been logged in as MTS. The 7611-1 operator is told to mount the input tape IN for prestaging and the 7611-2 operator is told to mount the output tape OUT for post-staging.



```
JOB,CP70.
STAGE(IN,ST=AAA)
STAGE(OUT,POST,ST=MTS)
FILE(IN,RT=U)
FILE(OUT,RT=U)
COPY(IN,OUT)
6/7/8/9 in column one
```

CONTROL DATA

Example 6-3. Identifying the Station for Staging

CHARACTER CONVERSION AND PARITY

On 7-track tapes, binary data is written in odd parity and coded data is written in even parity. On 9-track tapes, data is always written in odd parity, regardless of mode.

The recording modes (binary and coded) are functions of the tape drivers. A user can select conversion mode only through parameters on the FILE statement or through a COMPASS language subroutine. The system default is binary mode because it is more efficient than coded mode, which requires character conversion.

Where the block length is an even number of words (a multiple of 120 bits) or an even number plus 48 bits, the data records are written without modification. When the block length is an odd number of words, however, or an odd number plus 48 bits, the 9-track tape driver must supply an extra 4 bits of padding to complete the last tape frame. When the tape is read, the record manager removes the padding. In general, the record manager removes any trailing bits that do not constitute a full 6-bit character.

SCOPE 2 mode selection for FORTRAN I/O is not like previous systems which considered FORTRAN READ/WRITE statements as coded and allowed user specification of mode on BUFFER IN/OUT statements. Specification of mode on BUFFER IN/OUT statements is ignored under SCOPE 2. READ/WRITE statements will handle either binary or coded tapes depending on the CM parameter on the FILE statement. Similarly, for copy routines, the statements COPYBR/COPYBF and COPYCR/COPYCF, which implied binary or coded copies on previous systems, do not affect tape mode under SCOPE 2 (see Section 10).

For a staged tape, conversion occurs at the station when the tape is prestaged or post-staged. Conversion prohibits the occurrence of a double colon (12 bits of zero) in the coded record. Zero bytes are converted to blanks. For the same reason, do not attempt to convert W, Z, or S records. SCOPE 2 cannot read coded tapes in these record formats.

7-Track Conversion

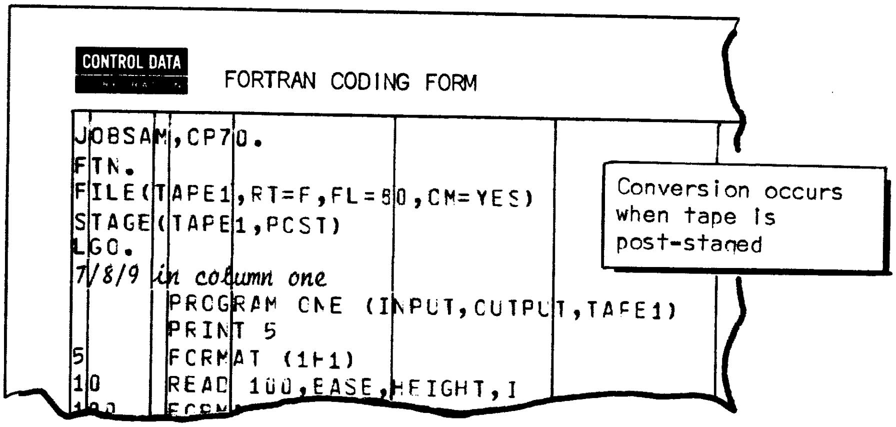
To select even parity, coded mode tape when using a 7-track magnetic tape unit, place the parameter CM=YES on the FILE statement.

```
FILE(lfn,...,CM=YES)
```

The FILE statement must precede the job step that first uses the file.

On output, CM=YES causes each binary 6 bits of data to be converted from display code to a 6-bit external BCD character (Appendix A) and recorded in even mode. On input, conversion from BCD to display takes place.

In Example 6-4, TAPE1 is post-staged in even mode with code conversion from display code to 6-bit external BCD.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
FILE(TAPE1,RT=F,FL=50,CM=YES)
STAGE(TAPE1,PCST)
LGO.
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAFE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ 100,EASE,HEIGHT,I
```

Conversion occurs when tape is post-staged

Example 6-4. 7-Track Code Conversion for Post-Staged Tape

9-Track Conversion

To select odd-parity coded-mode tape when using a 9-track magnetic tape unit, use CM=YES in the same way as for 7-track tape conversion. CM=YES on output causes each 6 bits of data to be converted from display code to an 8-bit system default (either ASCII† or EBCDIC) and recorded in odd parity. ASCII conversion can be requested explicitly by placing US on the STAGE statement.

```
FILE(lfn,CM=YES,...)
STAGE(lfn,NT,PE,US,...)
```

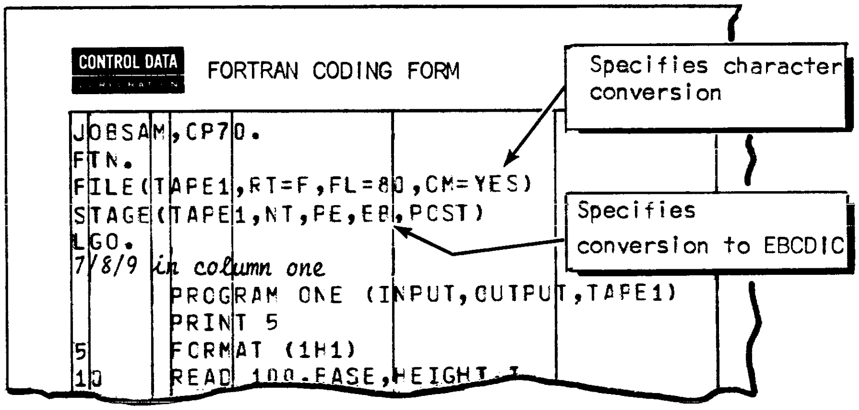
†ANSI Standard X3.4-1968 American Standard Code for Information Interchange

As an alternative, you can convert to or from a 64-character EBCDIC character set by supplying the parameter EB in place of US on the STAGE statement for the file. This parameter is relevant for data conversion only when the FILE statement specifies CM=YES and the STAGE statement specifies NT.

```
  FILE(lfn,CM=YES,...)
STAGE(lfn,NT,PE,EB,...)
```

On input, any lowercase letter is converted to uppercase. Any other character not in the 63-character subset is interpreted as a blank.

In Example 6-5, the sample program writes TAPE1 on a 9-track tape with conversion from display code to EBCDIC code.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
FILE(TAPE1,RT=F,FL=80,CM=YES)
STAGE(TAPE1,NT,PE,EB,PCST)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
```

Specifies character conversion

Specifies conversion to EBCDIC

Example 6-5. 9-Track Code Conversion for Post-Staged Tape

STAGING ALL OR PART OF FILES

The user can choose how much of his tape file is to be prestaged at a time. This affects how much of the file is concurrently accessible by the program.

The options available to the programmer are:

1. Stage-by-volume which permits an ordered progression of labeled reels (volumes) to be automatically prestaged upon completion of the previous volume.

2. Stage-by-partial volume, which permits a selected portion of a single unlabeled volume to be prestaged, and

3. Stage-by-file, which permits all volumes to be prestaged upon the first use of the file, and for the entire file to be on mass storage at the same time.

Stage by Volume

The system default causes a labeled file to be staged in a reel at a time. Each reel is called a volume. The first time the user program attempts to read the file, the record manager sends a message to the operator to mount the first volume of the file to be staged. When the operator has complied with the request, the system transfers the first or only volume to mass storage. Upon completion of the transfer, the job can begin processing the data on the mass storage copy of the file. When the program encounters the end-of-volume label, the record manager requests staging of the next volume and the process is repeated. Each staged volume overwrites the previous volume on mass storage. Staging of labeled multifile volumes is not possible.

For an unlabeled tape file, automatic volume switching does not take place. To the FORTRAN and COBOL user, this means that unlabeled tape files that exceed one volume should be staged by file, not by volume. Note, however, that if a COBOL programmer knows exactly how much data is on each volume, he can control volume switching within his program by using CLOSE REEL.

When a file is staged by volume, a rewind positions the file to the beginning of the first volume. The operator is told to remount the first volume. This allows multiple passes on a staged file. Backspacing or skipping backwards will not produce the same effect. You cannot backspace across volume boundaries.

End-of-volume on an unlabeled tape file created under SCOPE 2 consists of a double tapemark; it is the same as an end-of-information. Unlabeled volumes created under 6000 SCOPE are terminated by the end-of-volume label. Automatic volume switching is possible when using these tapes.

Using Volume Serial Numbers for Prestaging

When tapes are staged, the operator is told to mount a volume identified according to a volume serial number (vsn). The significance of the vsn differs for labeled and unlabeled tapes.

Unlabeled Tapes

In the case of unlabeled tape volumes, the vsn usually refers to a visual identifier manually provided in a sticker on the tape reel. It is provided primarily as an aid to the operator for locating the correct volume. As will be described later, it has additional significance when stage-by-file is requested.

When no VSN parameter is provided on the STAGE statement, the system uses SCRTCH as the vsn. To specify a vsn other than SCRTCH, use the following format:

STAGE(lfn, . . . , VSN=vsn_1/vsn_2/. . . /vsn_n)

vsn_i is a 1 to 6 alphanumeric character identifier to aid the operator in locating the correct tape.

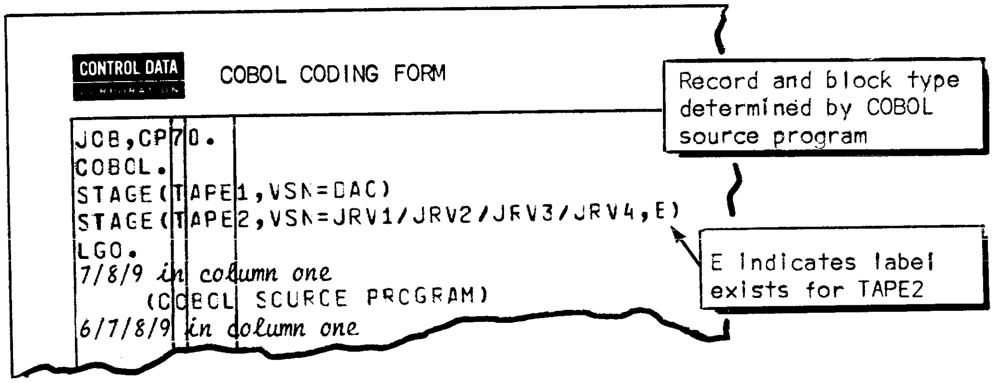
Labeled Tapes

For a labeled tape volume, the vsn refers to both the external sticker of a reel and the contents of a field in the HDR1 label (see Section 11). When no VSN parameter is supplied on the STAGE statement, the contents of the vsn field in the HDR1 label for the volume is checked against blanks. If the field contains other than blanks, an error message is issued saying that no vsn was specified on the STAGE statement. The VSN parameter must correctly list each of the vsn's for the volumes in the tape file. Otherwise, the operator must mount the correct tape or give permission to use the mounted tape.

STAGE(lfn,...,VSN=vsn_1/vsn_2/.../vsn_n,E)

Each vsn is a 1 to 6 character identifier matching the vsn used when the file was created. E indicates that a label exists (see Section 11).

Remember that the list must be long enough to encompass all the volumes of a file. In Example 6-6, TAPE1 is an unlabeled tape identified by vsn DAC; TAPE2 is a multivolume labeled tape identified by vsn's JRV1, JRV2, JRV3, and JRV4.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70.
COBOL.
STAGE(TAPE1,VSN=DAC)
STAGE(TAPE2,VSN=JRV1/JRV2/JRV3/JRV4,E)
LGO.
7/8/9 in column one
     (COBOL SOURCE PROGRAM)
6/7/8/9 in column one
```

Record and block type determined by COBOL source program

E indicates label exists for TAPE2

Example 6-6. Prestaging Using Volume Serial Numbers

Staging of Partial Volumes

The VSN parameter has an alternate application that allows portions of a volume to be staged. This feature is particularly convenient for a file that lacks a standard end-of-information. Partial staging should also be used when several partitions of a multi-partition tape file are to be processed or whenever several blocks of a large file are to be processed. Using partial staging increases throughput by decreasing mass storage requirements and staging time.

Only unlabeled tape files can make use of this feature and only portions of one volume can be staged.

The user has the choice of staging a partial volume by blocks or by tapemarks.

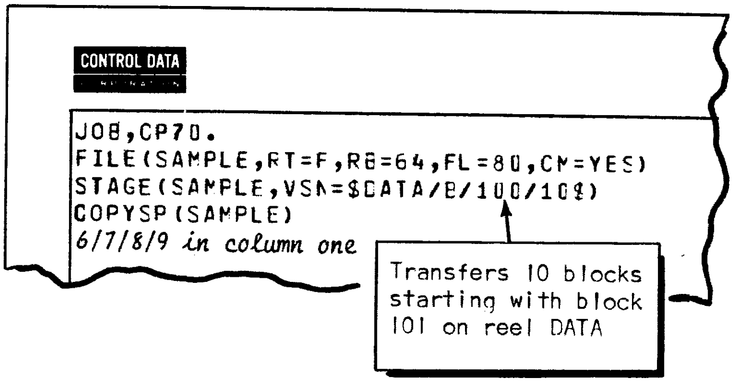
Prestaging by Blocks

To stage by blocks, use a VSN parameter with the following format:

VSN-$vsn/B/n/m$

vsn is the volume serial number (1 to 6 characters). B indicates blocks. n is the decimal count of blocks to be skipped before staging is to commence. If n is omitted, staging begins at the first block. For example, if the sixth block is the first block desired, set n to 5. The m parameter is the decimal count of blocks to be prestaged. If m is omitted, staging terminates when a tapemark is encountered.

Example 6-7 illustrates partial staging of a volume by blocks. In this example, the contents of blocks 101 through 110 (640 records) are copied to OUTPUT.


CONTROL DATA

```
JOB,CP70.
FILE(SAMPLE,RT=F,RB=64,FL=80,CM=YES)
STAGE(SAMPLE,VSN=$DATA/B/100/10$)
COPYSP(SAMPLE)
6/7/8/9 in column one
```

Transfers 10 blocks starting with block 101 on reel DATA

Example 6-7. Partial Staging by Blocks

Prestaging by Tapemarks

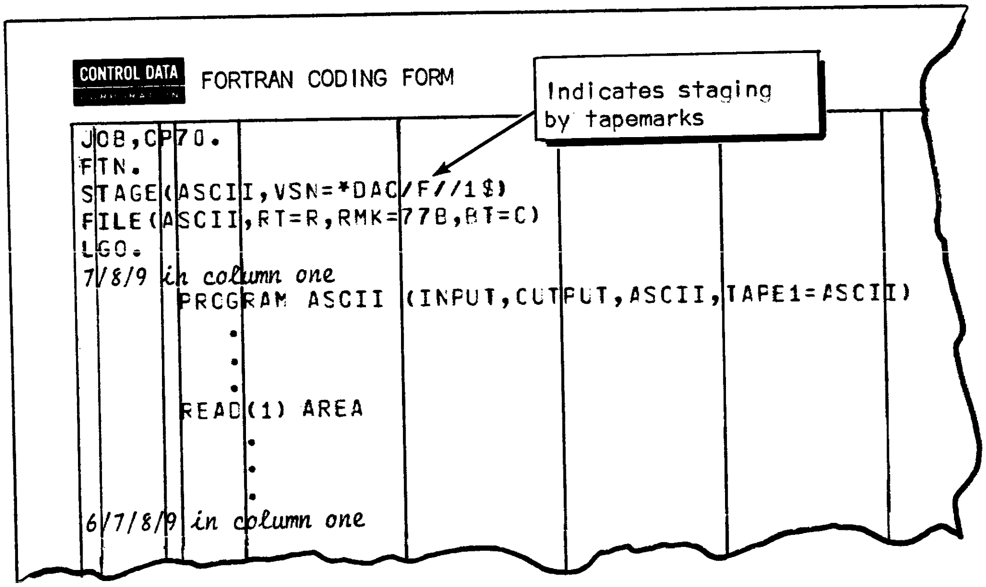
To stage by tapemarks, use a VSN parameter with the following format.

VSN=$vsn/F/n/m$

In this case, F indicates tapemarks. The parameter is a count of partitions on files with record type F, D, R, T, U, or Z with K or E blocking. For record types W, X, S, and Z with C blocking, tapemarks do not normally occur within the file.

The n parameter is the number of tapemarks to be skipped before staging can begin. m is the number of tapemarks to be prestaged.

In Example 6-8, file ASCII does not have a standard EOI but terminates with a single tapemark. It is successfully staged in by specifying staging of one tapemark.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN.
STAGE(ASCII,VSN=*DAC/F//1$)
FILE(ASCII,RT=R,RMK=77B,BT=C)
LGO.
7/8/9 in column one
      PROGRAM ASCII (INPUT,OUTPUT,ASCII,TAPE1=ASCII)
          .
          .
          .
      READ(1) AREA
          .
          .
          .
6/7/8/9 in column one
```

Indicates staging by tapemarks

Example 6-8. Partial Staging by Tapemark

Stage by File

Sometimes a user must have all volumes of a labeled tape file on mass storage concurrently so that multiple passes over the data are possible and so that the file can be repositioned meaningfully.

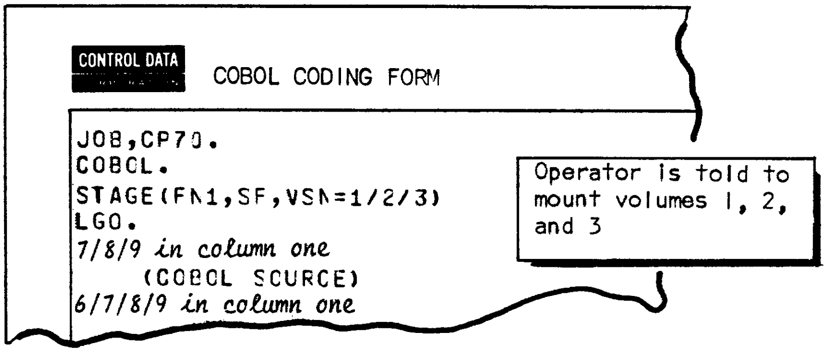
Specification of SF causes prestaging of the entire file rather than a single volume when the file is first opened. SF applies only for prestaging.

```
STAGE(lfn,...,SF)
```

If vsn's are specified, remember that you must list all vsn's comprising the multivolume file.

By using stage-by-file, it is possible to prestage a multivolume unlabeled tape file. Volumes are staged in until the list of vsn's is exhausted or until the mass storage limit is reached.

In Example 6-9, FN1 is a 3-volume unlabeled tape created under SCOPE 2.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70.
COBOL.
STAGE(FN1,SF,VSN=1/2/3)
LGO.
7/8/9 in column one
     (COBOL SOURCE)
6/7/8/9 in column one
```

Operator is told to mount volumes 1, 2, and 3

Example 6-9. Staging Entire File

Using Volume Serial Numbers for Post-Staging

When post-staging a tape file, you have the option of specifying a vsn. As for prestaging, the vsn when used for unlabeled tapes merely serves to identify the reel to be mounted by the operator. For labeled tapes, however, it results in an identifier being placed in a field in the HDR1 label.

For unlabeled tapes, when no VSN parameter is provided, the system uses SCRTCH for all volumes. For labeled tapes, when no VSN parameter is provided, a vsn is requested from the operator. The vsn supplied by the operator is then written in the vsn field in the HDR1 label.

To supply vsn's, use a parameter with the following format:

VSN=$vsn_1/vsn_2/vsn_3/\ldots/vsn_n$

For a labeled tape, be sure that there are enough vsn's specified for all volumes of the post-staged file. Otherwise, if the list is exhausted before the end-of-information is reached, the last volumes of the file will be created with operator supplied vsn's.

USING ON-LINE TAPES

Use of on-line magnetic tape units requires scheduling of the tape units on the job identification statement and insertion of a REQUEST statement prior to the job step using the unit.

NOTE

REQUEST statements for SCOPE 2 are not compatible with REQUEST statements for SCOPE 3.x.

SCHEDULING ON-LINE TAPE UNITS

To acquire on-line tape units, place one or both of the following parameters on your job identification statement.

MTd	to schedule d 7-track units
NTd	to schedule d 9-track units

In either case, d is a one or two digit octal number specifying the maximum number of units of each type that your job can use concurrently. When neither parameter is present, the job cannot use on-line tape units.

```
jobname,...,MTd,NTd.
```

The scheduling parameter does not cause the units to be assigned to the job. Your job will not be assigned the units or be charged for their use until they are requested by the job.

REQUESTING ON-LINE TAPE UNITS

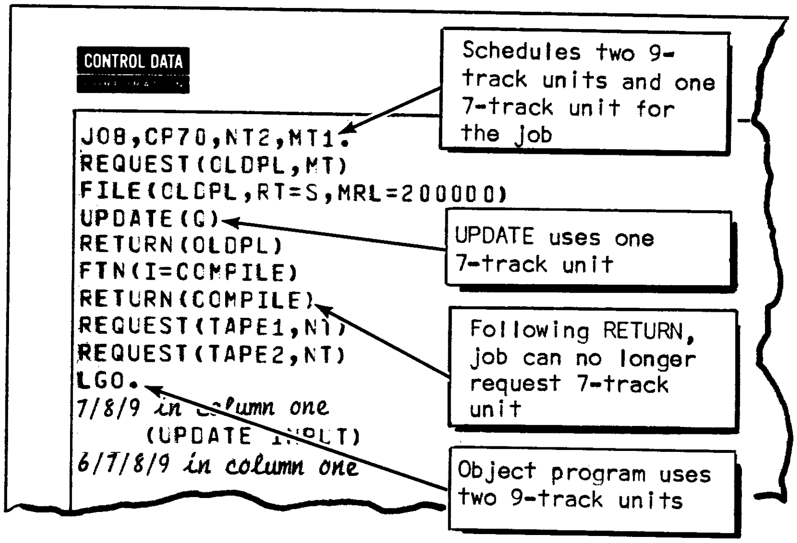
Place a REQUEST statement before the job step that requires the on-line tape file.

```
             MT
REQUEST(lfn, NT,...)
```

The MT or NT parameters must be present to distinguish the REQUEST from a mass storage REQUEST. MT requests a 7-track unit; NT requests a 9-track unit.

A new request may be issued for a file after the previously requested file has been unloaded. Otherwise, multiple REQUEST statements are not allowed for a file. A STAGE statement and a REQUEST statement cannot be used concurrently for the same file.

Example 6-10 illustrates a job that uses a 7-track unit and two 9-track units.



```
JOB,CP70,NT2,MT1.
REQUEST(OLDPL,MT)
FILE(OLDPL,RT=S,MRL=200000)
UPDATE(C)
RETURN(OLDPL)
FTN(I=COMPILE)
RETURN(COMPILE)
REQUEST(TAPE1,NT)
REQUEST(TAPE2,NT)
LGO.
7/8/9 in column one
    (UPDATE INPUT)
6/7/8/9 in column one
```

Schedules two 9-track units and one 7-track unit for the job

UPDATE uses one 7-track unit

Following RETURN, job can no longer request 7-track unit

Object program uses two 9-track units

Example 6-10. Scheduling and Requesting On-Line Tape Units

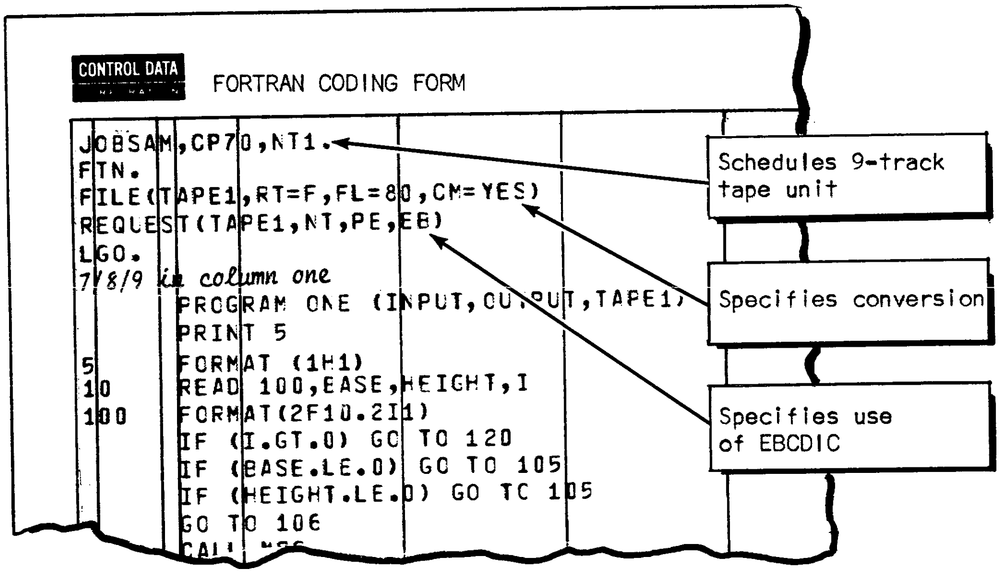
SPECIFYING TAPE DENSITY FOR ON-LINE TAPES

Tape density parameters serve the same purpose for on-line tapes as they do for staged tapes. See page 6-4.

CHARACTER CONVERSION AND PARITY

Character conversion for on-line tapes follows the same conversions as for staged tapes. For on-line tapes, however, the tape driver performs conversion on each physical read or write of the tape. See page 6-6.

Example 6-11 illustrates use of a 9-track on-line tape with conversion to EBCDIC code.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70,NT1.
FTN.
FILE(TAPE1,RT=F,FL=80,CM=YES)
REQUEST(TAPE1,NT,PE,EB)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,EASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
      IF (HEIGHT.LE.0) GO TC 105
      GO TO 106
      CALL
```

Schedules 9-track tape unit

Specifies conversion

Specifies use of EBCDIC

Example 6-11. 9-Track Code Conversion for On-Line Input Tape

POSITIONING ON-LINE MAGNETIC TAPE FILES

A rewind of an on-line magnetic tape file positions the tape at load point for the current volume. To rewind to the beginning of the first volume, close and unload the file and re-request the first volume.

Similarly, you cannot backspace or skip backward across volume boundaries.

When processing the file in the forward direction, upon encountering end-of-volume on the current reel, the system rewinds and unloads the reel and issues a request to the operator to mount the next volume.

The VF parameter on the FILE statement allows a user to override the system default. VF requests that the volume be simply rewound, or neither rewound nor unloaded when end-of-volume is reached.

```
FILE(lfn,...,VF=x)
```

Use VF=R to specify rewind and use VF=N to specify no rewind. VF=U is the same as the system default which specifies rewind and unload.

USING VOLUME SERIAL NUMBERS WITH ON-LINE TAPES

Volume serial numbers serve the same purpose for on-line tapes as for staged tapes. That is, for unlabeled tapes, they serve to aid the operator in identifying the reel to be mounted and for labeled tapes they aid in checking or creating the vsn field of the HDR1 labels (Section 11).

The parameter has the same form as for staged tapes.

VSN=$\text{vsn}_1/\text{vsn}_2/\text{vsn}_3/\dots/\text{vsn}_n$

However, the alternate form of vsn allowed for partial prestaging (VSN=$vsn/t/n/m$) has no application for on-line tapes.

MOUNT OPTION

To reduce the time required for operator intervention on multivolume files, you can specify M2 to request that two tape volumes be mounted concurrently. The system alternately accesses volumes from the two units.

```
            MT
REQUEST(lfn,  ,M2,...)
            NT
```

When using this parameter, schedule enough tape units of the required type on your job identification statement. Remember that you will be charged for both units until you return the file. When the parameter is omitted or specified as M1, only one volume can be mounted at a time. The only allowable forms of the parameter are M1 and M2.

SUPPRESSING READ—AHEAD/WRITE—BEHIND

Usually, the record manager remains a step ahead of the user when he is reading or writing an on-line tape. That is, if your job reads an on-line tape, the record manager acquires the next block and places it in an LCM buffer before it is requested. If your job is writing an on-line tape, the record manager lags the write request by at least one block. This technique, sometimes called multiple buffering, requires additional buffer space in LCM, but expedites processing when the tape is being read sequentially. Multiple buffering may not be desirable when the job is not accessing the tape file sequentially or is frequently repositioning the file. You might prefer to reduce LCM requirements by suppressing the read-ahead/write-behind processing.

To suppress the read-ahead/write-behind processing and cause system reads and writes to be synchronized with those in your program, place the SPR=YES parameter on the FILE statement for the file.

```
FILE(lfn,...,SPR=YES)
```

If the parameter is omitted or if SPR=NO is used instead, normal read-ahead/write-behind processing takes place.

UNLOADING/RETURNING ON—LINE MAGNETIC TAPE UNITS

Occasionally, you may want to rewind and unload an on-line magnetic tape file before the job has completed processing. To do this without reducing the number of tape units scheduled for your job on the job identification statement, use the UNLOAD statement.

```
UNLOAD(lfn₁,lfn₂,...,lfnₙ)
```

Suppose that at least one of the lfn's is the name of an on-line magnetic tape file. Following the UNLOAD, your job can use a REQUEST statement to acquire the magnetic tape unit for a different file.

Do not use UNLOAD if the job no longer has a need for the units on which the files are mounted. Use RETURN, instead.

```
RETURN(lfn₁,lfn₂,...,lfnₙ)
```

While both RETURN and UNLOAD reduce the number of active tape units assigned to your job, RETURN is preferable because it reduces the needs of the job and may allow it to complete processing sooner.

In either case, returning or unloading the file causes the file to be detached from your job. If you UNLOAD your unit and your job is not the job with the lowest tape unit requirements, the unit may be reassigned to another job.

Example 6-10 illustrates a job that returns a 7-track tape unit to the system. Note that the job identification statement has a request for one 7-track unit and two 9-track units. Following the RETURN, only 9-track units can be used by the job.

MAGNETIC TAPE RECOVERY PROCEDURES

The action taken by the tape driver when it encounters a parity error on a read or write is called a recovery procedure. The procedures are different for each station and for on-line tapes.

STANDARD RECOVERY PROCEDURES

Procedures for on-line tapes and the 6000 Station generally follow the CDC Magnetic Tape Error Recovery Standard 1.87.004. The 7611-1 I/O Station does not follow these standards. The 7611-2 Magnetic Tape Station procedures deviate from the standard, especially in the area of system noise blocks. (See the 7611-2 Operator's/Reference Manual.) No recovery is attempted by the 7611-1 I/O Station.

If recovery is not possible, the operator is given control to retry the procedures, to accept the data, or to terminate the job.

Specifying No Recovery

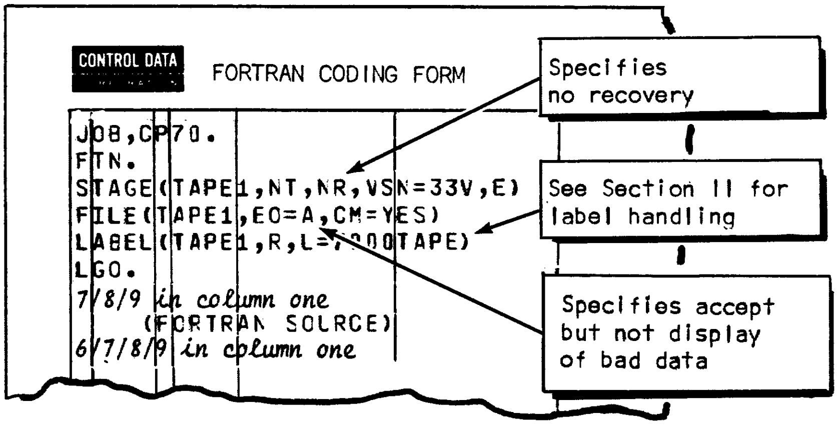
On a tape read parity error, under some conditions the user may elect to accept the bad block. When reducing analog data, for example, the presence of a bad block may not adversely affect the results.

Place NR on the STAGE or REQUEST statement to notify the driver that it is not to attempt recovery of a read parity error but is to pass the data to the CPU.

```
REQUEST(lfn,...,NR)          or          STAGE(lfn,...,NR)
```

If NR is specified or if the operator has authorized continued processing, the action taken by the record manager when it encounters the read parity error depends on the setting of the error option (EO) field in the FIT for the file as described in Section 5.

In Example 6-12, NR is specified on the STAGE statement and EO=A is specified on the FILE statement.



```
JOB,CP70.
FTN.
STAGE(TAPE1,NT,NR,VSN=33V,E)
FILE(TAPE1,EO=A,CM=YES)
LABEL(TAPE1,R,L=7000TAPE)
LGO.
7/8/9 in column one
      (FORTRAN SOLRCE)
6/7/8/9 in column one
```

Specifies no recovery

See Section 11 for label handling

Specifies accept but not display of bad data

Example 6-12. No Recovery and Accept Data Options

Noise Blocks

NOTE

> A tape written using standard error recovery procedures may not be acceptable for input at a station such as the 7611-1 because of the presence of system noise blocks. Similarly, a data tape prepared on some other computer system could have valid blocks composed of 8 or less characters. When standard recovery is used these short blocks will be assumed to be noise blocks and will be filtered out of the data.

When standard error recovery procedures are in effect, blocks less than a specified minimum are considered noise and are not passed to the CPU. For 7-track tapes, blocks shorter than 8 characters are considered noise; for 9-track tapes, blocks shorter than 6 characters are considered noise.

If noise blocks are not removed from the file by a station, they are passed to the CPU. The record manager attempts to read them as if they were good data.

MASS STORAGE FILES 7

The only storage and peripheral devices that your program can directly access are system mass storage devices and on-line tape units. This section describes how space is allocated on system mass storage.

System mass storage offers efficient, easy-to-use, data storage capacity. The only mass storage device currently available is the 7638 Disk Storage Subsystem. These are nonremovable disk files. This section will be expanded as other devices are added to the system.

HOW MASS STORAGE FILES ORIGINATE

An output file referenced by your job is assigned to system mass storage unless you explicitly define the file as being an on-line magnetic tape file. That is, the system default always assigns an output file to mass storage. Subsequently, a job can use the mass storage file in any of the following ways:

1. It can use the file as a temporary scratch file that provides input to the same program or to some later job step and is then released. The LGO file is a very common example of a scratch file.

2. The job can use the file as a scratch file and then catalog it as a permanent file before job end.

3. The job can post-stage the file to magnetic tape. In this case, the STAGE statement requesting post-staging must precede the first reference to the file.

4. The job can route the file to a printer or punch.

SCOPE 2 assigns mass storage and associated LCM buffers to the file before writing on the file.

Similarly, an input file referenced by your job is assumed by the system to be on mass storage unless your job has explicitly defined the file as being an on-line magnetic tape file. The file may have originated in one of the following ways:

1. It may have originated from punched cards read in as the INPUT portion of the job deck.

2. It may be a prestaged magnetic tape file. The STAGE statement describing the file as prestaged must precede the first reference to the file. The first attempt to open the file causes all or part of the file to be transferred from the station to mass storage.

3. The input file may be a permanent mass storage file. In this case, an ATTACH statement precedes the first use of the file. For a CYBER 70/6000 Station permanent file, the first attempt to open the file for input causes the file to be transferred from the station to SCOPE 2 mass storage.

4. The file may be a scratch file created earlier in the same job.

INTRODUCTION TO REQUEST STATEMENT

When a file is on mass storage, the user has the option of supplying a REQUEST statement to define characteristics relating to mass storage usage. The REQUEST statement differs from that for on-line magnetic tapes in that no device type is specified. The control statement has the following format:

REQUEST(lfn, p_1, p_2, p_n)

p_i	Significance
An	Allocation unit
Tn	Transfer unit size
Un	Mass storage unit assignment
WCK	Write check

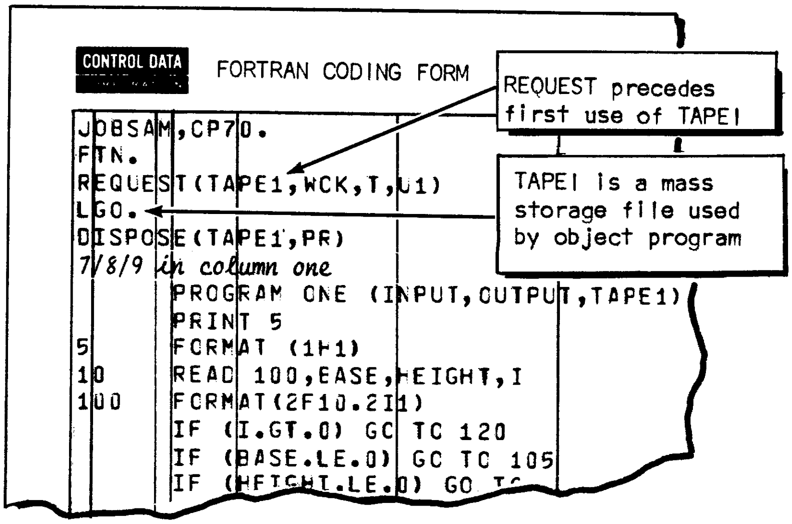
The preceding parameters, which can be in any order, are described in greater detail later in this section.

The same parameters that appear on a mass storage REQUEST statement can also appear on a STAGE statement and affect the mass storage allocated for the file. The REQUEST statements and STAGE statements are mutually exclusive, that is, you cannot use both statements for a file.

The FILE statements and REQUEST statements are complementary, however, and often appear together for a file. They can be in any order but the REQUEST statement must be in the job control section before the program that first accesses the file. Example 7-1 illustrates REQUEST statement placement.

NOTE

Multiple REQUEST statements are not allowed for a file.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
REQUEST(TAPE1,WCK,T,U1)
LGO.
DISPOSE(TAPE1,PR)
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ 100,EASE,HEIGHT,I
100   FCRMAT(2F10.2I1)
      IF (I.GT.0) GC TC 120
      IF (BASE.LE.0) GC TC 105
      IF (HEIGHT.LE.0) GO T
```

REQUEST precedes first use of TAPE1

TAPE1 is a mass storage file used by object program

Example 7-1. REQUEST Statement Placement

ASSIGNMENT OF SPACE ON MASS STORAGE

The record manager assigns space to a mass storage output file as needed. Through parameters of the REQUEST statement, the user can control the size of the transfers and the amount of mass storage allocated on each mass storage assignment. The user can also specify that the file be assigned to a specific mass storage unit.

MINIMUM ALLOCATION UNITS

The unit of storage used for assignment of mass storage is called a minimum allocation unit (MAU) (see Figure 7-1). An MAU consists of 25,600 characters (five sectors on the 7638 Disk Storage Subsystem).

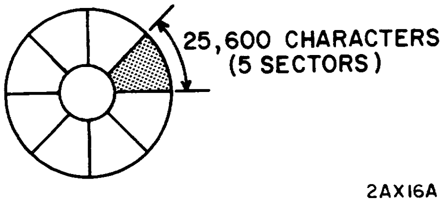


Figure 7-1. Minimum Allocation Unit

The first time a file is opened, one or more contiguous MAUs are allocated to it. More space is allocated as it is used. The number of MAUs assigned at a time is determined by the following factors:

1. For the INPUT file and permanent files attached from the 6000/CYBER 70 Station, the number of MAUs is determined by an installation option.

2. The number of MAUs can be specified through use of an An parameter on a REQUEST statement or STAGE statement.

   REQUEST(lfn,...,An) or STAGE(lfn,...,An)

3. The number of MAUs can be set by system default if no An parameter is specified. The default is normally set for one unit.

As shown in Table 7-1 and Figure 7-2, the number of units allocated increases as an exponent of 2 as you increase n. The allocation parameter impacts the maximum size allowed for a file because SCOPE 2 allows space to be allocated a maximum of 1012 times. (The limit is imposed by the size of a table which has a maximum of 1012 entries.)

TABLE 7-1. RELATION OF ALLOCATION PARAMETER TO FILE SIZE

ALLOCATION SIZE					MAXIMUM FILE SIZE		
An	MAUs	Sectors	Tracks	Characters	Sectors	Tracks	Characters
A0	1	5	1/8	25,600	5060	126.5	25,907,200
A1	2	10	1/4	51,200	10,120	253	51,814,400
A2	4	20	1/2	102,400	20,240	506	103,628,800
A3	8	40	1	204,800	40,480	1012	207,257,600
A4	16	80	2	409,600	80,960	2024	414,515,200

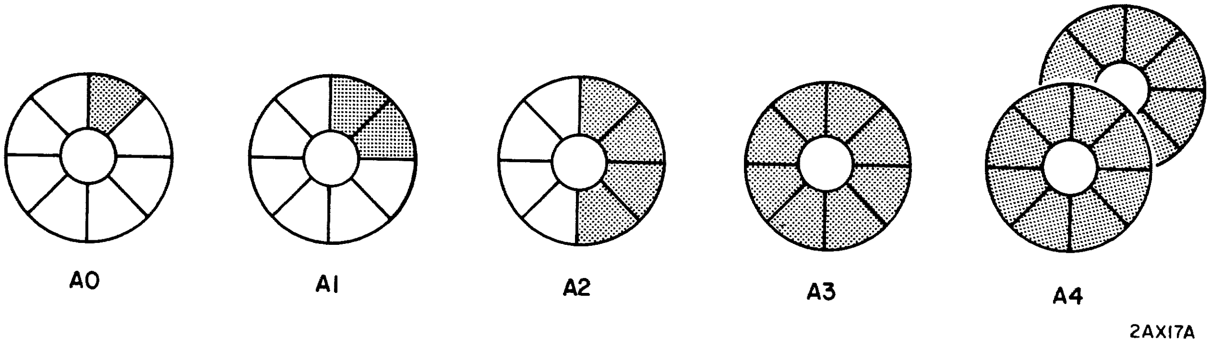
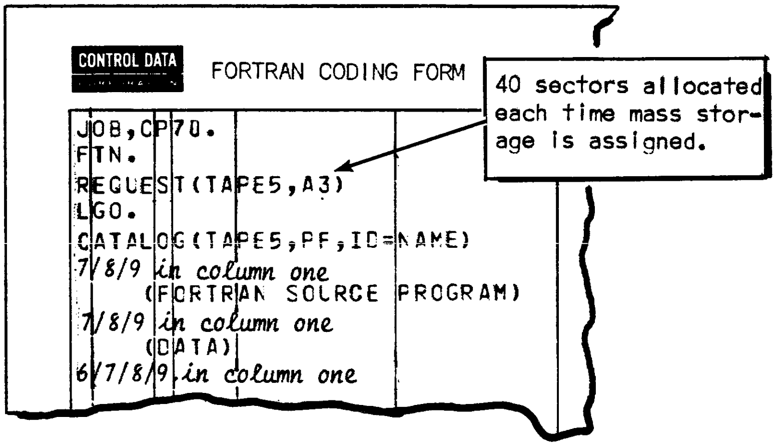


Figure 7-2. MAUs Assigned as a Result of Allocation Parameter

When specifying the An parameter, keep in mind that your job is charged for all allocated mass storage. Thus, if An is very large, and you use but a small portion of the last set of MAUs allocated, you will be charged for a great deal of unused space. This condition is even more undesirable when applied to a permanent file because the wasted space is tied up indefinitely.

Example 7-2 shows a REQUEST statement for a FORTRAN output file (TAPE 5) which the user knows is going to be very large.



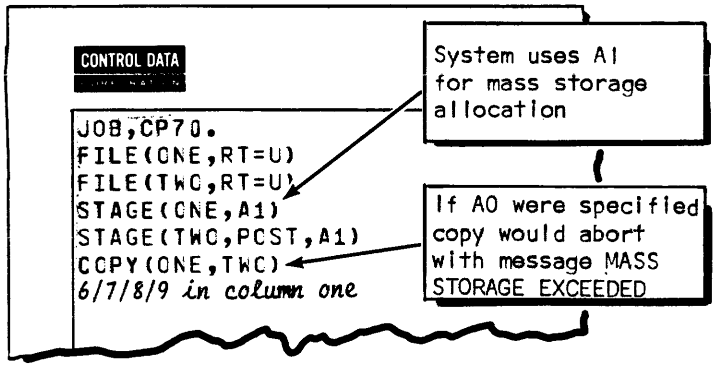
FORTRAN CODING FORM

```
JOB,CP70.
FTN.
REQUEST(TAPE5,A3)
LGO.
CATALOG(TAPE5,PF,ID=NAME)
7/8/9 in column one
      (FORTRAN SOURCE PROGRAM)
7/8/9 in column one
      (DATA)
6/7/8/9 in column one
```

40 sectors allocated each time mass storage is assigned.

Example 7-2. Using the REQUEST Statement Allocation Parameter

Example 7-3 shows a tape-to-tape copy of a full 2400-foot reel at 800 bits per inch. If A0 were used as the An parameter for the post-staged file, the file could be staged in but not staged out since A0 is not adequate for containing a full reel of tape.



```
JOB,CP70.
FILE(ONE,RT=U)
FILE(TWO,RT=U)
STAGE(ONE,A1)
STAGE(TWO,POST,A1)
COPY(ONE,TWO)
6/7/8/9 in column one
```

System uses A1 for mass storage allocation

If A0 were specified copy would abort with message MASS STORAGE EXCEEDED

Example 7-3. Using the STAGE Statement Allocation Parameter

TRANSFER UNIT SIZE

The user can specify the size of the LCM buffer for a mass storage file. He does this by specifying the Tn parameter on the REQUEST statement where n is omitted or is 1 to 4. The smallest unit that can be requested is one sector; the largest is 80 sectors. The allocation parameter (An) affects the size you can specify for Tn because the amount of data transferred cannot exceed the amount of space allocated at a time. However, less data can be transferred and you can reduce your LCM requirements by specifying a transfer unit smaller than the number of allocation units.

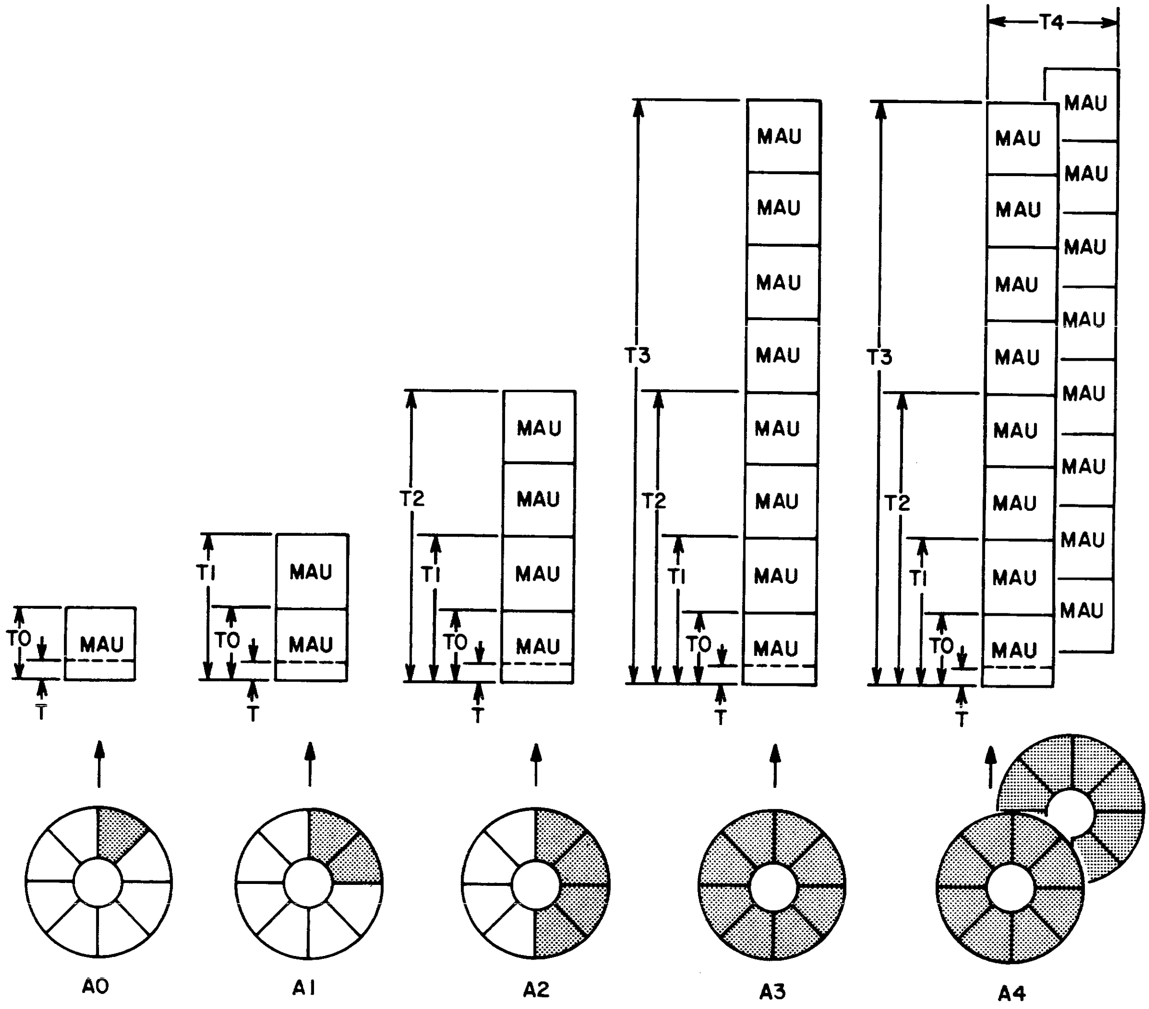
```
REQUEST(lfn,...,Tn)
```

Transfer Parameter	Buffer Area in Characters	
T	5,120	Allocation can be A0, A1, A2, A3, or A4
T0	25,600	
T1	51,200	Allocation can be A1, A2, A3, or A4
T2	102,400	Allocation can be A2, A3, or A4
T3	204,800	Allocation can be A3 or A4
T4	409,600	Allocation must be A4

If no Tn parameter is specified, the system uses the An parameter as the default for the Tn parameter. In this case, the LCM buffer size is equal to the sum of the MAUs allocated at a time. Space is allocated for each mass storage transfer. This relationship is shown in Table 7-2 and Figure 7-3.

TABLE 7-2. RELATIONSHIP OF AN AND TN PARAMETERS

Tn An	T	T0	T1	T2	T3	T4
A0	5120	25,600				
A1	5120	25,600	51,200			
A2	5120	25,600	51,200	102,400		
A3	5120	25,600	51,200	102,400	204,800	
A4	5120	25,600	51,200	102,400	204,800	409,600



2AX18A

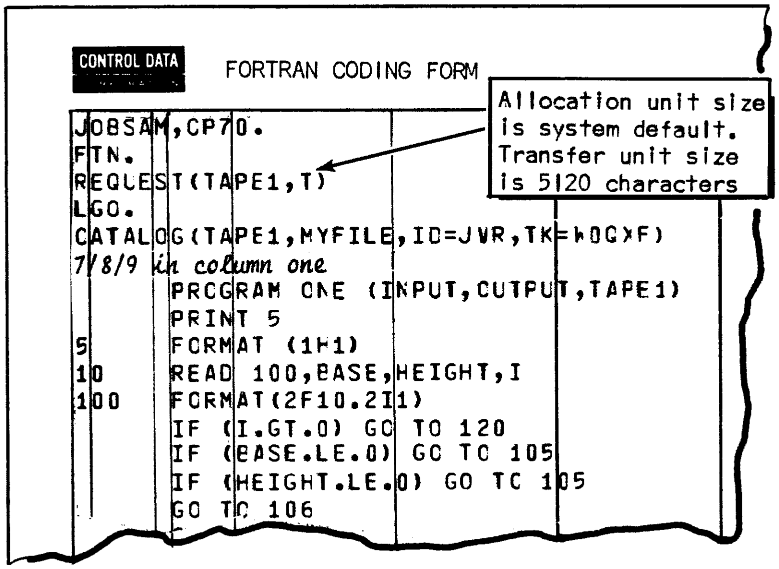
Figure 7-3. Relationship of MAUs and Transfer Unit Size

The An and Tn parameters must be considered on an input file as well as an output file. For a permanent file attached from SCOPE 2 mass storage, Tn cannot be set larger than the allocation parameter used when the file was created. The An parameter must be the same as when the file was created. If the allocation parameter is not readily known, it may have to be obtained by trial and error. However, the system default is usually adequate.

For the INPUT file and permanent files attached from the 6000/CYBER 70 Station, the transfer unit size equals the allocation units size defined by system default.

Usually, the system default (A0,T0) is the most efficient. The Tn parameter can also be used on a STAGE statement for a staged file. Referring back to Example 7-3, the transfer unit size for both files ONE and TWO is two MAUs by default.

In Example 7-4, notice that there is a REQUEST statement for TAPE1.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
REQUEST(TAPE1,T)
LGO.
CATALOG(TAPE1,MYFILE,ID=JVR,TK=NOGXF)
7/8/9 in column one
      PRCGRAM CNE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GC TO 120
      IF (BASE.LE.0) GC TC 105
      IF (HEIGHT.LE.0) GO TC 105
      GO TC 106
```

Allocation unit size is system default. Transfer unit size is 5120 characters

Example 7-4. Using the REQUEST Statement Transfer Unit Parameter

MASS STORAGE UNIT ASSIGNMENT

The Un parameter (where n is an octal unit number) on the REQUEST statement is provided in the event that a user wishes his mass storage file assigned to a particular mass storage unit.

```
REQUEST(lfn,...,Un)
```

If the 7638 Advanced Disk Subsystems are installed so that they are not all the same speeds, you may want to explicitly request the higher speed unit. Some installations may also prefer that certain types of files (for example, permanent files) be directed to a specific unit.

Unit number assignments and default assignments are installation dependent and will vary from site to site. Generally, a systems analyst can tell you assignments at your site. If the specified unit is not available, the job aborts.

The Un parameter can also be used on a STAGE statement.

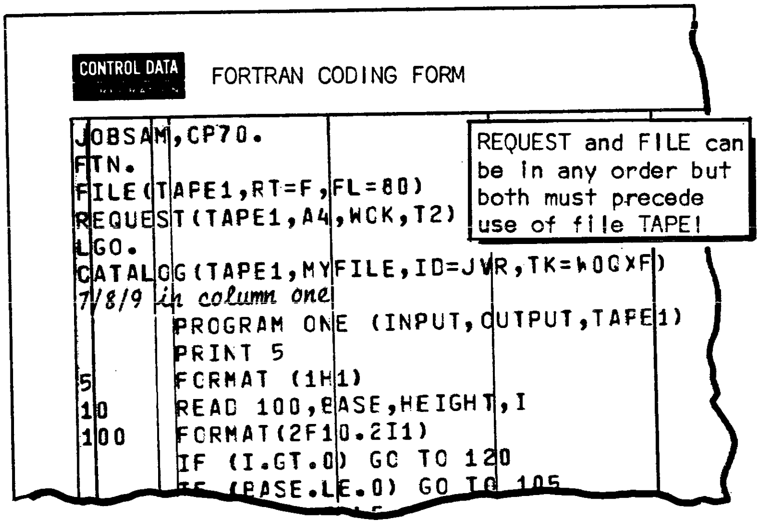
WRITE CHECK OPTION

An option of the REQUEST statement permits a user to request that the record manager follow each write with a read. If the read fails, the record manager aborts the job. The feature provides slower I/O but reduces the possibility of wasting a great deal of time in generating a bad file.

To specify write check, include WCK in the parameter list for the REQUEST statement for the file.

```
REQUEST(lfn,...,WCK)
```

Example 7-5 illustrates a FORTRAN program that uses WCK on file TAPE1. Here, use of WCK could prevent cataloging of a useless file. There is no way to recover the file following the abort.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
FILE(TAPE1,RT=F,FL=80)
REQUEST(TAPE1,A4,WCK,T2)
LGO.
CATALOG(TAPE1,MYFILE,ID=JWR,TK=W0GXF)
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GC TO 120
      IF (BASE.LE.0) GO TO 105
```

REQUEST and FILE can be in any order but both must precede use of file TAPE!

Example 7-5. Use of the REQUEST Statement Write Check Parameter

WCK is also allowed on a post-stage STAGE statement. This feature allows mass storage errors to be detected as the file is written rather than later when the completed file is to be staged out.

RETURNING MASS STORAGE FILES

You can cause one or more files and their resources to be returned to the system by using a RETURN statement. Reasons for doing so include the following:

1. Receive a partial output of a very large file
2. Assure that output is not lost if an abort occurs later in the job
3. Reduce resources used by the job, thus reducing job cost and promoting efficient use of system resources
4. Post a dependency (TRANSF)
5. Close a blocked permanent file and reattach it with some other description

INPUT is the only file that cannot be returned.

Use a RETURN statement, as follows:

RETURN(lfn_1, lfn_2, lfn_3, . . . , lfn_n)

Returning a scratch file causes its mass storage space to be released. Returning an implicitly disposed file (for example, OUTPUT, PUNCH, or PUNCHB) causes the accumulated data to be disposed. If the file is reopened, a file with the same name and disposition code becomes available. An implicitly disposed file can be returned many times in the same job.

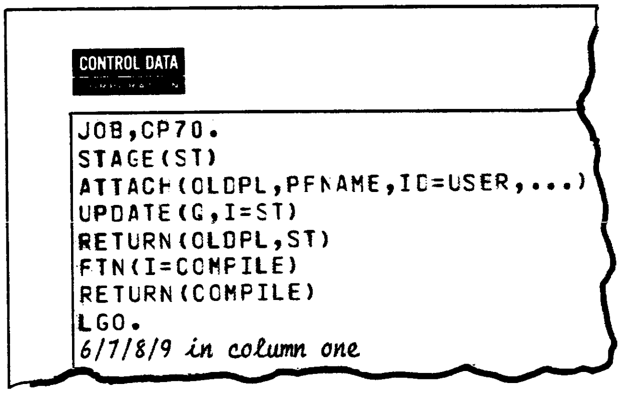
Returning a file routed with a delayed DISPOSE statement (see Section 9) causes the file to be immediately disposed to a station unit record device.

Returning a post-staged file causes immediate staging out of the file. Returning a prestaged file causes its mass storage space to be released. Returning a permanent file causes it to be detached from the job. If the file is a 6000/CYBER 70 Station permanent file, the 7600 mass storage used for the working copy is released. If a new cycle is to be cataloged, the cycle is staged to the 6000/CYBER 70 Station. Returning an attached 7600 permanent file causes it to be detached from the job. If it was attached with permissions other than read, returning the file makes it available for other users.

An UNLOAD statement has the same effect as RETURN for mass storage files.

When a file is returned, its LCM buffers are released.

Example 7-6 illustrates a job that returns files ST, OLDPL, and COMPILE when they are no longer needed by the job.



```
JOB,CP70.
STAGE(ST)
ATTACH(OLDPL,PFNAME,ID=USER,...)
UPDATE(G,I=ST)
RETURN(OLDPL,ST)
FTN(I=COMPILE)
RETURN(COMPILE)
LGO.
6/7/8/9 in column one
```

Example 7-6. Returning Mass Storage Files

JOB MASS STORAGE LIMIT

SCOPE 2 provides a LIMIT statement which can be used as a precaution against having a programming error occur that would result in an abnormal amount of mass storage being used.

```
LIMIT(n)
```

LIMIT acts on a job basis; n specifies the number of multiples of 40960 characters of mass storage that can be assigned to all the files in your job before the job will abort.

If no limit is defined, a default value set by the installation is used.

PERMANENT FILES 8

A permanent file is a 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 become immediately available to requesting jobs. Permanent files cannot be destroyed accidentally during normal system operation including deadstart; they are protected by the system from unauthorized access according to the privacy controls specified when they are created.

Any mass storage file available to your job that is not already permanent can be made permanent. A mass storage file is not made permanent unless you explicitly specify that it be made permanent through a CATALOG request. You may acquire (attach) a permanent file from either of two sources: SCOPE 2 system mass storage or SCOPE 3.x mass storage at the CYBER 70/Model 72, 73, or 74 (or 6000 Series) Station if your system has one. You may also catalog a permanent file at either location.

USING SCOPE 2 PERMANENT FILES

CYCLES

The SCOPE 2 permanent file manager maintains one to five separate and distinct permanent files under each permanent file name. Each one of these separate files is called a cycle and is identified by the permanent file name and a cycle number. All cycles of a permanent file share the access restrictions (permissions) applicable to that permanent file name. Most commonly a new cycle is a more recent version of a previous cycle.

CYCLE NUMBERS

A cycle number is a decimal number (1 to 63, inclusive). Duplicate cycle numbers under a single permanent file name are not allowed. If you do not specify a cycle number when you catalog a file, the permanent file manager assigns cycle number 1. If you fail to specify a cycle number when you attach a permanent file, the cycle with the largest number is attached.

LOGICAL AND PERMANENT FILE NAMES

Each permanent file has a permanent file name under which its entry in the permanent file directory is listed. Permanent file statements that associate a permanent file with a user's program must include a logical file name. User programs reference logical file names only.

The CATALOG statement declares a logical file name (lfn) to be a permanent file under the permanent file name (pfn). pfn is 1 to 40 alphanumeric characters. If characters other than 0 through 9 or A through Z are desired, the special characters occur in a literal (that is, they must be preceded and followed by a dollar sign. The dollar signs do not become part of the name).

CREATOR IDENTIFICATION

The creator identification is a 1 to 9 alphanumeric character identifier entered into the permanent file directory that identifies the creator of the permanent file. The purpose of the

ID parameter is to supply a file owner/user name to the permanent file manager. It is recommended that an ID for a permanent file be defined when the first cycle of the file is cataloged. Examples in this guide assume that ID parameters are defined.

SCOPE 2 has no IDs reserved in contrast to SCOPE 3.x which assigns special meaning to the IDs of PUBLIC and SYSTEM.

MULTIPLE READ ACCESS

Any number of users may simultaneously attach any number of cycles of a permanent file for read-only access. Any user that attaches a cycle for a purpose other than reading obtains exclusive access to that cycle. If the cycle is in use when the request for the cycle is made, the job must wait for the cycle to become available. If others request the cycle while a user has exclusive access, they must wait until the job with exclusive access releases that cycle of the permanent file. Multiread access is possible only when the extend, modify, and control passwords are established for the file.

CATALOGING AN INITIAL CYCLE

Cataloging a permanent file merely means declaring a logical file associated with the creator's program to be a permanent file. Cataloging is accomplished by the CATALOG statement which creates one cycle of a file at a time. CATALOG creates an entry for the permanent file in the permanent file directory. This entry includes the permanent file name, the cycle number, the creator identification, and any passwords defined by the creator of the file.

As the creator of a permanent file you can grant or withhold permissions. When you catalog the initial cycle of a permanent file, you can supply passwords protecting the file from unauthorized operations such as increasing the size of the file, writing over existing data, cataloging a new cycle, or purging an existing cycle. If you do not specify any passwords when you catalog the initial cycle, any program that attaches the file has exclusive access to the file with all permissions. If you specify passwords covering one or more operations, anyone to whom you reveal the passwords can exercise the permissions that the passwords represent. If you specify all passwords but reveal none of them, only you as the creator has any of the permissions relating to the permanent file.

In this way, the creator of the initial cycle of a permanent file determines how permissions to deal with all cycles under that permanent file name are granted or withheld. Those who wish to perform operations on that cycle or any other cycle of the permanent file use the passwords they know to establish their rights to access the file.

If you catalog a file, the permanent file becomes available to other jobs when your job terminates or when your job releases the file through a RETURN or UNLOAD statement. Until then, the job remains attached to your job with all permissions granted.

Catalog With No Passwords

If you have a relatively limited or short-term need for a permanent file and are not concerned about file privacy or multiread access, you may want to catalog a file with no password requirements using the following control statement.

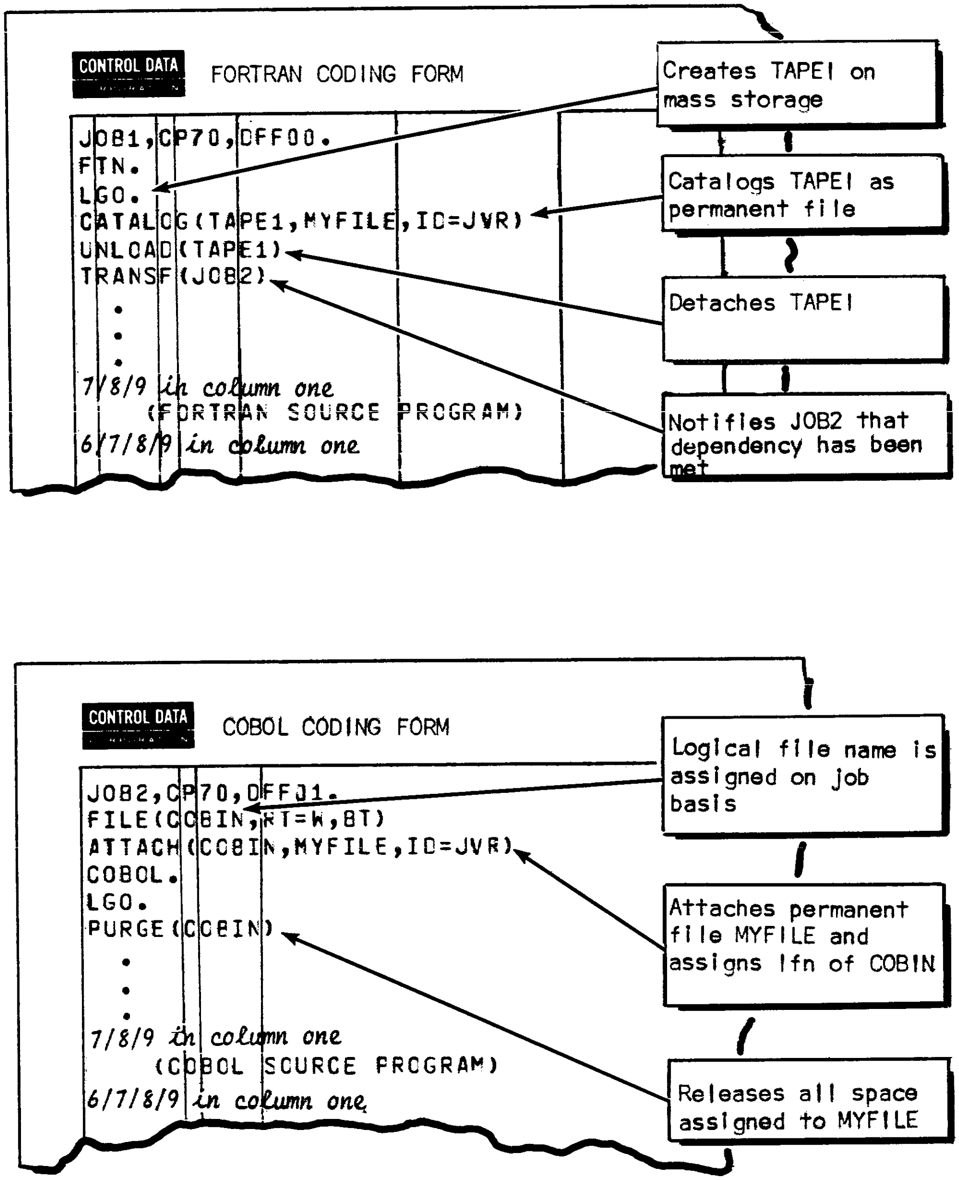
```
CATALOG(lfn, pfn, ID=id)
```

This allows any user to attach the file simply by using the corresponding ATTACH statement.

```
ATTACH(lfn, pfn, ID=id)
```

Because all permissions are implicit to this type of catalog, multiread access is not possible.

Example 8-1 illustrates two interdependent jobs that use the same file. TAPE1 is declared as a permanent file on mass storage so that it survives job termination. JOB1 then returns the file to the system and posts the dependency so that the file can be used by JOB2 which is waiting. JOB2 is then able to begin processing. It attaches the file, uses it, and purges it using the lfn. As will be shown, any job that attaches a cycle of a file with control permission can purge the cycle.



CONTROL DATA
FORTRAN CODING FORM

```
JOB1,CP70,DFF00.
FTN.
LGO.
CATALOG(TAPE1,MYFILE,ID=JVR)
UNLOAD(TAPE1)
TRANSF(JOB2)
 .
 .
 .
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
6/7/8/9 in column one
```

Creates TAPE1 on mass storage

Catalogs TAPE1 as permanent file

Detaches TAPE1

Notifies JOB2 that dependency has been met

CONTROL DATA
COBOL CODING FORM

```
JOB2,CP70,DFF01.
FILE(COBIN,RT=W,BT)
ATTACH(COBIN,MYFILE,ID=JVR)
COBOL.
LGO.
PURGE(COBIN)
 .
 .
 .
7/8/9 in column one
     (COBOL SOURCE PROGRAM)
6/7/8/9 in column one
```

Logical file name is assigned on job basis

Attaches permanent file MYFILE and assigns lfn of COBIN

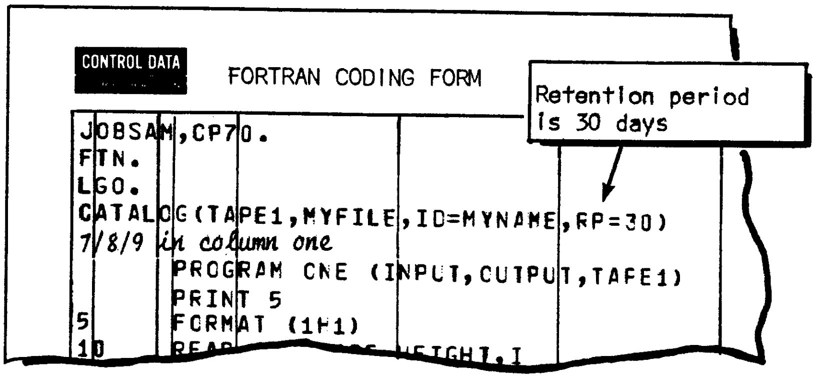
Releases all space assigned to MYFILE

Example 8-1. Permanent File With No Password Requirements

Using the Retention Parameter

As the creator of a cycle, you can specify the period for which the cycle is to be retained, in the form RP=n where n indicates 0 to 999 days. A retention period of 999 days is interpreted as an indefinite retention period. If no retention period is specified when you catalog a cycle, the default retention period is 1 day. Expiration of a retention period does not cause the cycle to be automatically purged. Usually, a systems analyst or operator obtains a list of expired files which are considered candidates for a periodic purging.

Example 8-2 illustrates a job that defines the retention period as 30 days for cycle one of permanent file MYFILE.



CONTROL DATA

FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
LGO.
CATALOG(TAPE1,MYFILE,ID=MYNAME,RP=30)
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAFE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ          WEIGHT,I
```

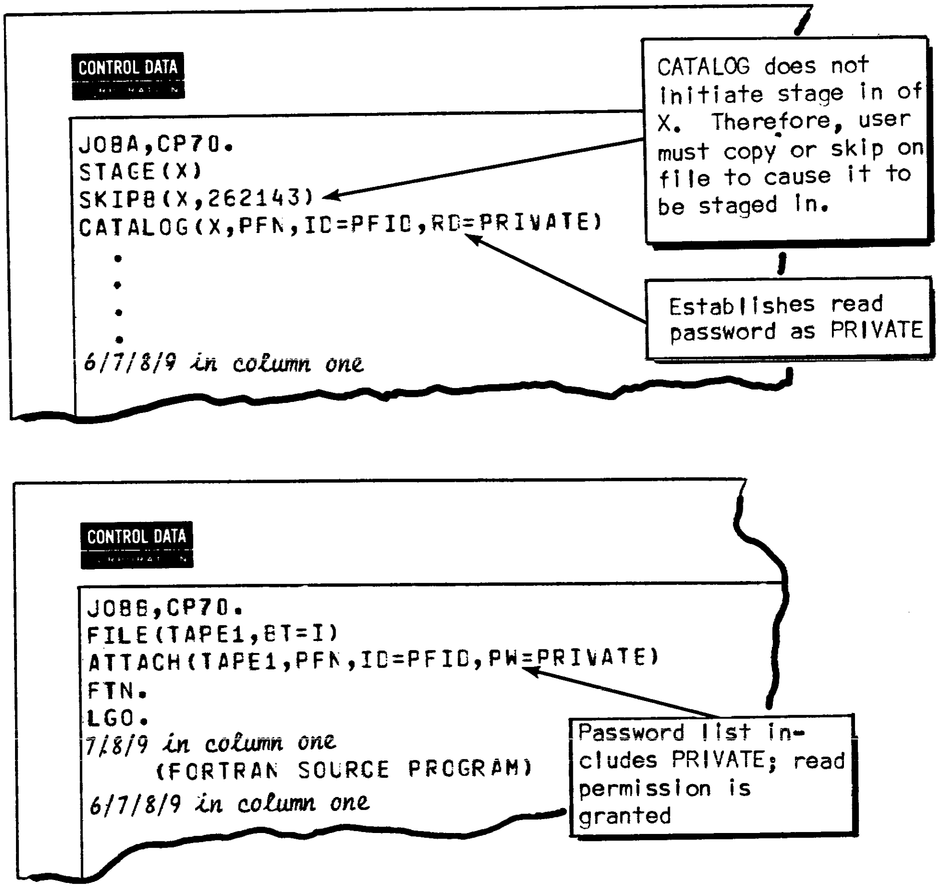
Retention period is 30 days

Example 8-2. Using the Retention Parameter

Using the Cycle Number Parameter

When cataloging or attaching a cycle of a permanent file, you have the option of assigning a cycle number in the form CY=n, where n is a decimal value from 1 to 63. This parameter is seldom specified when cataloging the initial cycle but becomes significant when cataloging a new cycle. The cycle number, if unspecified on a catalog defaults to 1. If cycle n already exists, another cycle n is not allowed. The cycle number if unspecified on an ATTACH defaults to the highest numbered cycle in the permanent file.

In Example 8-3, file X of JOBA contains information of a confidential nature so the creator wishes to restrict reading to those whom he has given the read password. JOBB attaches the file with read permission. It is accessed as TAPE1 by the FORTRAN object program.



```
JOBA,CP70.
STAGE(X)
SKIPB(X,262143)
CATALOG(X,PFN,ID=PFID,RD=PRIVATE)
  •
  •
  •
  •
6/7/8/9 in column one
```

CATALOG does not initiate stage in of X. Therefore, user must copy or skip on file to cause it to be staged in.

Establishes read password as PRIVATE

```
JOBB,CP70.
FILE(TAPE1,BT=I)
ATTACH(TAPE1,PFN,ID=PFID,PW=PRIVATE)
FTN.
LGO.
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
6/7/8/9 in column one
```

Password list includes PRIVATE; read permission is granted

Example 8-3. Defining the Read Password

Using the Extend Password

The extend password (EX=expw, where expw is 1 to 9 characters), if specified on the catalog of the initial cycle of a permanent file, permits any user who knows the password to increase or decrease the size of any cycle of a permanent file.

The cycle number is unrelated to the number of cycles currently in the permanent file.

Catalog With Passwords

Passwords specified for read, modify, extend, or control operations restrict the exercise of that permission to those who know the password. If any password is undefined on an initial catalog, its corresponding permission is automatically granted on any attach of the file. Passwords are further protected through the practice of censoring their listing in the dayfile for the job.

When cataloging additional cycles of a permanent file, the passwords for later cycles are the same as for those established when the first cycle was cataloged. You cannot establish new passwords when cataloging additional cycles.

NOTE

Multiread access is possible only when the EX, MD, and CN passwords are established on the initial catalog. Specification of these three passwords is recommended.

The CATALOG statement with password definitions has the following format when used to create the initial cycle of a permanent file. The password parameters can be omitted or can be in any order after the lfn and pfn parameters. The same character string can be used for more than one password.

```
CATALOG(lfn,pfn,ID=id,...,RD=rdpw,EX=expw,MD=mdpw,CN=dnpw,TK=tkpw)
```

Using the Read Password

The read password (RD=rdpw, where rdpw is 1 to 9 characters), if specified on the initial cycle of a permanent file, permits only those users who know the password to read any portion of any cycle of the permanent file. That is, any user attaching the file for the purpose of reading it must include the read password on the ATTACH statement, as follows.

```
ATTACH(lfn,pfn,ID=id,...,PW=rdpw)
```

The extend permission does not imply read permission; that is, if you specify the extend password, you must also specify the read password to be able to read the file. Without extend permission, you can write beyond the EOI for your file, but the information does not become a permanent part of the file. When your job terminates, the information beyond the original EOI is lost. When the extend password has been defined, any user attaching any cycle of the file for the purpose of extending or altering the size of the file must include the extend password on the ATTACH statement, as follows.

```
ATTACH(lfn,pfn,ID=id,...,PW=expw)
```

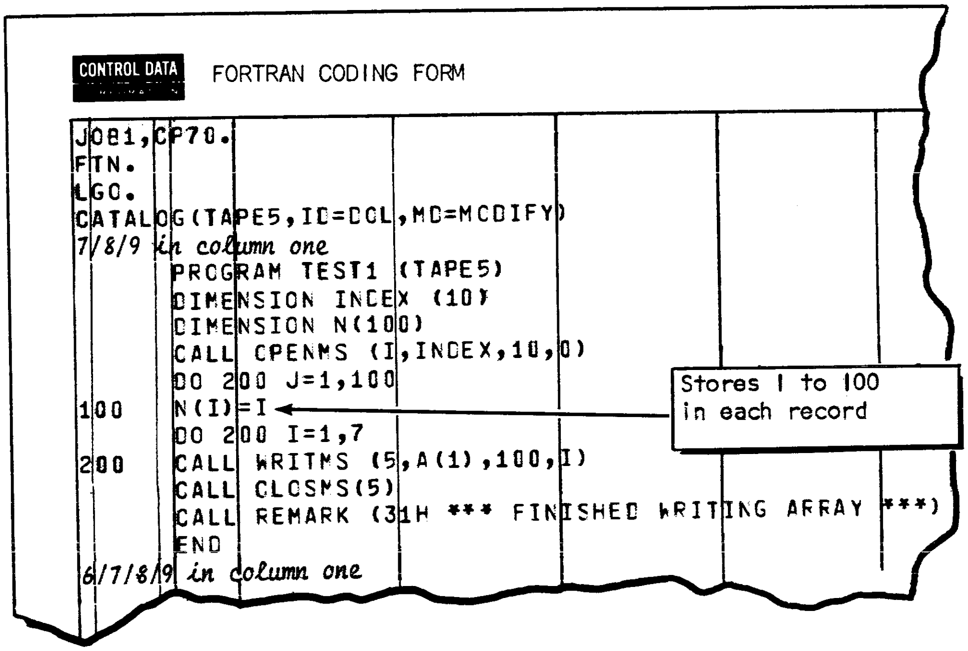
When a user attaches a file and obtains extend permission, he obtains exclusive access to that cycle of the permanent file.

Using the Modify Password

The modify password (MD=mdpw, where mdpw is 1 to 9 characters), if specified on the catalog of the initial cycle of a permanent file, permits any user who knows the password to rewrite any portion of the existing data on any cycle of a permanent file. The modify permission does not imply read permission; that is, if you specify the modify password, you must also specify the read password to read the file. When the modify password has been defined, any user attaching any cycle of the file for the purpose of modifying data up to the EOI must include the modify password on the ATTACH statement, as follows.

```
ATTACH(lfn,pfn,ID=id,...,PW=mdpw)
```

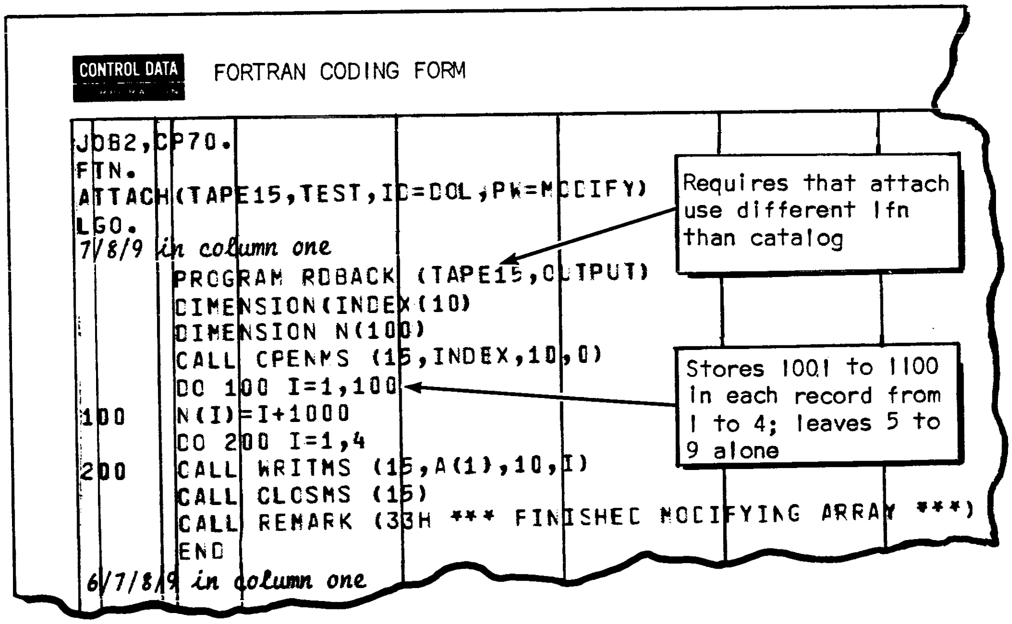
When a user attaches a file with modify permission, he obtains exclusive access to that cycle of the permanent file. In Example 8-4, JOB1 creates a word addressable file. JOB2 changes some of the records on the file. The modify permission when used in conjunction with the extend permission also permits you to reduce the size of your file.



CONTROL DATA FORTRAN CODING FORM

```
JOB1,CP70.
FTN.
LGO.
CATALOG(TAPE5,ID=DCL,MD=MCDIFY)
7/8/9 in column one
      PROGRAM TEST1 (TAPE5)
      DIMENSION INCEX (10)
      DIMENSION N(100)
      CALL CPENMS (I,INCEX,10,0)
      DO 200 J=1,100
100   N(I)=I
      DO 200 I=1,7
200   CALL WRITMS (5,A(1),100,I)
      CALL CLCSMS(5)
      CALL REMARK (31H *** FINISHED WRITING ARRAY ***)
      END
6/7/8/9 in column one
```

Stores 1 to 100
in each record



CONTROL DATA FORTRAN CODING FORM

```
JOB2,CP70.
FTN.
ATTACH(TAPE15,TEST,ID=DOL,PW=MCDIFY)
LGO.
7/8/9 in column one
      PROGRAM RDBACK (TAPE15,OUTPUT)
      DIMENSION(INDEX(10)
      DIMENSION N(100)
      CALL CPENMS (15,INDEX,10,0)
      DO 100 I=1,100
100   N(I)=I+1000
      DO 200 I=1,4
200   CALL WRITMS (15,A(1),10,I)
      CALL CLCSMS (15)
      CALL REMARK (33H *** FINISHEC MODIFYING ARRAY ***)
      END
6/7/8/9 in column one
```

Requires that attach use different lfn than catalog

Stores 1001 to 1100 in each record from 1 to 4; leaves 5 to 9 alone

Example 8-4. Modifying a Permanent File

Using the Control Password

The control password (CN=cnpw, where cnpw is 1 to 9 characters), if specified on the catalog of the initial cycle of a permanent file, permits the user who knows the password to catalog a new cycle or attach a cycle and purge it from the permanent file. The control permission implies no other permission.

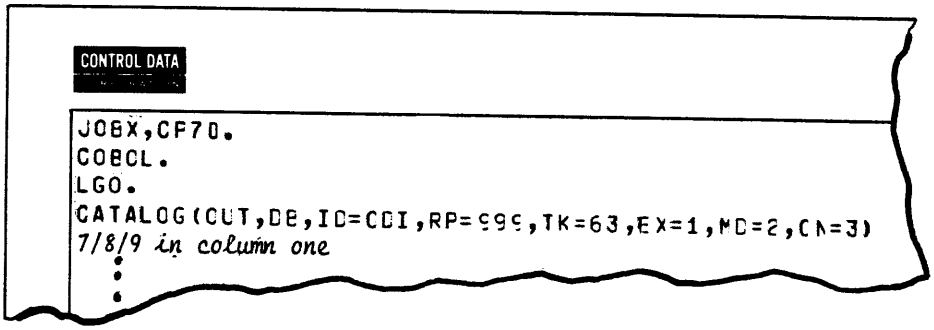
Using the Turnkey Password

The turnkey password (TK=tkpw, where tkpw is 1 to 9 characters) restricts no specific permissions; however, if a turnkey password is defined on the catalog of the original cycle of a permanent file, only the user who knows the password can exercise permissions specified or defaulted for any cycle of the permanent file.

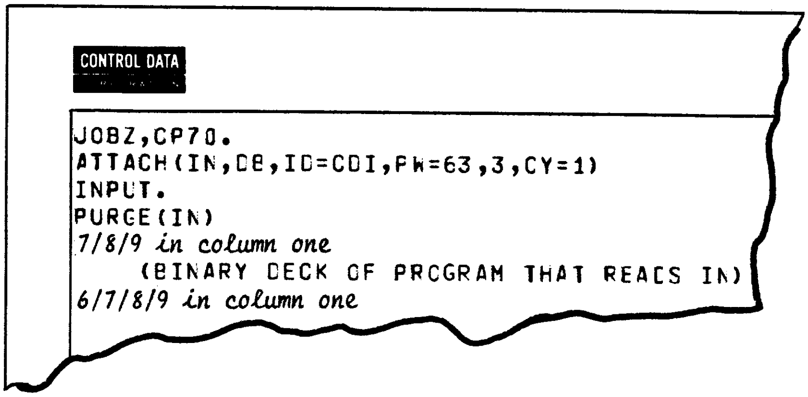
Thus, the turnkey password must be specified in addition to any other passwords when you wish to attach a file or catalog additional cycles of a file.

```
ATTACH(lfn, pfn, ID=id, . . . , PW=tkpw)
```

Example 8-5 shows how the turnkey password is used. JOBX shows the initial catalog. JOBZ shows attaching of a later cycle with control permission. In this case, the turnkey password also grants read permission since no read password is defined on the initial catalog.



```
JOBX,CF70.
COBOL.
LGO.
CATALOG(OUT,DB,ID=CDI,RP=999,TK=63,EX=1,MD=2,CN=3)
7/8/9 in column one
   •
   •
   •
```



```
JOBZ,CP70.
ATTACH(IN,DB,ID=CDI,PW=63,3,CY=1)
INPUT.
PURGE(IN)
7/8/9 in column one
     (BINARY DECK OF PROGRAM THAT READS IN)
6/7/8/9 in column one
```

Example 8-5. Using the Turnkey Password

Password List

On the initial catalog of the first cycle of a permanent file, each password is listed separately because this is how they are established. Once established, these passwords are referred to in password lists on subsequent catalogs of new cycles and on attach requests for existing cycles.

```
ATTACH(lfn, pfn, ID=id, . . . , PW=password1, password2, . . . , passwordn)
    or
CATALOG
```

A password list is ignored if it is used on an initial catalog statement. Similarly, any attempt to establish or define a password (for example by using MD=mdpw) on a catalog of a new cycle is ignored. The password list can be in any order after the lfn and pfn parameters. Suppose that the initial catalog uses the following CATALOG statement.

```
CATALOG(ALPHA, MF, ID=ME, MD=MOD, EX=EXTEND, CN=CONTRL, TK=TURNKEY)
```

The only permission for which no password is defined is read permission. Thus, the following password lists grant permissions as indicated.

Password List	Permission Granted
PW=TURNKEY	Read
PW=TURNKEY, EXTEND	Read and extend
PW=TURNKEY, EXTEND, MOD	Read, extend, and modify
PW=TURNKEY,CONTRL	Read and control (catalog and purge)
PW=EXTEND, MOD	None; turnkey password is missing
PW=TURNKEY, MOD	Read and modify

Note that when a turnkey is defined, it must be supplied on any attach. Failure to supply the turnkey password would mean that no permissions would have been granted on the attach, a condition that causes job termination.

All Passwords the Same

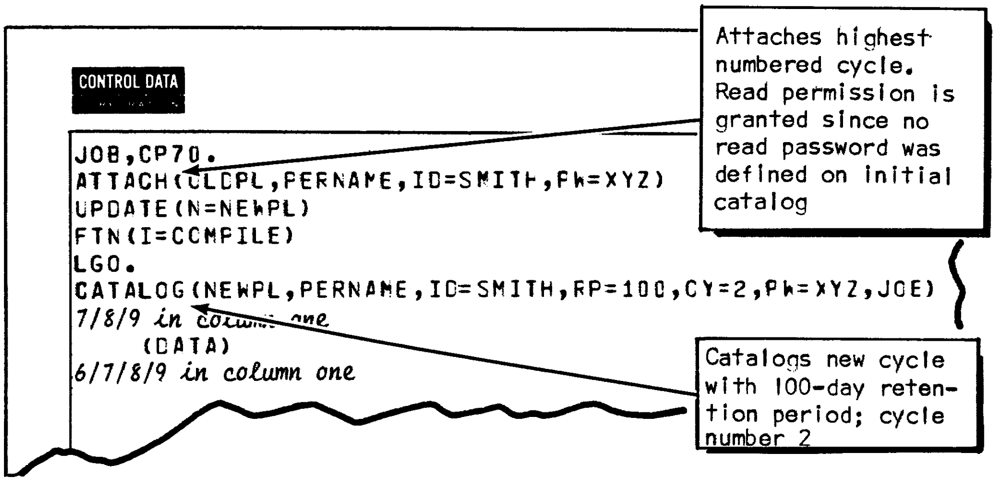
If you catalog an initial cycle of a file with one or more of the passwords having identical character strings (for example, MD=XYZ and EX=XYZ), you can enable all of the passwords having identical character strings by supplying the string once in the password list. For example, PW=XYZ grants both modify and extend permissions.

CATALOGING SUBSEQUENT CYCLES

Creation of a subsequent cycle of an existing permanent file differs from the creation of the original cycle in that control permission and possibly the turnkey permission must be obtained. If neither permission was defined on the original catalog, no password list is required to obtain permissions. Use the following form of CATALOG when adding a new cycle to an existing permanent file.

```
CATALOG(lfn, pfn, ID=id, RP=rp, CY=n, PW=password1, password2)
```

The parameters for lfn, pfn, RP, CY, and ID have the same meanings as in the previous descriptions. The default for the cycle number is still 1; you will want to specify a cycle number on a new cycle catalog. Example 8-6 shows a job adding a new cycle to a permanent file. The initial cycle of the permanent file for which the pfn is PERNAME was cataloged with TK=XYZ and CN=JOE. The creator ID is SMITH.



```
JOB,CP70.
ATTACH(OLDPL,PERNAME,ID=SMITH,PW=XYZ)
UPDATE(N=NEWPL)
FTN(I=COMPILE)
LGO.
CATALOG(NEWPL,PERNAME,ID=SMITH,RP=100,CY=2,PW=XYZ,JOE)
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

Attaches highest numbered cycle. Read permission is granted since no read password was defined on initial catalog

Catalogs new cycle with 100-day retention period; cycle number 2

Example 8-6. Cataloging a New Cycle

ALTERING THE SIZE OF PERMANENT FILES

As previously noted, you can rewrite information in a cycle of a permanent file if you have modify permission. No permanent file statements other than the ATTACH statement are required.

If you want to change the size of a permanent file, however, you can do so only by using either the ALTER or the EXTEND control statements. The ALTER statement allows you to lengthen or shorten a cycle of a permanent file. You must have acquired both modify and extend permission when you attached a cycle of the file to be shortened. Only extend permission is required to lengthen a file.

The ALTER control statement has the following format.

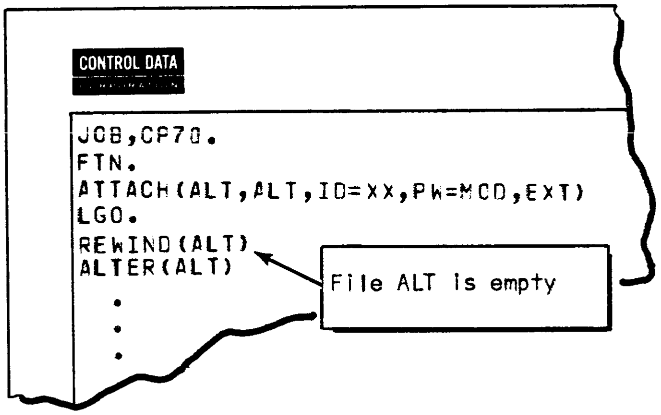
```
ALTER(lfn)
```

The EXTEND statement allows you to lengthen the cycle but not shorten it and requires only extend permission. The control statement has the following format.

```
EXTEND(lfn)
```

Because the attached file is used as a logical file, the only parameter required following the ALTER or EXTEND statement is the logical file name used when the file was attached. If a file is to be extended only, it must be positioned at end of information before any writing occurs. If it is a blocked file, it must be closed.

Example 8-7 illustrates how ALTER can be used to create an empty cycle of a permanent file.

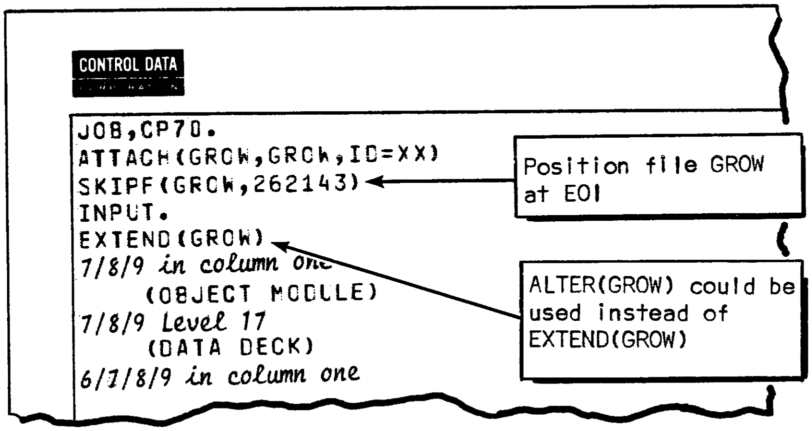


```
JOB,CP70.
FTN.
ATTACH(ALT,ALT,ID=XX,PW=MCD,EXT)
LGO.
REWIND(ALT)
ALTER(ALT)
   .
   .
   .
```

File ALT is empty

Example 8-7. Using the ALTER Control Statement

In Example 8-8 the permanent file GROW was initially cataloged with no password requirements. Hence, none need be specified to extend any cycle of the file. In the example, INPUT causes the program that writes on GROW to be loaded and executed. The EXTEND makes the addition permanent.



```
JOB,CP70.
ATTACH(GROW,GROW,ID=XX)
SKIPF(GROW,262143)
INPUT.
EXTEND(GROW)
7/8/9 in column one
     (OBJECT MODULE)
7/8/9 Level 17
     (DATA DECK)
6/7/8/9 in column one
```

Position file GROW at EOI

ALTER(GROW) could be used instead of EXTEND(GROW)

Example 8-8. Using the EXTEND Control Statement

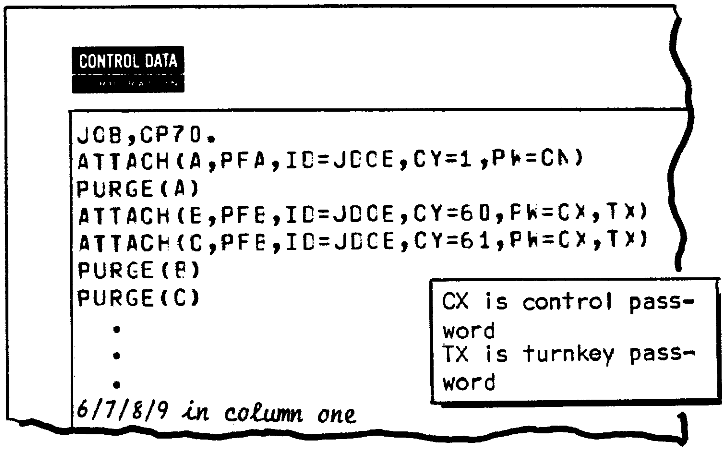
PURGING PERMANENT FILES

If you have attached a cycle of a permanent file with control permission, you have the authority to purge that cycle from the permanent file. The control statement has the following format.

```
PURGE(lfn)
```

When the job terminates, or if you issue a RETURN or UNLOAD of file lfn, the mass storage used by the file is returned to the system. No subsequent job can attach the file.

In Example 8-9, a user with the ID of JDOE decides to clean out three obsolete cycles from his permanent files to make room for new files.



CONTROL DATA

```
JOB,CP70.
ATTACH(A,PFA,ID=JDOE,CY=1,PW=CN)
PURGE(A)
ATTACH(B,PFB,ID=JDOE,CY=60,PW=CX,TX)
ATTACH(C,PFB,ID=JDOE,CY=61,PW=CX,TX)
PURGE(B)
PURGE(C)
   •
   •
   •
6/7/8/9 in column one
```

CX is control password
TX is turnkey password

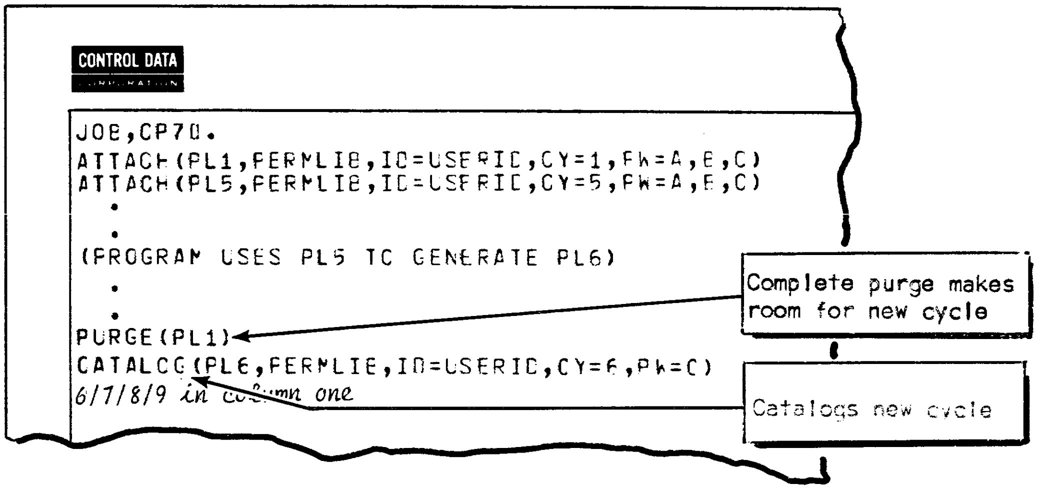
Example 8-9. Purging a Cycle of a Permanent File

The status of the purged file for the remainder of the job is determined by whether the purge is partial or complete. A partial purge requires control permission as a minimum and all but one permission as a maximum. After the PURGE statement is processed, file lfn still has all of the characteristics of a permanent file until the file is returned or the job terminates. Thus, for example, if you do not have read permission you cannot read the file even though it has been purged. This is because the pfn and all the permissions are still associated with the lfn.

Example 8-9 illustrates a partial purge because only control permission has been granted.

In contrast, a complete purge takes place when all permissions are granted. In this case, after the PURGE statement is processed, file lfn does not have the characteristics of a permanent file but instead resembles a temporary file. The lfn is no longer associated with the pfn. This means that if your permanent file contains five cycles, a complete purge can be immediately followed by a new cycle catalog in the same job. This would not be possible using a partial purge since with a partial purge the permanent file would appear to have all five cycles until the job ended or the file was returned or unloaded.

Example 8-10 shows a job that performs a complete purge of a cycle of a permanent file to make room for a new cycle.



```
JOB,CP70.
ATTACH(PL1,PERMLIB,ID=USERID,CY=1,PW=A,B,C)
ATTACH(PL5,PERMLIB,ID=USERID,CY=5,PW=A,B,C)
  •
  •
(PROGRAM USES PL5 TO GENERATE PL6)
  •
  •
PURGE(PL1)
CATALOG(PL6,PERMLIB,ID=USERID,CY=6,PW=C)
6/7/8/9 in column one
```

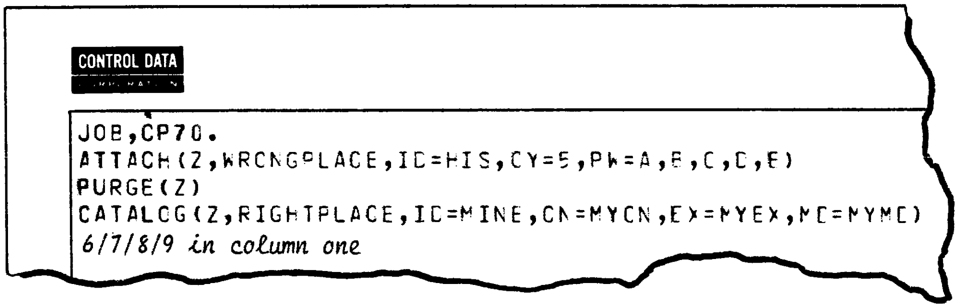
Complete purge makes room for new cycle

Catalogs new cycle

Example 8-10. Purging a Cycle and Adding a New Cycle

Another advantage of a complete purge is that it allows a user to recover a permanent file during EXIT processing if an abnormal termination occurs.

A complete purge also allows you to attach a cycle of a file, and without copying it, remove it from its current location under one file name and recatalog it under a different file name. In Example 8-11, the user moves a permanent file from a permanent file named WRONGPLACE and recatalogs it under RIGHTPLACE. Since this represents the initial catalog of a cycle under RIGHTPLACE, the CATALOG contains password definitions.



```
JOB,CP70.
ATTACH(Z,WRONGPLACE,ID=HIS,CY=5,PW=A,B,C,D,E)
PURGE(Z)
CATALOG(Z,RIGHTPLACE,ID=MINE,CN=MYCN,EX=MYEX,MD=MYMD)
6/7/8/9 in column one
```

Example 8-11. Moving a Cycle From One Permanent File to Another

INSTALLATION DEFINED PRIVACY PROCEDURES

Your site may define its own privacy procedures to be used instead of or in addition to the permissions described in this manual. Check with your systems analyst to see if your site has defined additional requirements for accessing permanent files.

USING SCOPE 3.x PERMANENT FILES

So far, you have learned how to use permanent files on system mass storage at the CYBER 70/Model 76 or 7600 Computer System. Suppose your site has a CYBER 70 Series Station or a 6000 Series Station, and you know of a permanent file there that you would like to access or perhaps you would like to maintain a permanent file of your own at the 6000 Station. SCOPE 2 allows both of these operations. You cannot, however, modify, extend, or purge a file at the 6000 Station. These restrictions arise from the fact that when you attach a cycle of a 6000 Station permanent file, you do not obtain direct access to the file on the 6000. Instead, the first time your attached file is opened, the 6000 Station sends a working copy of the cycle requested over to 7600 mass storage where it resembles a temporary file more than it does a permanent file. Similarly, a catalog of a 6000 Station permanent file results in a copy of your file on 7600 mass storage being sent over to the 6000 Station as soon as the file is unloaded.

Before attempting to use a permanent file at the 6000 Station, read the section on permanent files in the SCOPE 3.3 or SCOPE 3.4 Reference Manual, depending on the operating system in use at the 6000 Station.

Restrictions on use of SCOPE 3.x permanent files are:

1. You cannot dispose or stage a file either attached from the 6000 Station or cataloged at the 6000 Station.

2. Under certain conditions, the user must assure that the transfer of an attached file has taken place before his job accesses the file.

3. The SCOPE 2 file organization library (LB) cannot be used under SCOPE 3.4. You can catalog files having this organization at the 6000 Station but they cannot be accessed by a SCOPE 3.4 job. None of the SCOPE 2 file organizations is available under SCOPE 3.3.

4. SCOPE 3.4 requires blocking for sequential (SQ) files.

5. SCOPE 3.4 permanent files using SIS, SDA, and IORANDM are not interchangeable.

6. If you do not supply a FILE statement for the attached SCOPE 3.x permanent file, it is assumed by default to be record type W, unblocked. Remember that the default file type for a file created under SCOPE 3.x is S record type, C blocked. S records are the only file type interchangeable with SCOPE 3.3.

7. Files to be cataloged at the 6000 Station from the 7600 must be unlabeled.

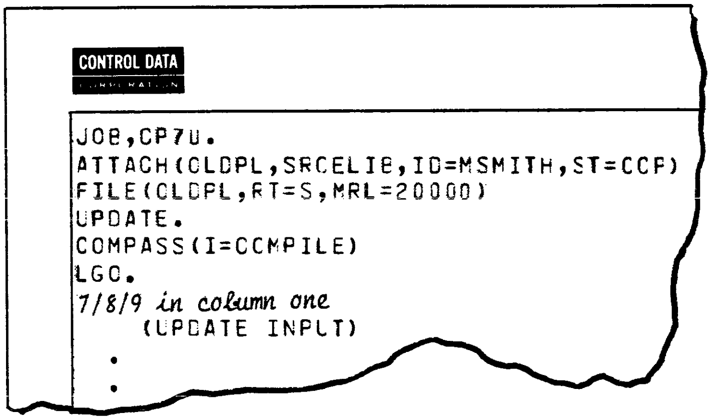
8. Although exceptions exist, relocatable binary files are, in general, not interchangeable.

USING THE STATION PARAMETER

Addition of the station parameter (ST=ggg, where ggg is the station identifier) to the ATTACH statement notifies SCOPE 2 that the file to be attached is at the 6000 Station. Adding ST=ggg to CATALOG causes the file to be cataloged at the 6000 Station. You can catalog initial cycles or new cycles of existing files. When the ST parameter is present, parameters generally are subject to the requirements of the SCOPE 3.x permanent file manager. One exception is that SCOPE 2 requires that the ATTACH or CATALOG specify an lfn parameter even though pfn is specified (a feature of SCOPE 3.4 permits lfn or pfn to be omitted) and that if pfn is omitted that the positional comma be present. Another exception is that if the cycle number is specified on a catalog of a new cycle or the attach of a cycle, that it be in the range 1 through 64 even though SCOPE 3.4 now allows cycle numbers to go as high as 999. To avoid this restriction, it is best to always omit the cycle number; SCOPE 3.4 default for an omitted cycle number on a catalog of a new cycle provides the highest numbered cycle plus 1.

When the ST parameter is used on a CATALOG or ATTACH statement that is continued on additional cards, be sure that the parameter is on the first card of the statement. SCOPE 2 searches only the first 80 characters of the statement for the parameter.

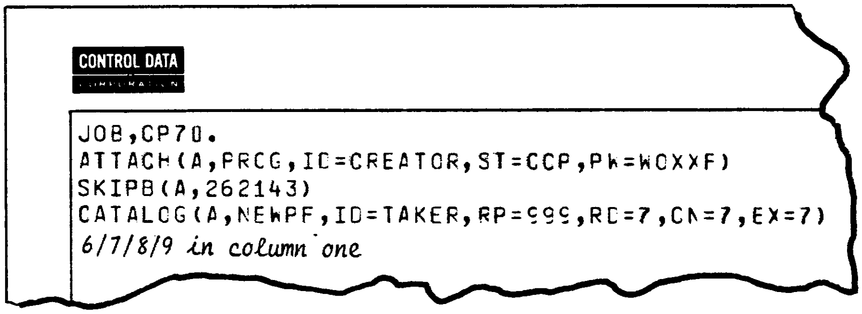
Example 8-12, shows how to acquire an UPDATE old program library that exists as a permanent file at the 6000 Station for assembly and execution on the 7600. The old program library is S record format, requiring a FILE statement. No read password is defined for SRCELIB, so anyone is free to read the file.



```
JOB,CP70.
ATTACH(OLDPL,SRCELIB,ID=MSMITH,ST=CCF)
FILE(OLDPL,RT=S,MRL=20000)
UPDATE.
COMPASS(I=COMPILE)
LGO.
7/8/9 in column one
    (UPDATE INPUT)
   •
   •
```

Example 8-12. Attaching a 6000/CYBER 70 Station Permanent File

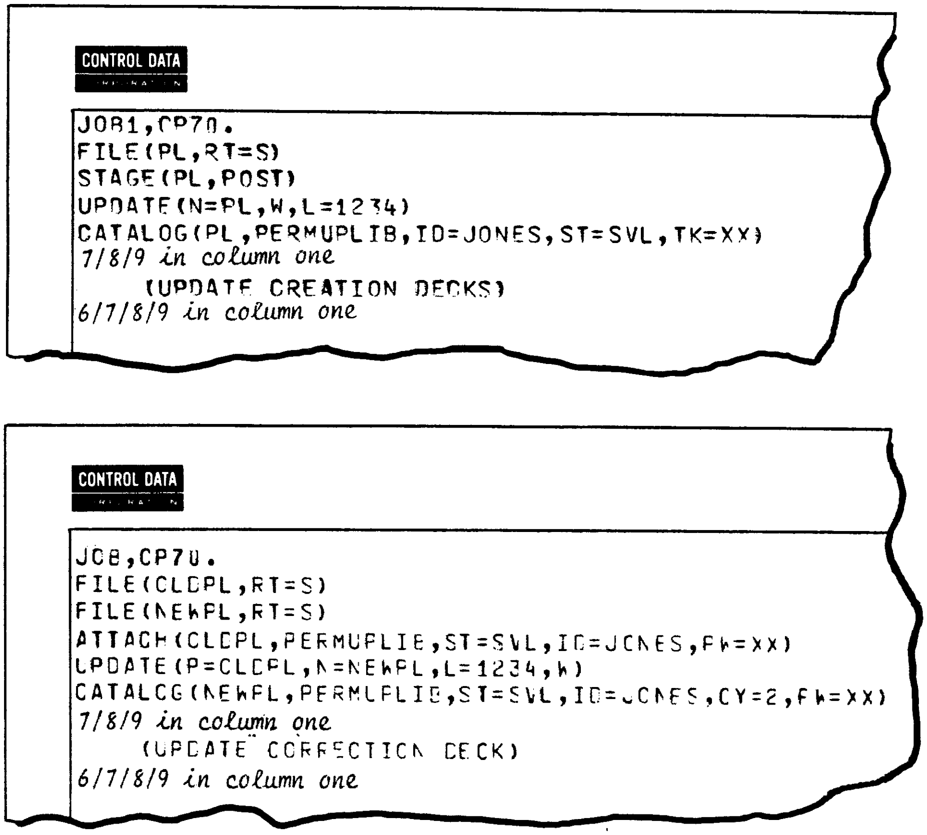
In Example 8-13, a user wishes to obtain a copy of a cycle of a 6000 permanent file and recatalog it at the 7600 so that the information exists under both permanent file systems. In this case, the user must initiate the transfer of the file from the 6000 Station through a copy or skip operation since the CATALOG statement does not open the file. The transfer to the 7600 occurs when the user first opens the file.



```
JOB,CP70.
ATTACH(A,PROG,ID=CREATOR,ST=CCP,PW=WOXXF)
SKIPB(A,262143)
CATALOG(A,NEWPF,ID=TAKER,RP=999,RD=7,CN=7,EX=7)
6/7/8/9 in column one
```

Example 8-13. Making a 6000/CYBER 70 Permanent File into a 7600 Permanent File

Example 8-14 shows an UPDATE old program library being created under SCOPE 2 and cataloged under SCOPE 3.4 at the 6000 Station (identified as SVL). This cycle is then attached using a job that can be processed under either system. In this case, it is important to supply a cycle number since the default for cycle number on a new catalog differs between systems. The ST parameter is ignored by SCOPE 3.4.



```
JOB1,CP70.
FILE(PL,RT=S)
STAGE(PL,POST)
UPDATE(N=PL,W,L=1234)
CATALOG(PL,PERMUPLIB,ID=JONES,ST=SVL,TK=XX)
7/8/9 in column one
    (UPDATE CREATION DECKS)
6/7/8/9 in column one
```

```
JOB,CP70.
FILE(OLDPL,RT=S)
FILE(NEWPL,RT=S)
ATTACH(OLDPL,PERMUPLIB,ST=SVL,ID=JONES,PW=XX)
UPDATE(P=OLDPL,N=NEWPL,L=1234,W)
CATALOG(NEWPL,PERMUPLIB,ST=SVL,ID=JONES,CY=2,PW=XX)
7/8/9 in column one
    (UPDATE CORRECTION DECK)
6/7/8/9 in column one
```

Example 8-14. Maintaining UPDATE OLDPL at 6000/CYBER 70 Station

UNIT RECORD DEVICES AT STATIONS 9

All unit record devices are at stations and are not directly accessible to the CPU. In particular, this includes all card readers, printers, and card punches. Tape units are described in Section 6.

When you originate data at a unit record device, the station passes it to SCOPE 2 as a mass storage file which is accessible to your program. Thus, when your program reads a card from your job deck, it actually reads the image from a mass storage file.

When you want data to be put out on a unit record device (that is, printed or punched), your program places the data on a mass storage file which is then passed to the station for disposition upon request or when the job terminates. This means that when a program "punches" a card, it actually writes the card image on mass storage. When the job terminates, the station punches the card.

This section familiarizes you with the programmer's role in such offline transfers of data, referred to as "spooling."

CARD READER INPUT

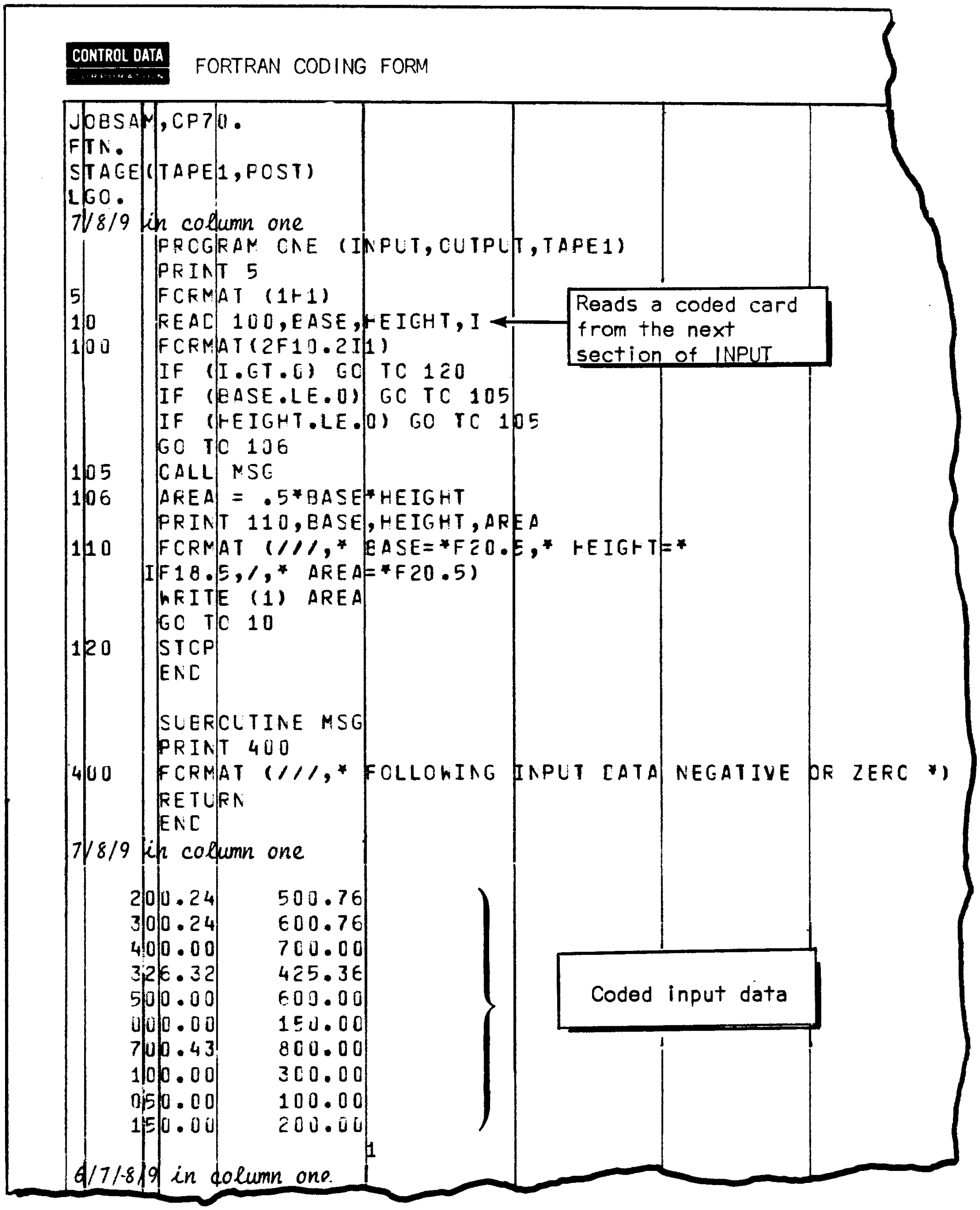
The INPUT file contains the portion of your job deck that most concerns you. Each section or partition in your job deck becomes a section or partition of INPUT. A section consists of sequential, unblocked W records that vary in size according to whether your deck section is coded cards, formatted binary cards, or free form binary cards.

SCOPE 2 normally terminates the INPUT file with an EOS/EOP/EOI sequence. If the last card in the job deck is an EOP, SCOPE adds a second EOP before the EOI. In this card there is no EOS preceding the EOP.

The FORTRAN language READ fn,iolist statement automatically refers to the next record on INPUT (see Example 9-1). Any other file can be equated to INPUT by adding the parameter TAPExx=INPUT to the program statement. In this case, the READ or BUFFER IN statement that references the unit as xx will actually read from INPUT. It is also possible to equate TAPExx=INPUT to enable the INPUT file to be referenced by a unit number. This allows you to check for EOF on INPUT, since the IF(EOF) statement requires a unit number.

You can cause your COBOL program to read from INPUT through use of the COBOL ASSIGN clause.

The INPUT file cannot be returned or unloaded. Although it is not protected from being redefined as some other file organization or record type, you should avoid redefining it. You can also change the MRL between job steps.



CONTROL DATA

FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,POST)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
      IF (HEIGHT.LE.0) GO TO 105
      GO TO 106
105   CALL MSG
106   AREA = .5*BASE*HEIGHT
      PRINT 110,BASE,HEIGHT,AREA
110   FORMAT (///,* BASE=*F20.5,* HEIGHT=*
     IF18.5,/,* AREA=*F20.5)
      WRITE (1) AREA
      GO TO 10
120   STOP
      END

      SUBROUTINE MSG
      PRINT 400
400   FORMAT (///,* FOLLOWING INPUT DATA NEGATIVE OR ZERO *)
      RETURN
      END
7/8/9 in column one
    200.24    500.76
    300.24    600.76
    400.00    700.00
    326.32    425.36
    500.00    600.00
    000.00    150.00
    700.43    800.00
    100.00    300.00
    050.00    100.00
    150.00    200.00
                    1
6/7/8/9 in column one
```

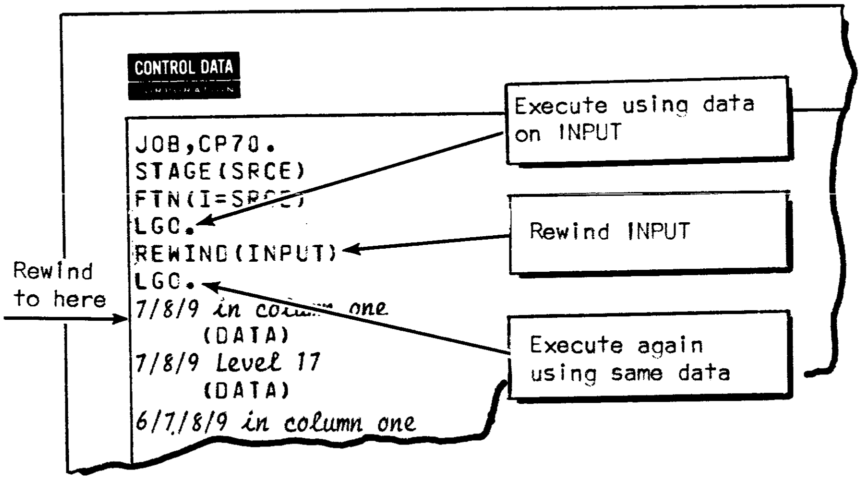
Reads a coded card from the next section of INPUT

Coded input data

Example 9-1. Reading Cards from INPUT

Remember, that unlike 6000 SCOPE, the INPUT file does not contain control statements. Thus, a rewind or skip to BOI positions your file at the second section in your job deck. Example 9-2 illustrates this feature.

Unlike other files, the loader does not rewind INPUT before loading from it.



```
JOB,CP70.
STAGE(SRCE)
FTN(I=SRCE)
LGO.
REWIND(INPUT)
LGO.
7/8/9 in column one
     (DATA)
7/8/9 Level 17
     (DATA)
6/7/8/9 in column one
```

Example 9-2. Rewinding INPUT

CODED PUNCHED CARD INPUT

You are probably familiar with 80-column punched cards. However, not all punched cards use the same punch conventions. The two sets supported by SCOPE 2 are the 026 (Hollerith) set and the 029 (ASCII)† set. Look at Appendix A; you will see that for letters and numbers the punches are the same. It is in special characters that the sets differ. Thus, depending on the punch used for keypunching your cards, you will want to signal SCOPE that the card deck is 026 coded or 029 coded.

Unless you indicate otherwise, the system assumes that cards are coded using 026 punch format.††

†ANSI Standard x3.4-1968

††As an installation option, the default can be changed to 029 at the 6000/CYBER 70 Station and the 7611-11 Station. The relationship of 026 and 029 is then reversed.

Hollerith (026) Punched Card Input

Usually, your job identification statement, control statements, source language program, directives, and data are punched using the Hollerith (026) character set (Figure 9-1).

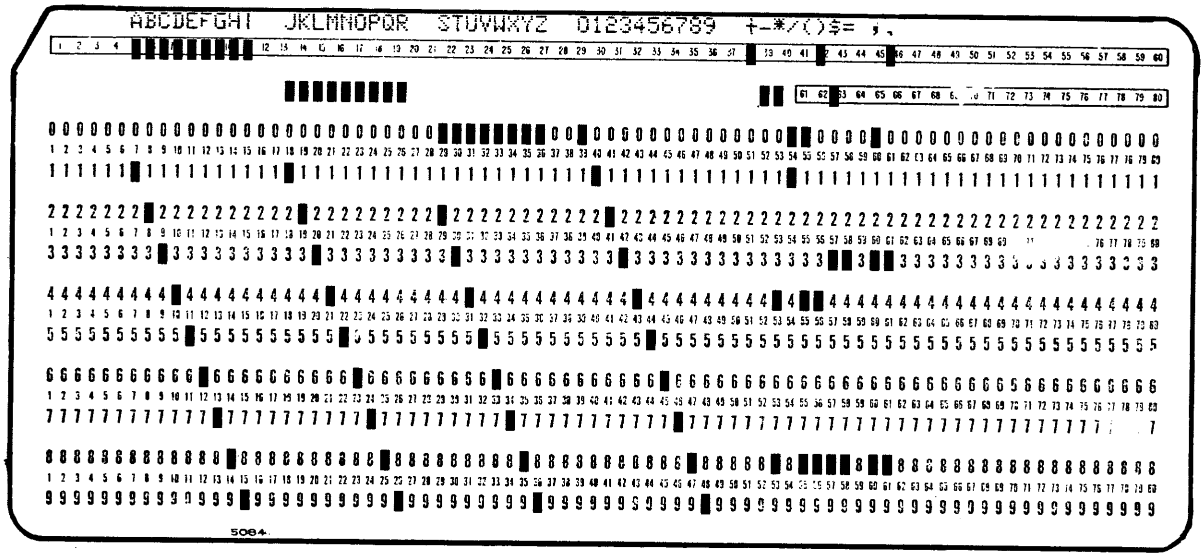


Figure 9-1. Hollerith (026) Coded Card

ASCII (029) Punched Card Input

The control statements, source language program, and data can be punched using the 029 character set (Figure 9-2). The set is limited to the 64 characters given in Appendix A.

To signal that 029 punched cards follow, punch the characters 29 in columns 79 and 80 of the job identification statement, EOS, or EOP card preceding the data. The system assumes all coded information following is in 029 punched format until it encounters an EOS or EOP that contains a 26 punch in columns 79 and 80. The 029 mode terminates upon encountering the EOI for the job.

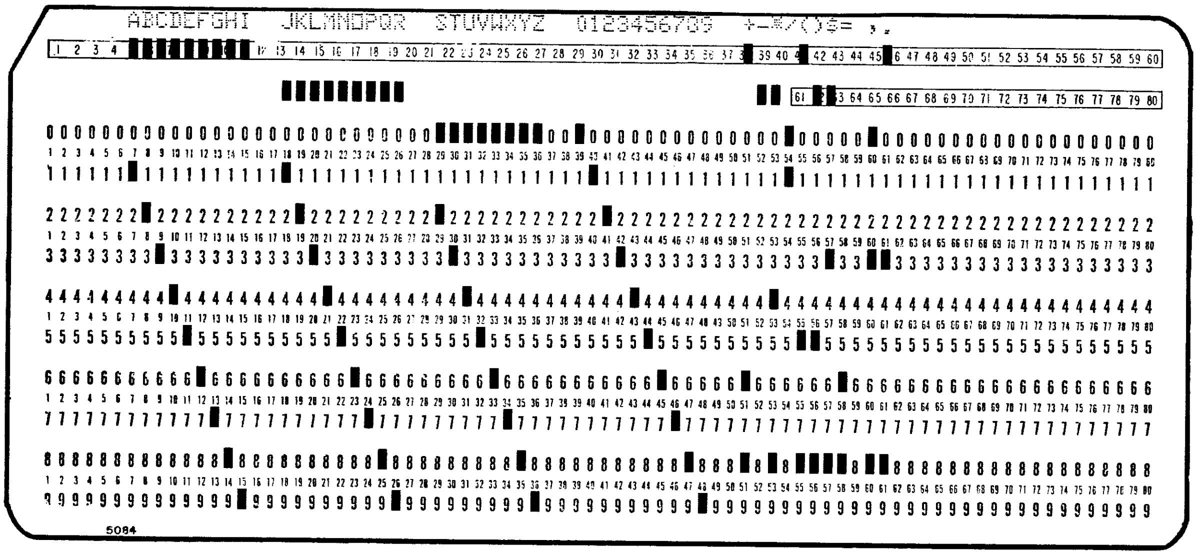
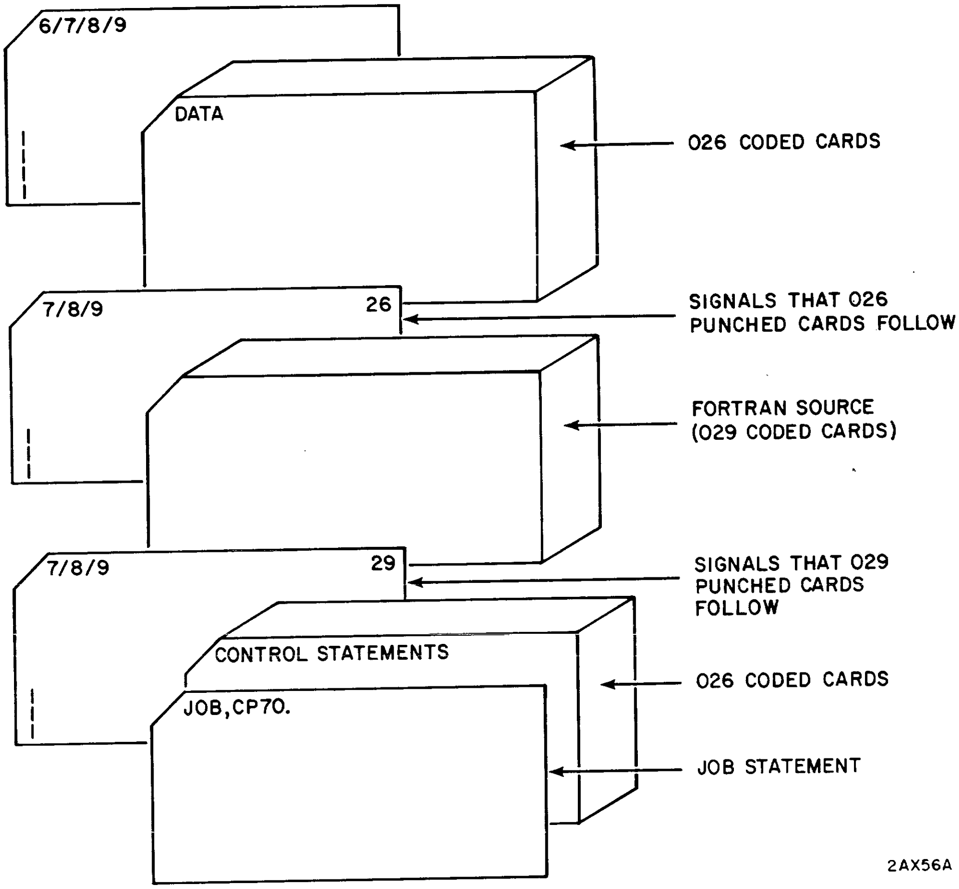


Figure 9-2. ASCII (029) Coded Card

Example 9-3 illustrates a job containing a FORTRAN source deck punched in the 029 punch format.


6/7/8/9
DATA
026 CODED CARDS
7/8/9
26
SIGNALS THAT 026 PUNCHED CARDS FOLLOW
FORTRAN SOURCE (029 CODED CARDS)
7/8/9
29
SIGNALS THAT 029 PUNCHED CARDS FOLLOW
CONTROL STATEMENTS
026 CODED CARDS
JOB,CP70.
JOB STATEMENT
2AX56A

Example 9-3. ASCII (029) Coded Punch Input

INPUT Record Size for Coded Cards

Each coded card becomes a W record containing display code characters. The size of the record is determined by the station of origin. Trailing blanks are truncated. There is no record of the number of blanks truncated. If they are significant in your data, there is no way to recreate them.

6000 and 7611-11 Stations

A coded card from the 6000 Station or 7611-11 Service Station has trailing blanks truncated to 2 characters less than a full word, or if the card contains information in columns 79 or 80, the station adds blanks up to 88 characters. Thus, a coded card image is a minimum of 8 characters and a maximum of 88 characters (Figure 9-3). This procedure is required for the JANUS interface between the two systems, which uses a zero-byte terminator.

When reading cards from INPUT, be sure to allow sufficient buffer size. For example, do not set the MRL on a FILE statement for INPUT to 80 for copy routines.

7611-1 Station

A coded card from the 7611-1 Station has all of the trailing blanks removed. If the card contains 79 or 80 characters of data, the record is 79 or 80 characters, respectively. Figure 9-3 illustrates coded card images on INPUT.

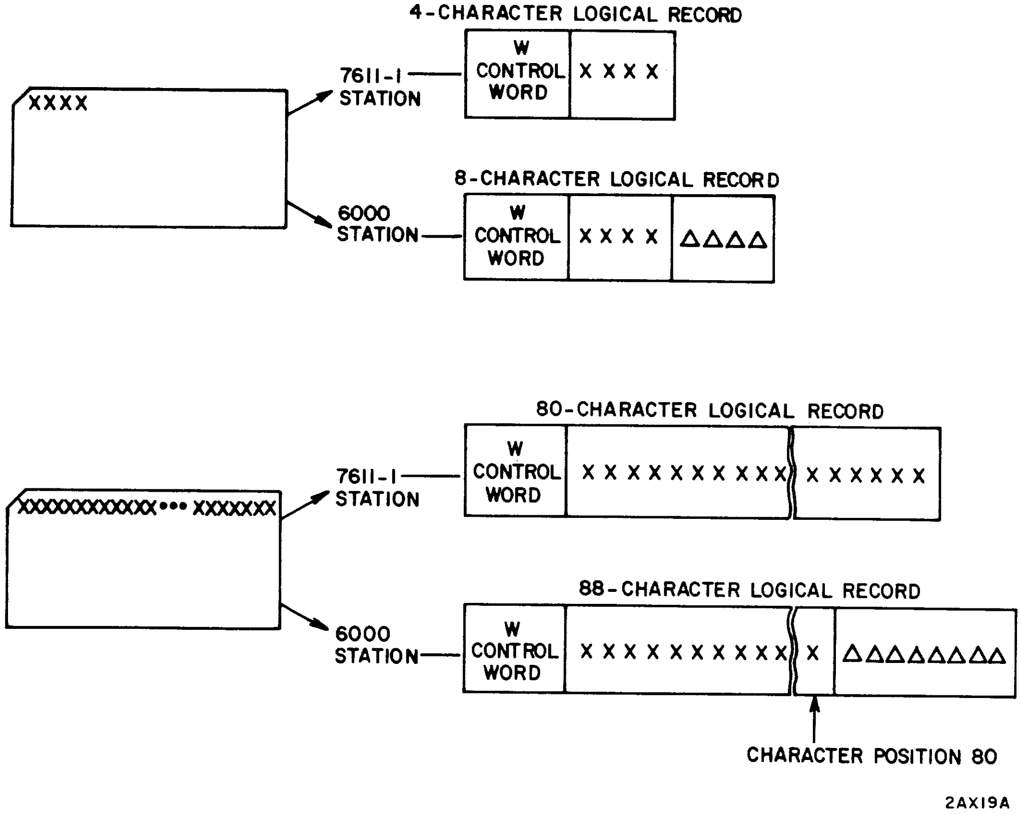


Figure 9-3. Coded Card Images as W Records on INPUT

SCOPE BINARY CARD INPUT

Figure 9-4 illustrates a SCOPE binary card. The station recognizes the card as SCOPE binary by the presence of the 7/9 punch in column 1. It takes the data from columns 3 through 77 and packs it into W records, the size of which is determined by the station. For the 6000 Station, for example, each W record is 630 characters.

A binary card contains up to 15 60-bit CPU words starting at column 3. Column 1 contains a count of 60-bit words in rows 0, 1, 2, and 3, plus a check indicator in row 4. If row 4 of column 1 is 0, column 2 is used as a checksum for the card on input; if row 4 is 1, no check occurs on input.

Column 78 of a binary card is unused; columns 79 and 80 contain a binary sequence number.

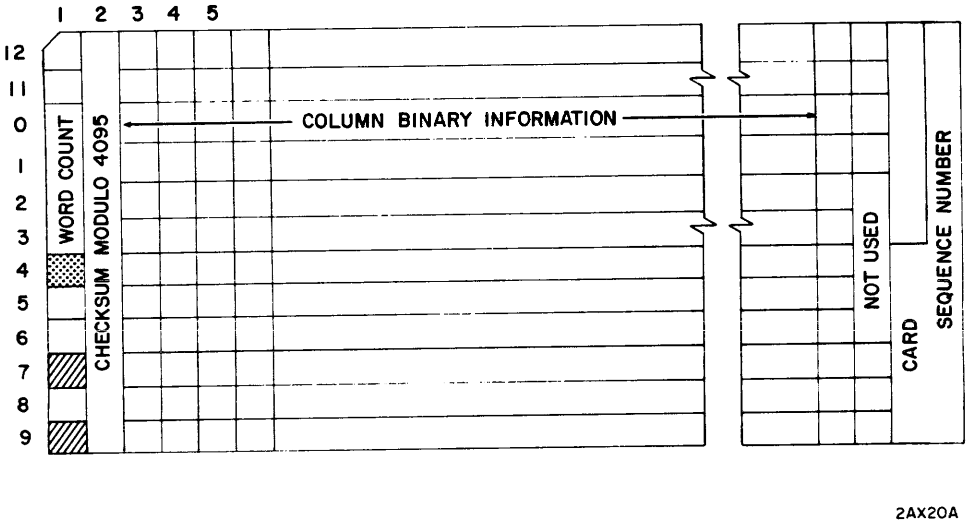


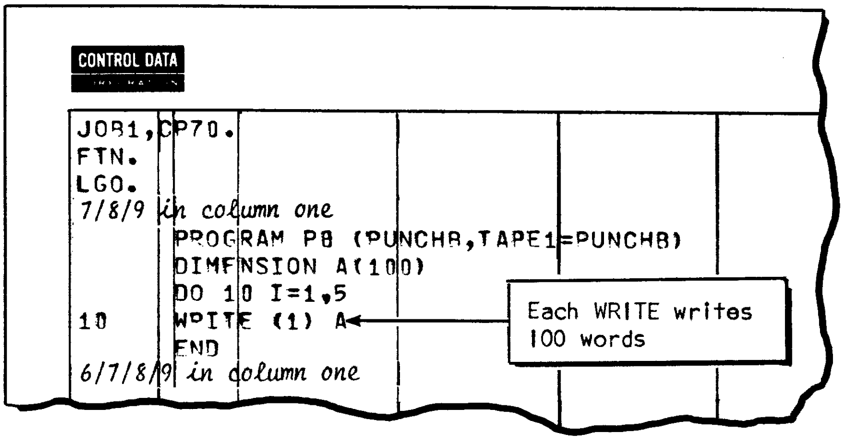
Figure 9-4. SCOPE Binary Card

All of the decks to be loaded by the loader, that is, object modules and core image modules consist of SCOPE binary cards. For a more complete description of the loader tables that comprise the object module or core image module, refer to the Loader Reference Manual. The loader ignores the W record delimiters when it reads SCOPE binary decks. The record delimiters are significant in a FORTRAN or COBOL program.

NOTE

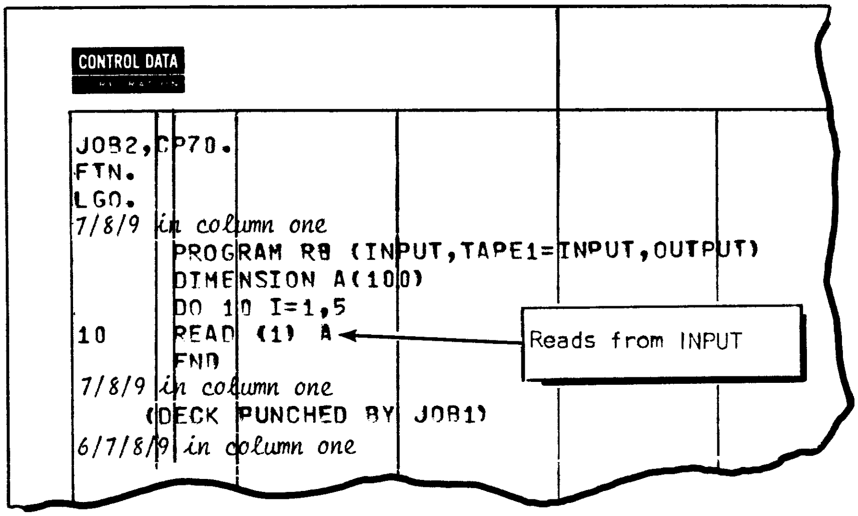
When you use FORTRAN or COBOL I/O statements to read a formatted binary record, you do not receive the data between two 7/8/9 cards as you would on previous operating systems. Instead, you receive a W record, the size of which depends on the station of origin. For example, for a 6000 Series or 7611-11 Station, you receive 630 6-bit characters of data.

As shown in Example 9-4, this does not promote a one-to-one correspondence between output statements and input statements. (See also, Punched Card Output, page 9-18.)



```
JOB1,CP70.
FTN.
LGO.
7/8/9 in column one
      PROGRAM PB (PUNCHB,TAPE1=PUNCHB)
      DIMENSION A(100)
      DO 10 I=1,5
10    WRITE (1) A
      END
6/7/8/9 in column one
```

Each WRITE writes 100 words



```
JOB2,CP70.
FTN.
LGO.
7/8/9 in column one
      PROGRAM RB (INPUT,TAPE1=INPUT,OUTPUT)
      DIMENSION A(100)
      DO 10 I=1,5
10    READ (1) A
      END
7/8/9 in column one
     (DECK PUNCHED BY JOB1)
6/7/8/9 in column one
```

Reads from INPUT

Example 9-4. FORTRAN Binary Input (6000/CYBER 70 Station)

FREE-FORM BINARY CARD INPUT

Free-form binary cards, also referred to as 80-column binary, contain data in all 80 columns.

When you place free-form binary decks in your job deck, a binary flag card must precede and follow the free-form deck (see Figure 9-5). This flag card has all rows of column 1 punched and all rows of any other column punched. The card can contain no other punches.

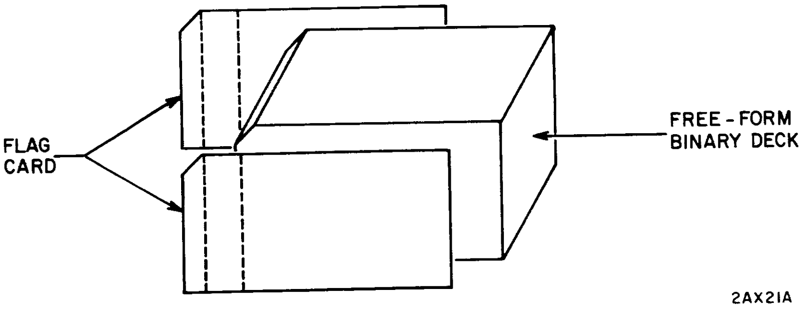


Figure 9-5. Flag Cards to Delimit Free-Form Binary Deck

When SCOPE 2 produces free-form binary decks, the first card and the last card of the deck (except for EOS, EOP, or EOI cards) are flag cards. If the free-form deck was created under some other system, you must add these cards.

The station does not transfer the flag cards to the INPUT file. Each free-form binary card is transferred as a W record consisting of 160 display-coded characters (Figure 9-6).

EOS and EOP cards are not recognized inside of free-form binary decks.

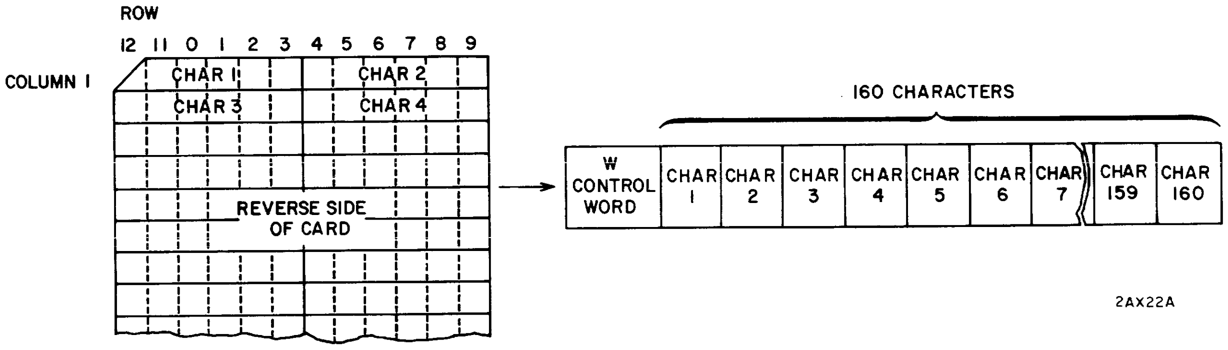


Figure 9-6. Free-Form Binary Card Translation

NOTE

When you use an I/O statement to read free-form binary, you do not receive all the data between the flag cards. Instead, you receive a W record, that is, 160 characters of data.

PRINTER (LIST) OUTPUT

Every job processed by SCOPE 2 results in some printer output, even if it consists merely of the dayfile history of the job. Much of the printer output you will receive is generated by the compilers, assemblers, and utility programs. You will have more direct control over other list output resulting from output statements you have used in your source language program.

Whether you are responsible for formatting the list output or whether it is formatted for you, the data to be printed must adhere to certain requirements and conventions.

1. To be printed, a file must consist of unblocked W records.

2. The file is assumed to consist of display code characters. That is, every six bits are interpreted as the display code representation for the character to be printed. (See Appendix A.)

3. The first character of each W record is assumed to be a carriage control character.

4. Normally, 137 characters including the carriage control character constitute a print line. For the 6000/CYBER 70 Station and 7611-11 Station, if a record is greater than 137 characters, additional characters are continued on subsequent lines. Only the first character of the record is interpreted as a printer control character. For the 7611-1 Station, characters beyond the 137th are ignored.

   When printing at the 6000/CYBER 70 Station or 7611-11 Service Station, a double colon (zero byte) in the low order 2 characters of a word prematurely terminates a print line.

Generally, the object program as well as the compilers and assemblers will be writing the list output on the system file named OUTPUT. This file is automatically printed when the job terminates. Many of the compilers and assemblers let you direct your list output to a file other than OUTPUT. Also, you may want to write your output on a file other than OUTPUT and list it. In this case, it is your responsibility to use a DISPOSE statement to route the file to a station where it will be printed. If you do not properly dispose a file, it is lost.

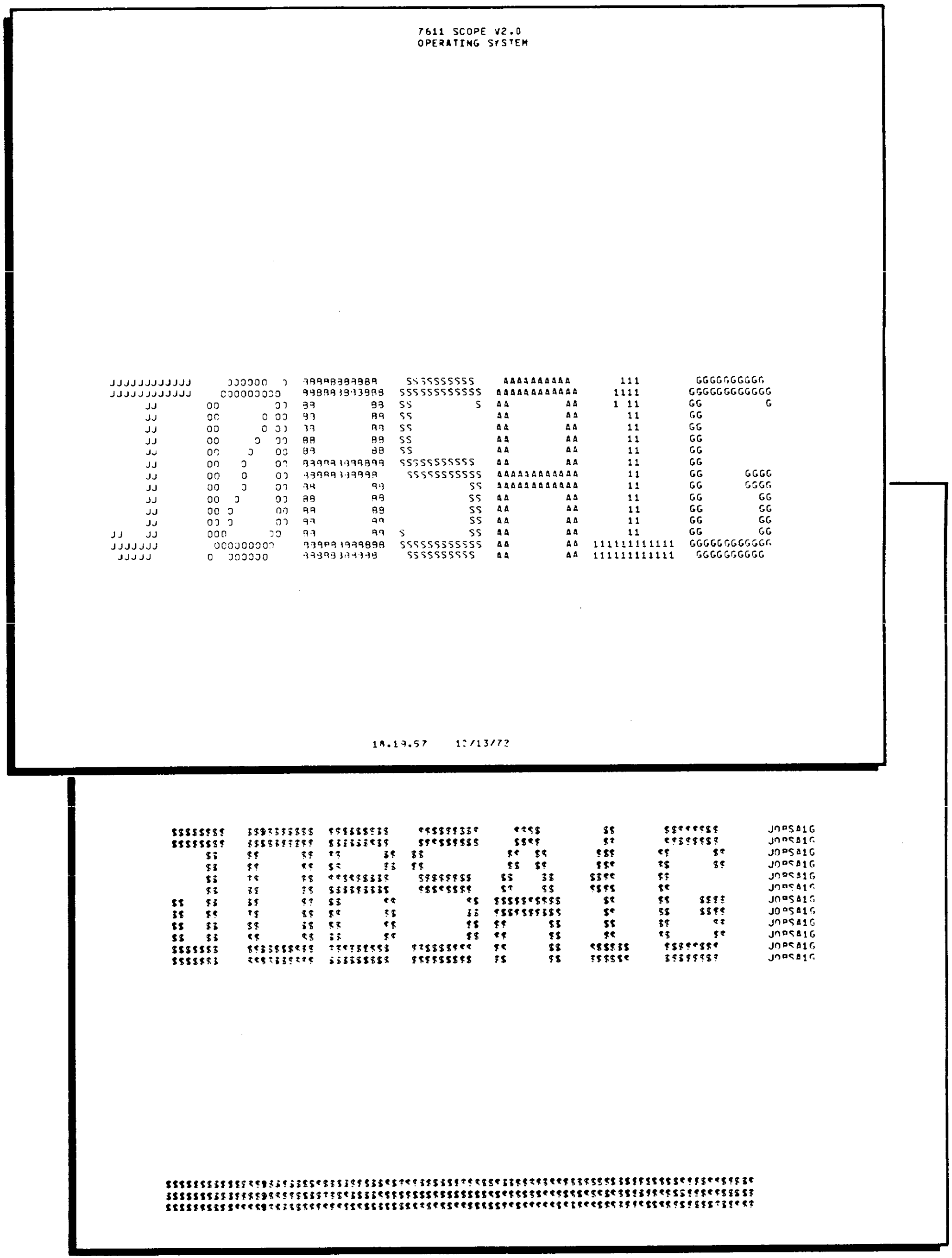
IDENTIFYING PRINTER OUTPUT

Each printout of a file generated by a job begins with two pages containing the modified job name in large characters (Figure 9-7). These are called banner pages and precede data on the OUTPUT file or any other file routed for printing. The banner page helps the operator identify list output for your job and route it back to you.

PRINTER CARRIAGE CONTROL

The first character of each logical record to be printed must be a carriage control character. The character itself is not printed but is part of the record on mass storage. Buffer areas must be large enough to accommodate this character. When the record is copied to a device other than the printer, the character is considered as the first character of the data.

Table 9-1 gives the printer control characters.



```
7611 SCOPE V2.0
OPERATING SYSTEM

18.19.57    12/13/72
```

```
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
JOBSA1G
```

2AX57A

Figure 9-7. Printer Banner Pages (Two Styles)

TABLE 9-1. CARRIAGE CONTROL CHARACTERS

Character	Action
A	Single space, post-print page eject
B C	Single space, post-print skip to last line of page (line 64)
D	Single space, post print page eject
E	Single space, skip to line 4 or 34
F	Single space, skip to line 4, 24, or 44
G	Single space, skip to line 4, 19, 34, or 49
H	Single space, post-print skip to line 2
I J	Single space, post-print page eject
K	Single space, post-print skip to last line of page
L	Single space, post-print skip to line 1
M N O P	Single space (like blank)
PM	Display remainder of line to operator; wait for operator action; not recognized by 7611-1 Station.
Q	Clear auto page eject, suppress remainder of line; default for 6000/CYBER 70 and 7611-11 Stations.
R	Select auto page eject, suppress remainder of line; default for 7611-1 Station
S	Select 6 lines per inch; suppress remainder of line
T	Select 8 lines per inch; suppress remainder of line
U V W	Single space (like blank)
X	Page eject
Y	Skip to last line of page
Z	Skip to line 1
0	Double space
1	Page eject
2 3	Skip to last line of page
4	Page eject
5	Skip to line 4 or 34
6	Skip to line 4, 24, or 44
7	Skip to line 4, 19, 34, or 49
8	Skip to line 2
9	Skip to line 4
+	No spacing
-	Triple space before printing
All special characters act like blanks	

Any preprint operation of 1, 2, or 3 lines that follows a post skip operation will be reduced to 0, 1, or 2 lines.

Issue the functions Q through T at the top of a page. S and T cause spacing to be different from the stated spacing if given in other positions on a page. Q and R cause a page eject before the next line is printed.

Printing 6 lines per inch allows a maximum of 65 lines per page. However, with automatic page eject, only lines 4 to 64 are used, thus allowing for 61 lines. If the printer is switched to 8 lines per inch (either manually or by the carriage control character T), 88 lines can be printed on a page but only lines 5 to 85 are used if pages are skipped automatically.

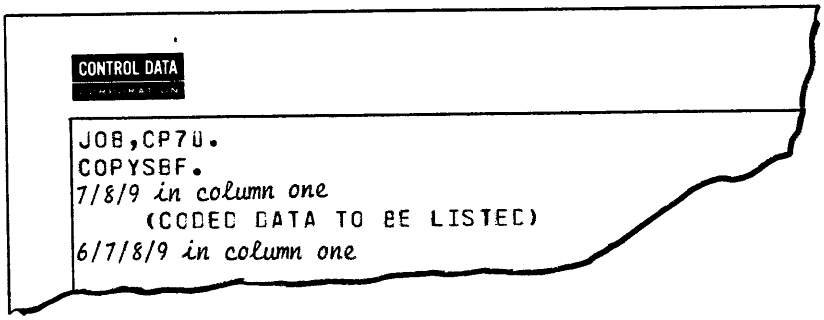
Printing Without a Printer Control Character

Suppose you want to print a file that contains your source language program. If you were to route it to a printer or copy it to OUTPUT, the first character of each line is data and would be interpreted erroneously as a carriage control character. A control statement that you can use to overcome this problem is the COPYSBF (or COPYSP). This stands for Copy Shifted Binary File (or Copy Shifted Partition). The control statement can have either of the following formats.

COPYSBF
or $(\mathrm{lfn}_{\mathrm{in}}, \mathrm{lfn}_{\mathrm{out}})$
COPYSP

The routine takes each record in the next partition of a file, inserts a blank character before the first character, and copies the record to another file. The second file is by default OUTPUT, but can be a file to be disposed. The file must be unblocked W records to be disposed, however. The input file is by default INPUT, but can be any file. Each end-of-section causes a page eject.

Example 9-5 illustrates a job that effectively performs a card-to-print operation; it copies a partition on INPUT to OUTPUT.


CONTROL DATA

```
JOB,CP7U.
COPYSBF.
7/8/9 in column one
     (CODED DATA TO BE LISTED)
6/7/8/9 in column one
```

Example 9-5. Copy INPUT to OUTPUT Shifting Each Record

DISPOSING PRINT FILES TO STATIONS

If you have a mass storage file that meets the requirements of a print file (unblocked W records containing display code characters with the first character a carriage control character), you can cause the file to be immediately routed to a station to be printed by placing the following control statement after the job step that last uses the file.

```
DISPOSE(lfn, PR)
```

You cannot dispose staged or on-line magnetic tape files because they are blocked. However, you can dispose permanent files if they are at the 7600, not at a 6000/CYBER 70 Station.

Print blocked data by copying it to an unblocked file and then dispose the copy of the file.

Type of Printer

Designate the type of printer to use at the 6000/CYBER 70 or 7611-11 Station by using P1 instead of PR to indicate a 501 printer or by using P2 to indicate a 512 printer. Your file will be placed in the queue for the printer of the requested type. The 7611-1 considers PR, P1, and P2 equivalent; all printing takes place on a 517 printer.

Forms Control

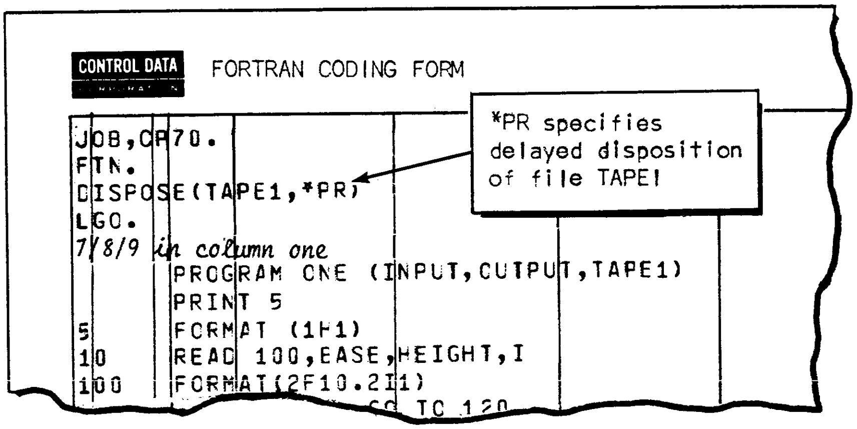
If you want the printing to take place on special forms, you can notify the operator by expanding the disposition code (PR, P1, or P2), as follows:

```
DISPOSE(lfn, Px=Cyy)
```

yy is a two-character code unique to your installation. Check with your systems analyst or the operator to determine what codes, if any, have been assigned to your installation. When your file is routed to the station, the operator is informed via a console display message that he must assign a printer and mount the form you requested.

Delayed Disposition

If you place the DISPOSE control statement before the job steps that create or use your print file (Example 9-6), you must prefix the disposition code with an asterisk to indicate that you do not want immediate routing of the file to the station. The file will be routed at job termination or upon issuing a RETURN or UNLOAD statement. Failure to use * causes an empty file to be sent for printing. This placement of the DISPOSE statement affords a degree of protection for the file description. That is, you will receive an error message if any job step attempts to redefine the file as blocked. It also guarantees that the file will be disposed if an abort condition occurs later in the job, as long as the fatal error did not directly involve the file.



CONTROL DATA
FORTRAN CODING FORM

```
JOB,CP70.
FTN.
DISPOSE(TAPE1,*PR)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
                      GO TO 120
```

*PR specifies delayed disposition of file TAPEI

Example 9-6. Placement of DISPOSE Statement

Routing to Another Station/Terminal

Consider the possibility that you want your file printed at some station or terminal other than the one through which your job was submitted. To specify the destination of your file, add the ST parameter to your DISPOSE statement:

```
DISPOSE(lfn, Px, ST=gggttt)
```

In this case, ggg is a 3-character station identifier assigned to the station when it was logged in (CCP is the default identifier for the 6000/CYBER 70 Station: IOS is the default identifier for the 7611-1 Station). The terminal identifier ttt is optional, but when it is present, it identifies a terminal linked to the station designated by ggg. If ttt is present, the Px parameter cannot include forms control.

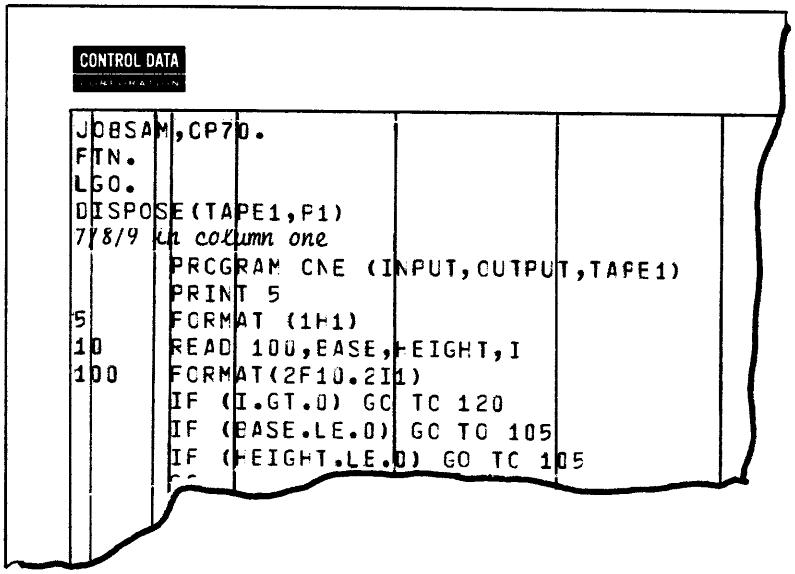
If you are routing to a 6000/CYBER 70 Station or 7611-11 Station EXPORT/IMPORT terminal or INTERCOM terminal, you can designate the terminal by expanding the disposition code rather than using ttt on the ST parameter. This form is compatible with 6000 SCOPE 3.4 disposition.

DISPOSE(lfn, Px=Eyy)	EXPORT/IMPORT terminal
DISPOSE(lfn, Px=Iyy)	IMPORT terminal

When these codes are used, you cannot use forms control nor ttt on the ST parameter.

yy is a 2-character code defined by the installation.

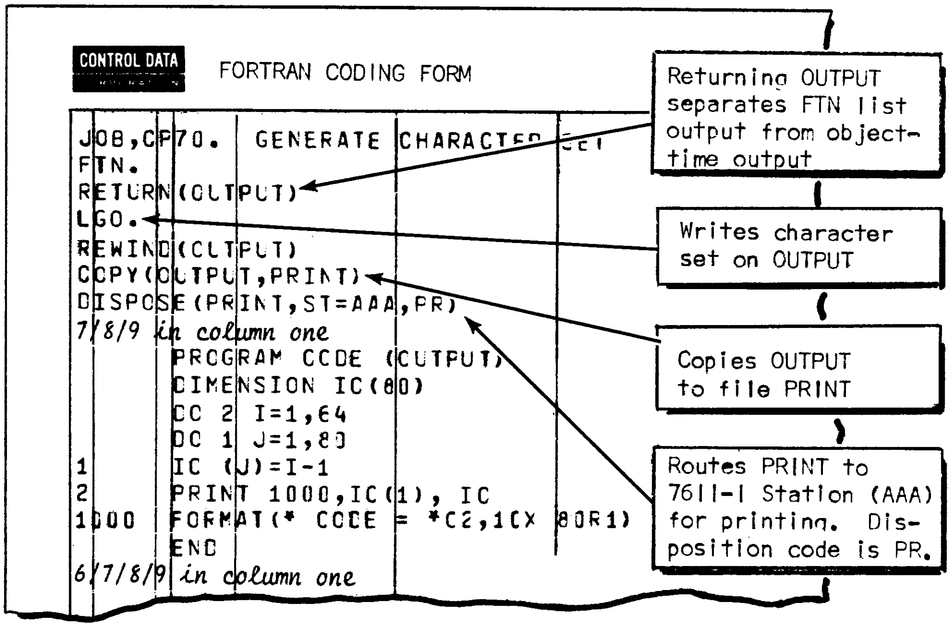
Example 9-7 illustrates a FORTRAN job that creates a print file named TAPE1 and disposes it to a 501 printer.



```
JOBSAM,CP70.
FTN.
LGO.
DISPOSE(TAPE1,P1)
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
      IF (HEIGHT.LE.0) GO TO 105
```

Example 9-7. Disposing Print File Created by FORTRAN Program

Example 9-8 illustrates a FORTRAN program that generates each of the 64 characters in the character set in display code and writes a line of the character on OUTPUT. OUTPUT will be printed on the next available printer at the station of job origin (in this case, a 6000 Station) when the job terminates. Before job termination, the job rewinds OUTPUT (the dayfile has not yet been added) and writes it on a file to be disposed at some other station for comparison. (In this example, the file is routed to a 7611-1.) Note that the printer driver for the 6000/CYBER 70 Station handles the % and : differently from the 7611-1 and that the FORTRAN program is unable to generate the 00 code.



```
JOB,CP70.  GENERATE CHARACTER SET
FTN.
RETURN(OUTPUT)
LGO.
REWIND(OUTPUT)
COPY(OUTPUT,PRINT)
DISPOSE(PRINT,ST=AAA,PR)
7/8/9 in column one
      PROGRAM CODE (OUTPUT)
      DIMENSION IC(80)
      DO 2 I=1,64
      DO 1 J=1,80
1     IC (J)=I-1
2     PRINT 1000,IC(1), IC
1000  FORMAT(* CODE = *C2,10X 80R1)
      END
6/7/8/9 in column one
```

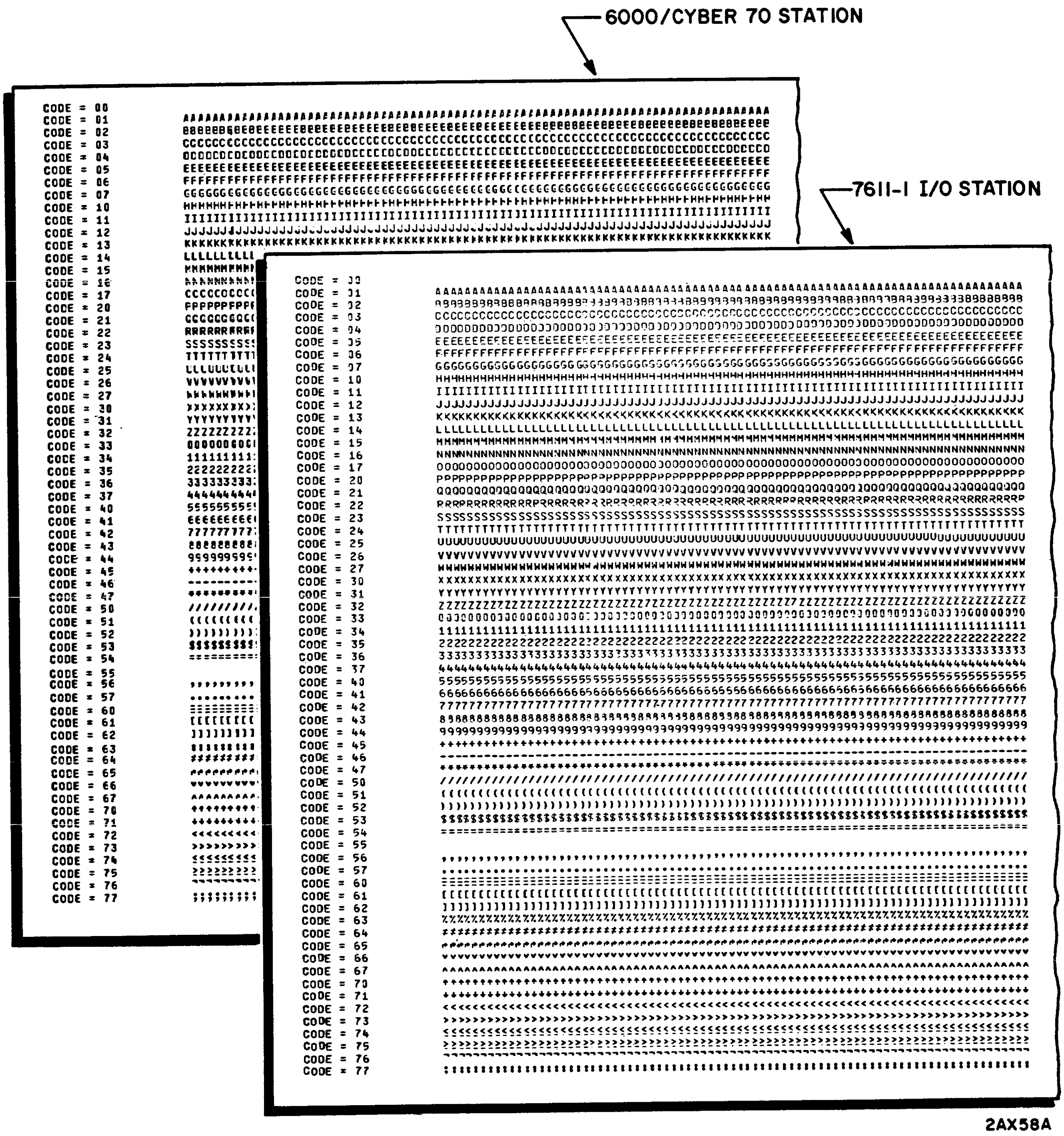
Returning OUTPUT separates FTN list output from object-time output

Writes character set on OUTPUT

Copies OUTPUT to file PRINT

Routes PRINT to 7611-1 Station (AAA) for printing. Disposition code is PR.

Example 9-8. Generate Printer Character Sets


6000/CYBER 70 STATION
7611-1 I/O STATION
2AX58A

```
CODE = 00
CODE = 01          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CODE = 02          BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CODE = 03          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CODE = 04          DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
CODE = 05          EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
CODE = 06          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CODE = 07          GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
CODE = 10          HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
CODE = 11          IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
CODE = 12          JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
CODE = 13          KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
CODE = 14          LLLLLLLLLLLL
CODE = 15          MMMMMMMMMMMM
CODE = 16          NNNNNNNNNNNN
CODE = 17          OOOOOOOOOOOO
CODE = 20          PPPPPPPPPPPP
CODE = 21          QQQQQQQQQQQQ
CODE = 22          RRRRRRRRRRRR
CODE = 23          SSSSSSSSSSSS
CODE = 24          TTTTTTTTTTTT
CODE = 25          UUUUUUUUUUUU
CODE = 26          VVVVVVVVVVVV
CODE = 27          WWWWWWWWWWWW
CODE = 30          XXXXXXXXXXXX
CODE = 31          YYYYYYYYYYYY
CODE = 32          ZZZZZZZZZZZZ
CODE = 33          000000000000
CODE = 34          111111111111
CODE = 35          222222222222
CODE = 36          333333333333
CODE = 37          444444444444
CODE = 40          555555555555
CODE = 41          666666666666
CODE = 42          777777777777
CODE = 43          888888888888
CODE = 44          999999999999
CODE = 45          ++++++++++++
CODE = 46          ------------
CODE = 47          ************
CODE = 50          ////////////
CODE = 51          ((((((((((((
CODE = 52          ))))))))))))
CODE = 53          $$$$$$$$$$$$
CODE = 54          ============
CODE = 55
CODE = 56          ,,,,,,,,,,,
CODE = 57          ...........
CODE = 60          ≡≡≡≡≡≡≡≡≡≡≡
CODE = 61          [[[[[[[[[[[
CODE = 62          ]]]]]]]]]]]
CODE = 63          %%%%%%%%%%%
CODE = 64          ≠≠≠≠≠≠≠≠≠≠≠
CODE = 65          →→→→→→→→→→→
CODE = 66          ∨∨∨∨∨∨∨∨∨∨∨
CODE = 67          ∧∧∧∧∧∧∧∧∧∧∧
CODE = 70          ↑↑↑↑↑↑↑↑↑↑↑
CODE = 71          ↓↓↓↓↓↓↓↓↓↓↓
CODE = 72          <<<<<<<<<<<
CODE = 73          >>>>>>>>>>>
CODE = 74          ≤≤≤≤≤≤≤≤≤≤≤
CODE = 75          ≥≥≥≥≥≥≥≥≥≥≥
CODE = 76          ¬¬¬¬¬¬¬¬¬¬¬
CODE = 77          ;;;;;;;;;;;
```

```
CODE = 00
CODE = 01          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CODE = 02          BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CODE = 03          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CODE = 04          DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
CODE = 05          EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
CODE = 06          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CODE = 07          GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
CODE = 10          HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
CODE = 11          IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
CODE = 12          JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
CODE = 13          KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
CODE = 14          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
CODE = 15          MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
CODE = 16          NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
CODE = 17          OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
CODE = 20          PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
CODE = 21          QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
CODE = 22          RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
CODE = 23          SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
CODE = 24          TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
CODE = 25          UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
CODE = 26          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
CODE = 27          WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
CODE = 30          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CODE = 31          YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
CODE = 32          ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
CODE = 33          000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
CODE = 34          111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
CODE = 35          222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
CODE = 36          333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
CODE = 37          444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
CODE = 40          555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
CODE = 41          666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
CODE = 42          777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
CODE = 43          888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
CODE = 44          999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
CODE = 45          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CODE = 46          ------------------------------------------------------------------------------------------------
CODE = 47          ************************************************************************************************
CODE = 50          ////////////////////////////////////////////////////////////////////////////////////////////////
CODE = 51          ((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
CODE = 52          ))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
CODE = 53          $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
CODE = 54          ================================================================================================
CODE = 55
CODE = 56          ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CODE = 57          ................................................................................................
CODE = 60          ≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡≡
CODE = 61          [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
CODE = 62          ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
CODE = 63          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CODE = 64          ≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠
CODE = 65          →→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→→
CODE = 66          ∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨
CODE = 67          ∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧∧
CODE = 70          ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
CODE = 71          ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
CODE = 72          <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CODE = 73          >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
CODE = 74          ≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤≤
CODE = 75          ≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥≥
CODE = 76          ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
CODE = 77          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Example 9-8. Generate Printer Character Sets (Cont'd)

PUNCHED CARD OUTPUT

Earlier, we described how to read punched cards from your job deck. Now, we shall describe how these cards can be generated as output from your job. Punch options include coded punched cards (026 or 029), formatted binary cards, and free-form binary cards.

IDENTIFYING PUNCHED OUTPUT

Each time a file is punched, the deck begins with a lace card that contains the modified job name in large characters. The lace card (Figure 9-8) precedes data automatically punched from files routed for punching. The lace card helps the operator identify punched output for your job so that he can route it back to you.

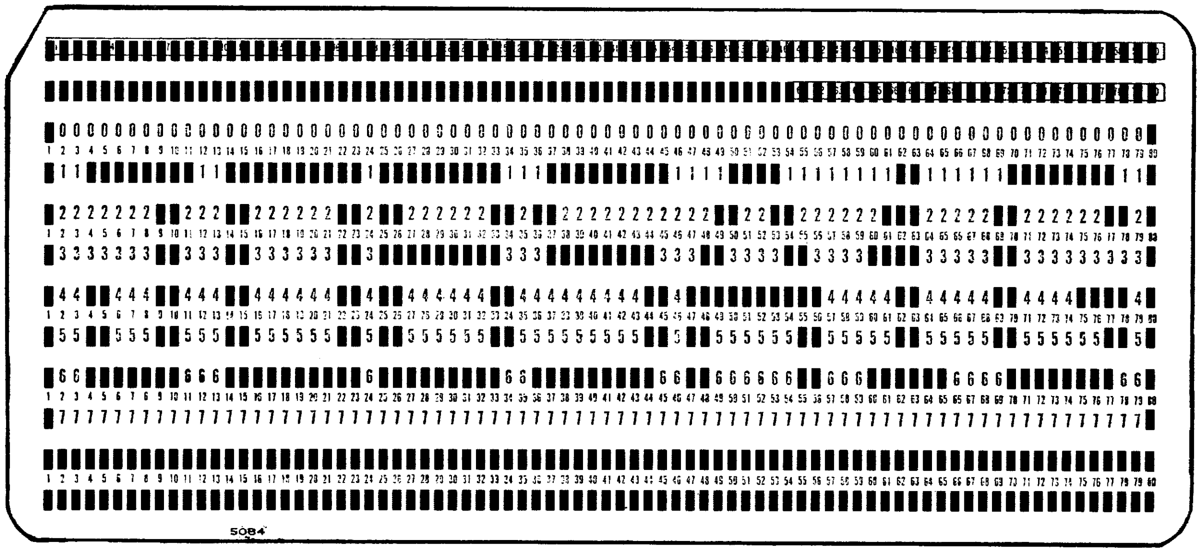


Figure 9-8. Lace Card

SEPARATOR CARDS

Any delimiters in the file (that is, EOS, EOP, or EOI W control words) cause a corresponding file separator card to be punched, and for some stations, offset in the punch hopper. W control words for records are not punched.

MISPUNCHED CARDS

If the system detects an error while punching a card, it offsets the card and repunches the data. If you have difficulty reading punched cards, check to see that these mispunched cards have been removed. In particular, check the sequence numbers (in columns 79 and 80) for SCOPE binary cards.

CODED PUNCHED CARDS

For a file to be punched as coded cards, it should conform to the following conventions.

1. It must contain unblocked W records.
2. The file is assumed to consist of display code characters, that is, every six bits are interpreted as the display code representation of the character to be punched.
3. Normally, the record consists of 80 or less characters which constitute a single punched card. If a record exceeds 80 characters, the 6000/CYBER 70 Station and 7611-11 Station continue the record on as many cards as are needed to hold the record. Each record begins a new card. Remember, that when the cards are read in, each card becomes a W record so there is not necessarily a one-to-one correspondence between reads and writes. For the 7611-1, if a record exceeds 80 characters, information beyond the 80th character is ignored; only one card is punched.

When the station punches a file, it does not punch the W control words. You can implicitly request coded punch output by writing on the PUNCH file, or you can explicitly route the file to a station for punching through the DISPOSE statement.

SCOPE BINARY PUNCHED CARDS

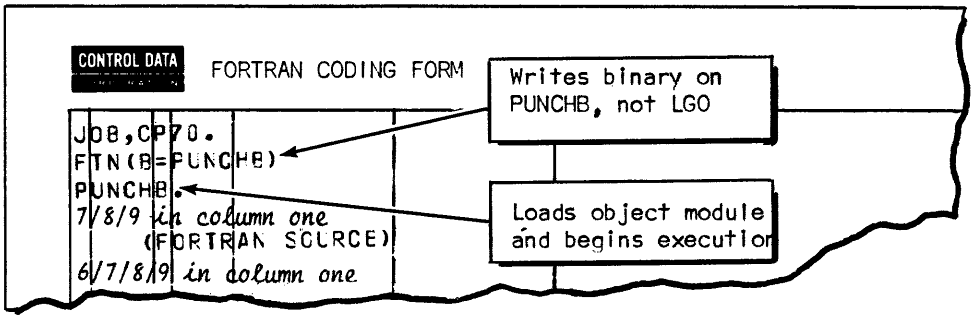
For a file to be punched as SCOPE binary cards, it must contain unblocked W records.

Normally, the record consists of a complete core image module or object module deck and is intended as input for the loader. The file is punched as if it contained pure binary information, but it can contain display code information. Punching coded information in binary (sometimes called "crunch" format) provides a more compact card deck than punching it in coded format, but presents some problems when you are trying to reconstruct the logical records on input.

When the station punches a binary file, it does not punch W control words. Thus, there is no way for you to know whether the data consisted of one or many records before it was punched or where the end-of-records occurred. Usually, the deck results from a single output statement and consists of a single record, the maximum size of which is limited only by the maximum for W records.

You can implicitly request formatted punched binary output by writing on the PUNCHB file, or you can explicitly route a punch file to a station by using the DISPOSE control statement.

Example 9-9 shows compiler output being directed to PUNCHB instead of to LGO.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN(B=PUNCHB)
PUNCHB.
7/8/9 in column one
     (FORTRAN SOURCE)
6/7/8/9 in column one
```

Writes binary on PUNCHB, not LGO

Loads object module and begins executio

Example 9-9. Punching Binary Output from Compiler

FREE-FORM BINARY PUNCHED CARDS

Any file containing unblocked W records can be punched in free-form binary. There is no system file that is automatically routed to a station and punched in free-form binary. You can request free-form binary only through the DISPOSE statement.

When the station punches free-form binary cards, it precedes and follows the deck with a free-form flag card.

W control words are not punched. Thus, there is no way for you to know by examining the deck whether the file consisted of one or many records or where the end-of-records occurred. On input, the deck will be divided into 160-character W records.

ROUTING PUNCH FILES TO STATIONS

If you have a mass storage punch file other than PUNCH and PUNCHB, you can route it to a station and have it punched by placing a DISPOSE statement after the job step that last uses the file. You cannot dispose staged or on-line magnetic tape files (remember they are blocked).

You can dispose permanent files at the 7600 but not at the 6000/CYBER 70 Stations. For tape files, you can copy the data to an unblocked file and then dispose the copy.

```
DISPOSE(lfn, disposition)
```

Disposition is a 2-character code designating the punch format, as follows:

PU	Punch Hollerith (026) coded format
PB	Punch SCOPE binary
P8	Punch free-form binary
PA	Reserved for future use
PZ	
P9	

Forms Control

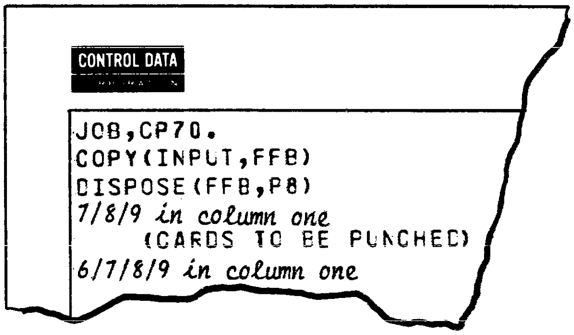
If you want to punch special cards, notify the operator by expanding the disposition code. For example,

```
DISPOSE(lfn, Px=Cyy)
```

yy is a 2-character code unique to your installation. Check with your system analyst or the operator to determine what codes, if any, have been assigned to your installation. When your file is routed to the station, the operator is informed via a console display message that he must assign a punch and load the cards you requested into the punch hopper. Your file is punched when he informs the system that he has taken the requested action.

Other options for disposing punch files resemble those described for print files, page 9-14.

Example 9-10 shows card input being copied to free-form binary. INPUT is copied to file FFB, which is disposed as P8. You will receive an exact copy of the input deck if it is in free-form binary. If the input deck is in coded format, the output will be compressed. If the input deck is SCOPE binary, the data in columns 3 through 77 will be copied onto the free-form binary cards.


CONTROL DATA

```
JOB,CP70.
COPY(INPUT,FFB)
DISPOSE(FFB,P8)
7/8/9 in column one
    (CARDS TO BE PUNCHED)
6/7/8/9 in column one
```

Example 9-10. Punching Free-Form Binary

COPYING AND POSITIONING FILES 10

This section presents the general principles of copying files. It does not consider the impact of factors such as code conversion, labels, station choice, device type, etc. The impact of these factors on files in general is covered in previous sections.

INTRODUCTION TO COPY ROUTINES

Users familiar with 6000 SCOPE 3.4 and earlier systems will be accustomed to the idea that a COPY statement is tailored to a specific purpose and that the contents and format of the output file exactly duplicate the contents and format of the input file. Thus, COPYBF served to copy a binary file and COPYCF served to copy a coded file. SCOPE 3.4 copy routines do not use record manager. Files are copied as binary or coded. Any FILE statements are ignored.

Under SCOPE 2, the copy routines all use the record manager, and as a result, the scope of the copy routines has been broadened considerably. Indeed, a copy as formerly viewed, becomes a special case - a case in which conversion does not take place. In this context, conversion means file redefinition, that is, any change between the copy input file and the output file record type, block type, or file organization. This is not to be confused with character code conversion which is described in Section 6, Magnetic Tape Files.

Copying of files generally has no effect on the data in the record. In some cases, however, if the record type has changed between the input file and the output file, the record is expanded to a new fixed length or is truncated as required by the new record type. (See Appendix E, Using Record Manager for File Conversion.)

HOW TO COPY FILES

Copying files involves the following steps.

Selecting a COPY statement

Describing the formats of the input file and the output file

Specifying the size of the copy buffer

Copy routines open the files for I/O. This initiates prestaging of the input file, if necessary. The routines do not reposition the files before copying. Upon completion, copy routines do not close the files. A message is placed in the dayfile stating the number of EOR, EOS, and EOP delimiters encountered during the copy. The routines do not copy labels.

SELECTING THE COPY

For all copies, the default input and output files are INPUT and OUTPUT; the default number of records, sections, or partitions (n) is 1.

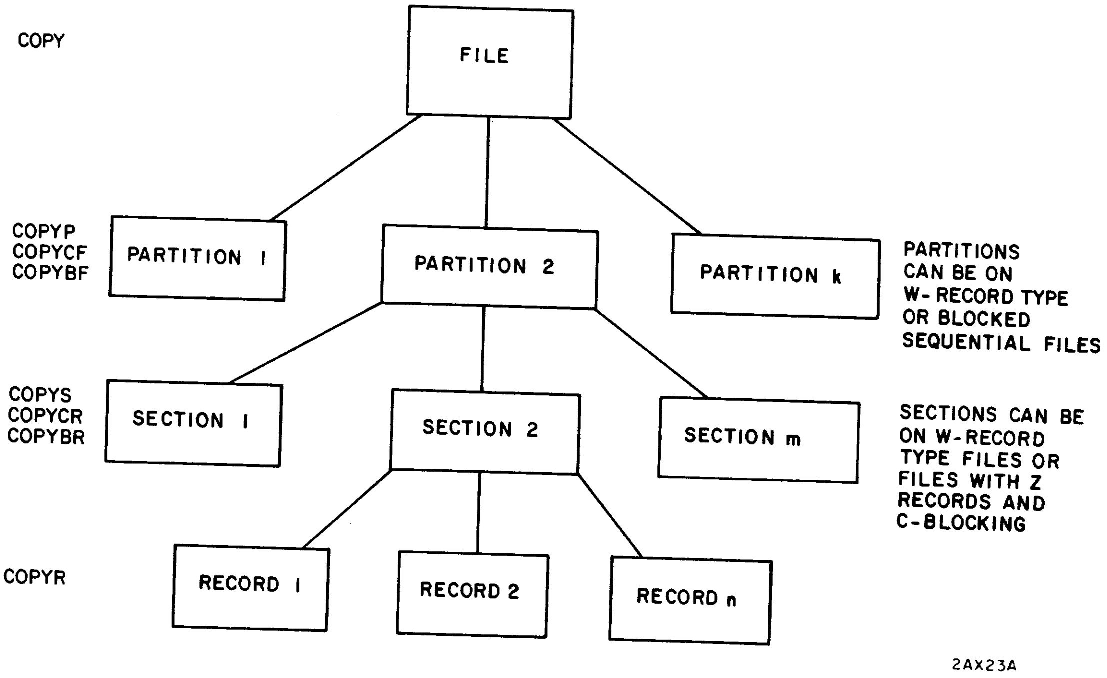

COPY
FILE
COPYP
COPYCF
COPYBF
PARTITION 1
PARTITION 2
PARTITION k
PARTITIONS CAN BE ON W-RECORD TYPE OR BLOCKED SEQUENTIAL FILES
COPYS
COPYCR
COPYBR
SECTION 1
SECTION 2
SECTION m
SECTIONS CAN BE ON W-RECORD TYPE FILES OR FILES WITH Z RECORDS AND C-BLOCKING
COPYR
RECORD 1
RECORD 2
RECORD n
2AX23A

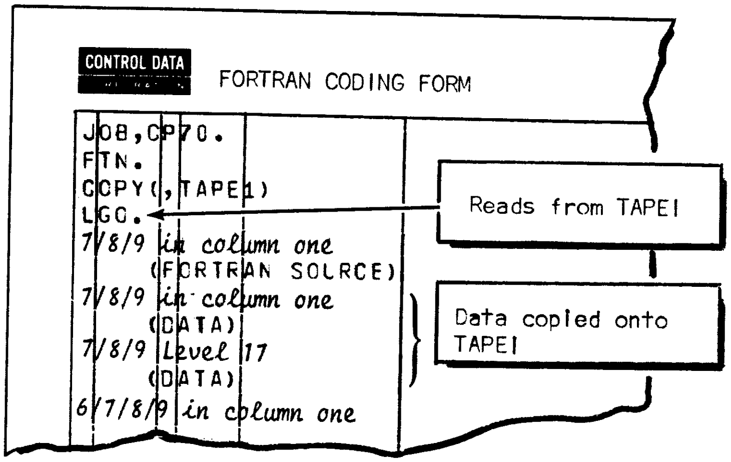
Figure 10-1. Selecting the Proper Copy Statement

File

To copy an entire file from its current position to end-of-information, use the following control statement.

COPY(lfn_{in},lfn_{out})

Example 10-1 shows COPY being used to copy the data on INPUT to file TAPE1.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN.
COPY(,TAPE1)
LGO.
7/8/9 in column one
      (FORTRAN SOURCE)
7/8/9 in column one
      (DATA)
7/8/9 Level 17
      (DATA)
6/7/8/9 in column one
```

Reads from TAPE1

Data copied onto TAPE1

Example 10-1. File-to-File Copy

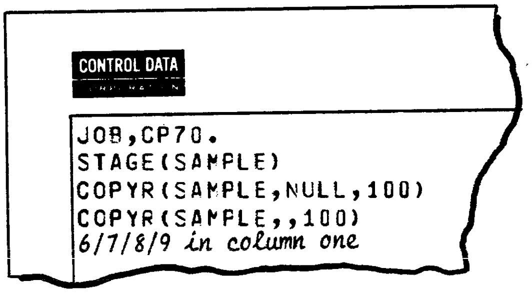
Records

When copying records, use the following control statement.

COPYR(lfn$_{in}$, lfn$_{out}$, n)

The number of records (n) is expressed in decimal.

Example 10-2 illustrates a job that effectively skips 100 records by copying them to a scratch file. It then copies the next 100 reçords to the OUTPUT file.



```
JOB,CP70.
STAGE(SAMPLE)
COPYR(SAMPLE,NULL,100)
COPYR(SAMPLE,,100)
6/7/8/9 in column one
```

Example 10-2. Copying Records

Sections

When copying sections, use one of the following control statements:

COPYS(lfn$_{in}$, lfn$_{out}$, n)

COPYBR(lfn$_{in}$, lfn$_{out}$, n)

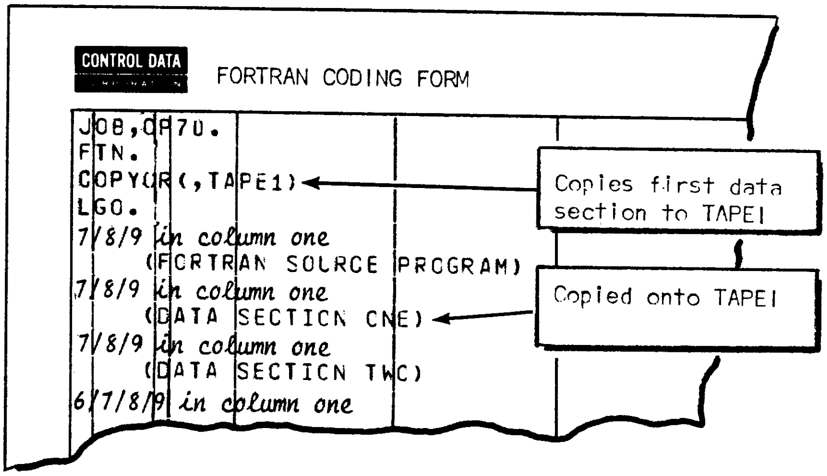
COPYCR(lfn$_{in}$, lfn$_{out}$, n)

The results are identical regardless of the statement used. However, COPYBR and COPYCR provide a degree of compatibility with SCOPE 3.4. COPYS is not available under SCOPE 3.4.

Copying sections is meaningful only when the input record type is W, or is S or Z with C blocking. In the special case of copying S records, each S record becomes a W section.

Upon completion of the copy, an EOS delimiter is written on the output file. The number of sections (n) is expressed in decimal.

In Example 10-3, the program reads from files TAPE1 and INPUT. The first data section on INPUT is copied onto TAPE1.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CM70.
FTN.
COPYCR(,TAPE1)
LGO.
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
7/8/9 in column one
     (DATA SECTION ONE)
7/8/9 in column one
     (DATA SECTION TWO)
6/7/8/9 in column one
```

Copies first data section to TAPE1

Copied onto TAPE1

Example 10-3. Copying Sections

Partitions

When copying partitions, use one of the following:

COPYP(lfn$_{in}$, lfn$_{out}$, n)

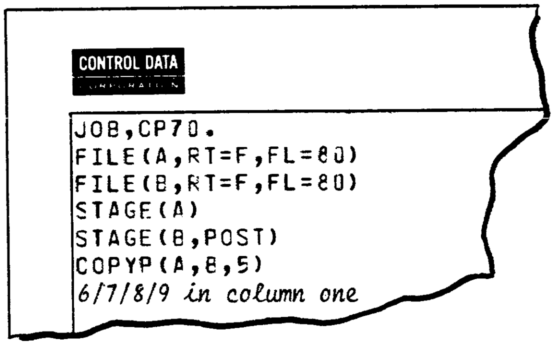
COPYBF(lfn$_{in}$, lfn$_{out}$, n)

COPYCF(lfn$_{in}$, lfn$_{out}$, n)

The results are identical regardless of the statement used. However, COPYBF and COPYCF provide a degree of compatibility with SCOPE 3.4. COPYP is not available under SCOPE 3.4.

Copying partitions is meaningful for W record type files (either blocked or unblocked) and for blocked files using other record types. It is not meaningful for unblocked sequential or word addressable files. Also, if you are performing a U to U copy of a file that contains W records or S or Z records with C blocking, the partition delimiters inherent to the original record type will not be recognized.

The number of partitions to be copied is expressed in decimal. Upon completion of the copy, an EOP is written on the output file. In Example 10-4, file A is a tape containing several partitions. Since it contains F records, partitions are indicated by tapemarks. The job copies the first five partitions to file B. File B is then post-staged to magnetic tape.


CONTROL DATA

```
JOB,CP70.
FILE(A,RT=F,FL=80)
FILE(B,RT=F,FL=80)
STAGE(A)
STAGE(B,POST)
COPYP(A,B,5)
6/7/8/9 in column one
```

Example 10-4. Copying Partitions

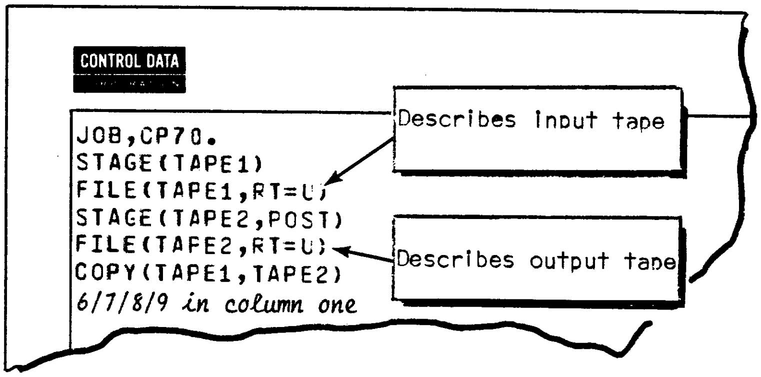
FILE DESCRIPTIONS FOR COPIES

Describe your input file and output file exactly the same, using FILE statements if necessary. The SCOPE 2 COPY routines do not check to see that the output file description matches the input file description. For an exact copy, make sure that both files are either unblocked or blocked by specifying the BT parameter. Be especially careful of this when copying to or from magnetic tape since a magnetic tape file is blocked by default and mass storage and unit record files are unblocked. You can, of course, purposely use this feature to block or deblock a file. The fastest copy occurs when both files have record type W. Deleted records on the input file are not copied to the output file. The record manager recognizes and regenerates all EOS and EOP delimiters (zero length W control words). However, the W control words on the new file may not compare bit-for-bit with those on the original file because the irrelevant fields on the new control words are not zeroed but are filled with transient information.

When the input file does not contain W records, redefining it as record type U is most efficient because there is much less data manipulation required by the record manager.

Example 10-5 illustrates a job that stages in a tape as record type U, block type K, copies it, and stages it out. Both tapes are recorded in odd parity (binary mode) at 800 bpi and are unlabeled.

The input tape consists of 5120-character blocks that actually contain a variable number of records in record mark (R) format. By defining the file as U type rather than R type, the R records are simply copied as data without being examined on input or output. This is a considerably faster mode of copying than defining the files as R record type. Upon completion of the copy, the dayfile actually contains a block count instead of a record count.



CONTROL DATA
Describes input tape
Describes output tape

```
JOB,CP70.
STAGE(TAPE1)
FILE(TAPE1,RT=U)
STAGE(TAPE2,POST)
FILE(TAPE2,RT=U)
COPY(TAPE1,TAPE2)
6/7/8/9 in column one
```

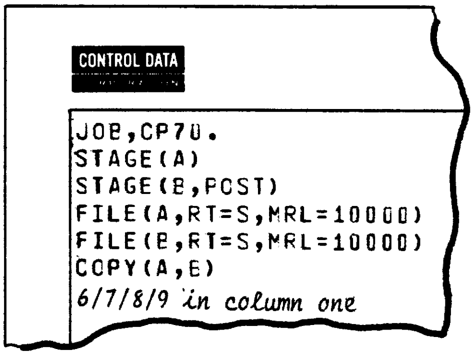
Example 10-5. Copying a Tape as Record Type U, Block Type K

SPECIFYING BUFFER SIZE

Check to see that the copy buffer is adequate for the copy. When SCM dynamic memory assignment is in effect, the size of the buffer used for copying will be the default maximum record length (5120 characters) unless a different MRL is specified on a FILE statement. MRL or FL can be specified for either file or both the input and output files. The copy routine uses whichever explicitly defined MRL or FL is larger for the two files. Thus, if no MRL or FL is specified for the input file and MRL=88 is specified for the output file, the buffer size used for the copy is 88 characters.

For T- and D-type records, MRL must be large enough to include the count field. Otherwise, MRL can be less than the actual record size. If records are short, it is desirable that MRL be specified to reduce core requirements. Copies are more efficient when the actual MRL for the files is specified.

Example 10-6 illustrates a tape-to-tape copy of SCOPE logical (S) records. The longest record on the file is 10000_8 characters.



CONTROL DATA

```
JOB,CP70.
STAGE(A)
STAGE(B,POST)
FILE(A,RT=S,MRL=10000)
FILE(B,RT=S,MRL=10000)
COPY(A,B)
6/7/8/9 in column one
```

Example 10-6. Setting MRL

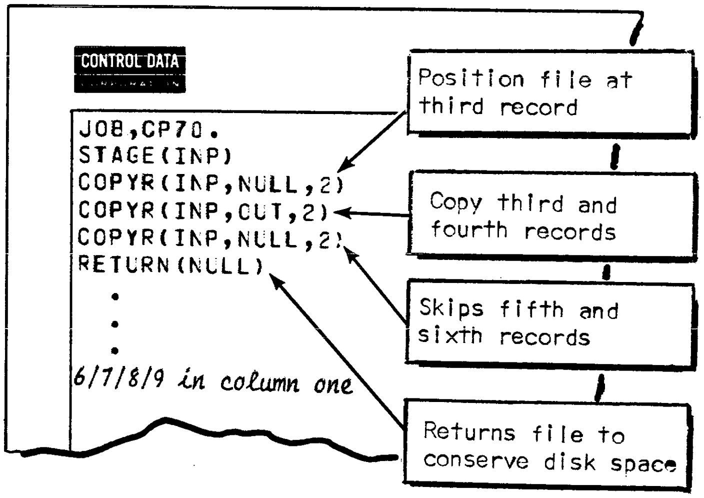
POSITIONING SEQUENTIAL FILES

Sequential files often require repositioning before or after their use. This is not true for files using other file organizations (for example, word addressable). A control statement request to position a file usually initiates staging in if the file is a prestaged file that has not yet been staged in. REWIND is an exception; it does not initiate staging. For on-line tapes and for tapes staged by volume, the operator is told to mount tapes if additional volumes are required to complete the positioning request in the forward direction. Staged tapes are rewound to the beginning of information but on-line tapes are rewound to the beginning of the current volume. Neither staged or on-line tapes can be skipped backward past the beginning of volume.

POSITIONING FILES FORWARD

Skipping Records Forward

SCOPE 2 does not have a SKIPF statement for skipping records forward. A technique that can be used to skip records is that of copying the records to be skipped onto a null or dummy file. Example 10-7 illustrates this technique.



```
JOB,CP70.
STAGE(INP)
COPYR(INP,NULL,2)
COPYR(INP,OUT,2)
COPYR(INP,NULL,2)
RETURN(NULL)
    •
    •
    •
6/7/8/9 in column one
```

CONTROL DATA
Position file at third record
Copy third and fourth records
Skips fifth and sixth records
Returns file to conserve disk space

Example 10-7. Skipping Records Forward

Skipping Sections Forward

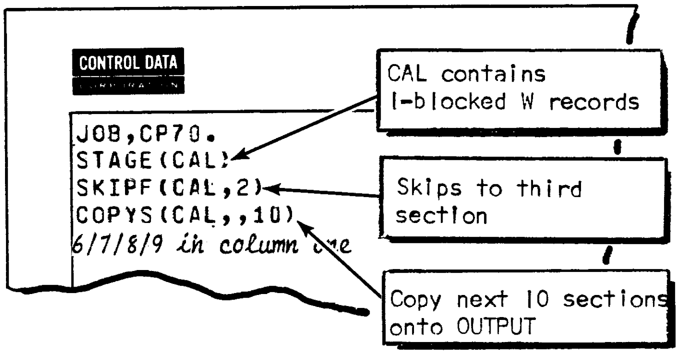
SCOPE 2 provides a SKIPF statement that can be used for skipping sections forward on those file types that support sections (Z with C blocking and W) and as a special consideration, can be used for skipping S records. The statement has the following format:

```
SKIPF(lfn, n)
```

Parameter n is a decimal count of the sections (or S records) to be skipped. The default is 1.

The SKIPF terminates at EOI if it fails to encounter the requested number of sections or S records on the file. Encountering an EOP does not terminate the skip nor is the EOP included in the count of sections - unless it is also an EOS. That is, on W records, the EOP is counted if it is not immediately preceded by an EOS W control word.

Example 10-8 illustrates skipping of sections forward followed by a tape-to-print copy of 10 sections.



```
JOB,CP70.
STAGE(CAL)
SKIPF(CAL,2)
COPYS(CAL,,10)
6/7/8/9 in column one
```

CONTROL DATA
CAL contains I-blocked W records
Skips to third section
Copy next 10 sections onto OUTPUT

Example 10-8. Skipping Sections Forward

Skipping Partitions Forward

SCOPE 2 provides a parameter on the skip forward statement (SKIPF) for skipping partitions on files having record types W, F, and R, or having record types X, S, and Z with C blocking. Skipping is possible on files having D, T, or U records if the block type is K and records per block are 1. For these record types, skipping forward can be achieved by copying partitions to a null file.

To indicate that partitions rather than sections are to be skipped, the third parameter on the SKIPF statement must be 17_8.

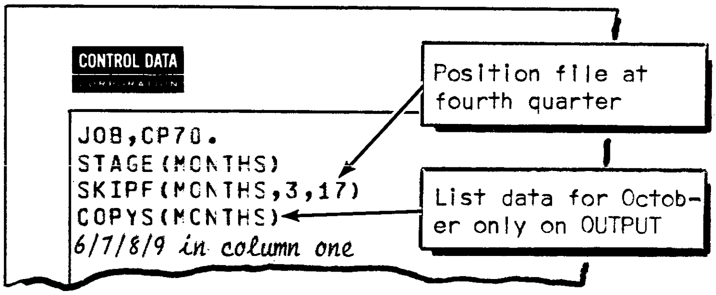
```
SKIPF(lfn,n,17)
```

The skip terminates at EOI if it fails to encounter the requested number of partitions on the file. The default for n is 1.

NOTE

SCOPE 3.4 allows a fourth parameter, the mode parameter, or SKIPF and SKIPB. This parameter, if present, is ignored by SCOPE 2.

Example 10-9 illustrates skipping of partitions on a file divided into four partitions, one for each quarter of the year. Each partition consists of three sections, one for each month in the section.



```
JOB,CP70.
STAGE(MONTHS)
SKIPF(MONTHS,3,17)
COPYS(MONTHS)
6/7/8/9 in column one
```

Position file at fourth quarter

List data for October only on OUTPUT

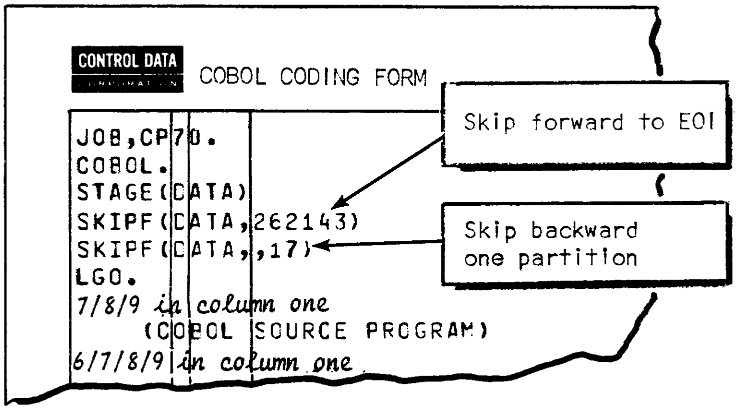
Example 10-9. Skipping Partitions Forward

Skipping to End-Of-Information

For compatibility with 6000 SCOPE systems, SCOPE 2 issues a skip to end-of-information upon encountering the following:

```
SKIPF(lfn,262143)
```

The statement is a no-op if issued when the file is at EOI. Example 10-10 illustrates skipping to EOI. The user knows the data is in the last partition on the file but he doesn't know how many partitions are on the file.



COBOL CODING FORM

```
JOB,CP70.
COBOL.
STAGE(DATA)
SKIPF(DATA,262143)
SKIPF(DATA,,17)
LGO.
7/8/9 in column one
    (COBOL SOURCE PROGRAM)
6/7/8/9 in column one
```

Skip forward to EOI

Skip backward one partition

Example 10-10. Skipping to End-Of-Information

As an added option on the FILE statement, you can specify OF=E to position a mass storage file at EOI when it is first opened. This is convenient when extending a permanent file.

```
FILE(lfn,...,OF=E)
```

POSITIONING FILES BACKWARD

Skipping Records Backward

SCOPE 2 does not have a SKIPB statement for skipping records in the reverse direction. The user must rewind the file and copy records to a null file.

Skipping Sections Backward

SCOPE 2 provides a skip backward statement (SKIPB) and a backspace statement (BKSP) for skipping sections in the reverse direction on those files that support section delimiters (Z with C blocking and W) and as a special consideration can be used for skipping S records backward. The statement formats are:

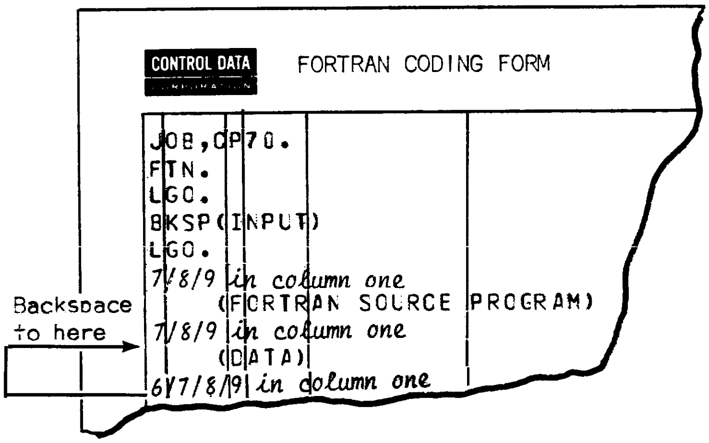
```
SKIPB(lfn,n)
```

```
BKSP(lfn,n)
```

Parameter n is a decimal count of sections to be skipped. The default is 1. The SKIPB or BKSP terminates at EOP or BOI if it fails to encounter the specified number of sections.

For INPUT and other system files, the system terminates the file with an EOS/EOP/EOI sequence. In this case, if you are at end-of-information you cannot skip sections backwards because the first EOP terminates the skip.

Example 10-11 illustrates how BKSP can be used to reposition the INPUT file. In this example, INPUT is backspaced one section before the second execution of LGO. LGO is loaded and executed twice using the same data.



CONTROL DATA FORTRAN CODING FORM

```
JOB,CP70.
FTN.
LGO.
BKSP(INPUT)
LGO.
7/8/9 in column one
    (FORTRAN SOURCE PROGRAM)
7/8/9 in column one
    (DATA)
6/7/8/9 in column one
```

Backspace to here

Example 10-11. Skipping Sections Backward on INPUT

Skipping Partitions Backward

SCOPE 2 provides a parameter on the skip backward statement (SKIPB) for skipping partitions in the reverse direction. Skipping backward is possible on files having record types W, F, and R or having record types S, X, and Z with C blocking. Skipping is possible on files with D, T, or U record types if the files are described as BT=K and RB=1.

To indicate that partitions rather than sections are to be skipped, the third parameter on the SKIPB statement must be 17_8.

```
SKIPB(lfn,n,17)
```

Parameter n is the number of partitions to be skipped. The default is 1. The SKIPB terminates at BOI if it fails to encounter the requested number of partitions.

Skipping to Beginning-Of-Information (Rewinding)

Use the REWIND statement to position one or more files to beginning-of-information (BOI). REWIND does not initiate staging of a prestaged file or transfer of a 6000 Station permanent file.

REWIND($lfn_1, lfn_2, \ldots, lfn_n$)

For compatibility with 6000 SCOPE systems, SCOPE 2 issues a rewind upon encountering either of the following statements:

```
SKIPB(lfn,262143)
```

```
BKSP(lfn,262143)
```

Either statement initiates staging if it is used before a file is prestaged. These statements are no-ops if issued when the file is at BOI.

Specifying Rewind or No Rewind on Open or Close

The OF=p and CF=p options on the FILE statement are allowed for specifying rewind (R) or no rewind (N) of a file when it is next opened (OF) or closed (CF).

```
FILE(lfn,...,OF=p,CF=p)
```

These options may conflict with macros internal to the program causing unpredictable results. In addition, since the loader does not open a file before loading from it, the parameters are not relevant for load files.

WRITING FILE DELIMITERS

A user can insert new delimiters when copying from one record type to another by writing null sections or partitions at the point at which the delimiter is desired. An attempt to write an end-of-section on other than Z with C blocking or W-type files is ignored except for S records where it causes a zero-length record.

COMPARING FILES

The information stored on one file can be compared with that stored on another file to see whether the contents of both units are identical. Often the comparison is desirable after copying to assure that the input file was copied without error. Labels, if present, are not compared. With SCOPE 2, because COMPARE uses the record manager, it is possible to compare files that are exactly identical, including record and blocking structure, or if you have changed the record type or blocking structure, it is possible to compare just the data in the logical records. To perform an exact comparison, both files must be defined the same. For the most straightforward and fastest comparison of two mass storage files, define them both as record type U, unblocked. For this comparison the record manager does not manipulate the data.

NOTE

> The two files will fail to compare using the U definition if they were copied using W definition. This occurs because records marked as deleted from the input file were not copied onto the output file and because zero-length W control words are not precisely duplicated.

For a logical comparison, the record manager gets records from the two files and compares just the data (without W control words, zero bytes, blocking control, etc.). The record types do not have to be the same. Thus, even though you have changed the logical structure of the data by copying to a different record type or block type, you can compare the data on the files as long as the copy did not add or delete any file delimiters (EOS or EOP). An exception is S records which can be compared only with S records. That is, if one file is S record type, both must be. (The restriction does not apply to Z records with C blocking.)

Comparison begins at the current position of each file and moves ahead record-by-record. In addition to comparing the contents of each pair of logical records, COMPARE checks file delimiters to see if they are the same level. They must match to produce a good compare and for the compare to continue. For example, if EOP status occurs on one file while EOS status occurs on another, the compare terminates.

COMPARING SECTIONS

Unless you specify otherwise, a compare will compare the contents of one section on one file with the contents of one section on another. If the file type does not support sections, the compare will terminate on an EOP or EOI.

COMPARE(lfn_1, lfn_2)

Using the preceding statement, one section will be compared. To specify more than one section, follow lfn_2 with a decimal count of the number of sections to be compared.

COMPARE(lfn_1, lfn_2, n)

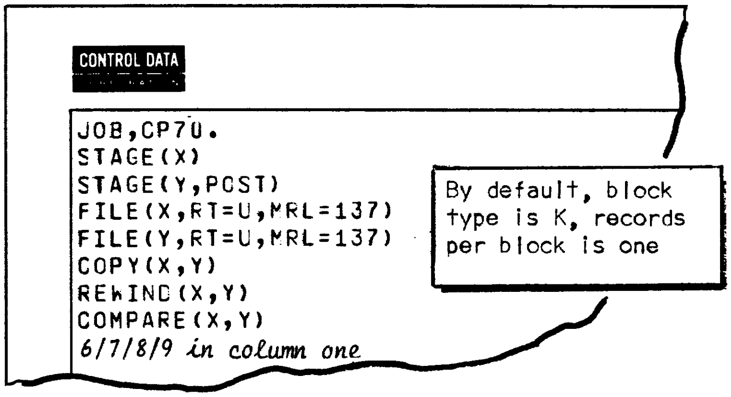
COMPARING PARTITIONS

The COMPARE statement includes a fourth parameter that allows you to specify that partitions rather than sections be compared; n then specifies the number of partitions. If n is omitted, its comma must still appear. Parameters on COMPARE are positionally dependent.

COMPARE(lfn_1, lfn_2, n, 17)

The 17 indicates partitions. If the file type does not support partitions, the comparison continues to EOI.

Example 10-12 illustrates a job that copies and compares a complete file. Both files are described as record type U, block type K. Each record/block is 137 characters.

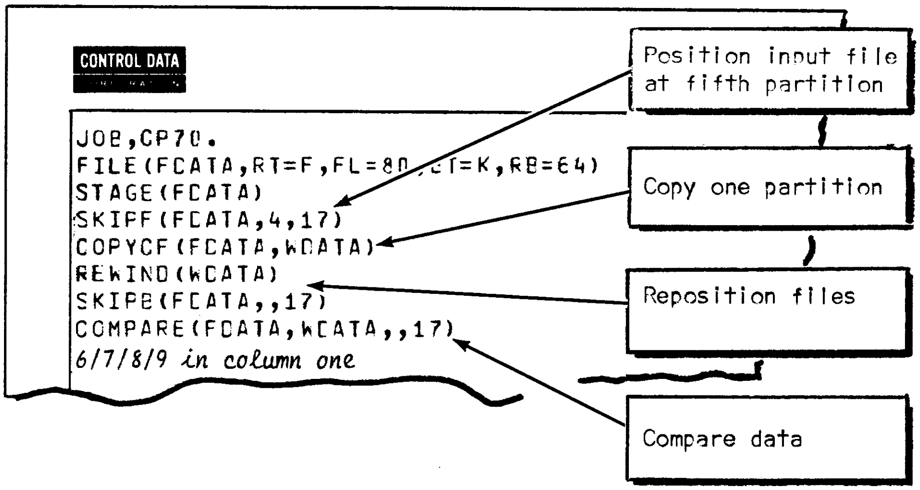


```
JOB,CP70.
STAGE(X)
STAGE(Y,POST)
FILE(X,RT=U,MRL=137)
FILE(Y,RT=U,MRL=137)
COPY(X,Y)
REWIND(X,Y)
COMPARE(X,Y)
6/7/8/9 in column one
```

By default, block type is K, records per block is one

Example 10-12. Copy and Compare Files

Example 10-13 illustrates a job that copies and compares the fifth partition on file FDATA. The input file contains K-blocked F records, the output file contains unblocked W records.



```
JOB,CP70.
FILE(FDATA,RT=F,FL=80,BT=K,RB=64)
STAGE(FDATA)
SKIPF(FDATA,4,17)
COPYCF(FDATA,WDATA)
REWIND(WDATA)
SKIPB(FDATA,,17)
COMPARE(FDATA,WDATA,,17)
6/7/8/9 in column one
```

Position input file at fifth partition

Copy one partition

Reposition files

Compare data

Example 10-13. Logical Compare of F Records and W Records

COMPARING S-RECORDS

When comparing S-type records, you can specify any level from 0 to 17_8 as the fourth parameter and the COMPARE will terminate when it has encountered the specified number of records that have a level number equal to or greater then the level specified.

ERROR RECORD COUNT

If your file contains a very large number of error records in one section or partition, or is not divided into sections or partitions, you can indicate the number of error records to be compared before termination through a decimal count (r parameter). Remember, the parameter is positionally dependent - the e parameter is described in the following text. The default for r is 30,000.

The r count will terminate the compare even if n sections, S records of level lev, or partitions have not been encountered.

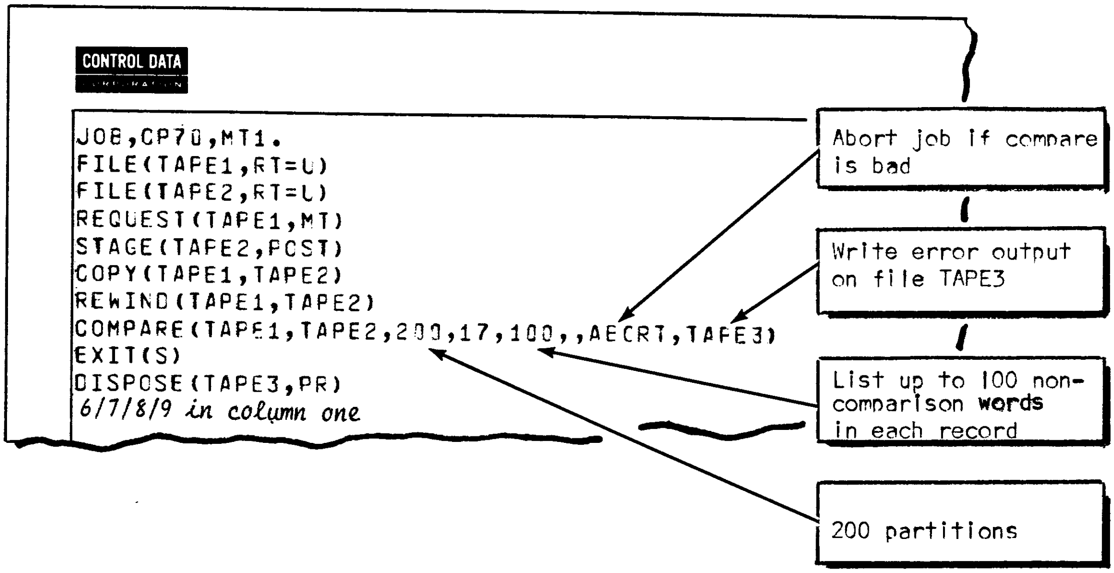
COMPARE(lfn$_1$, lfn$_2$n, lev, e, $\underline{r}$)

LIST CONTROL PARAMETERS

The previously mentioned COMPARE statements produce only messages in your dayfile, for example, UT070 GOOD COMPARE or UT071 BAD COMPARE. You can indicate on the COMPARE statement that you desire a listing of a specified number of noncomparison word pairs and a file to receive the noncomparison output through the e parameter and the lfn$_3$ parameter. e is expressed as a decimal number; the default is zero. lfn$_3$ is a file name; the default is OUTPUT. If you specify other than OUTPUT, you are responsible for having the file listed; for example, you can list it through a DISPOSE statement.

COMPARE(lfn$_1$, lfn$_2$, n, lev, $\underline{e}$, r, a, $\underline{\text{lfn}_3}$)

Example 10-14 illustrates a request for list output generated by a literal comparison of U record input and output.


CONTROL DATA
CORPORATION
Abort job if compare is bad
Write error output on file TAPE3
List up to 100 noncomparison words in each record
200 partitions

```
JOB,CP70,MT1.
FILE(TAPE1,RT=U)
FILE(TAPE2,RT=U)
REQUEST(TAPE1,MT)
STAGE(TAPE2,POST)
COPY(TAPE1,TAPE2)
REWIND(TAPE1,TAPE2)
COMPARE(TAPE1,TAPE2,200,17,100,,ABORT,TAPE3)
EXIT(S)
DISPOSE(TAPE3,PR)
6/7/8/9 in column one
```

Example 10-14. Literal Copy and Compare of Tapes Described as U Records

ABORT PARAMETER

Normally, a bad comparison produces only informative messages; the job is not terminated. If you wish to specify job termination on a bad comparison, enter a nonblank value for the a parameter.

COMPARE(lfn_1, lfn_2, n, lev, e, r, a, lfn_3)

Remember that the parameter is positionally dependent. Example 10-14 illustrates this feature. Notice that one of the tapes is on-line, the other is staged.

FILE LABELS 11

INTRODUCTION

As you learned earlier, a file is generally identified by a logical file name assigned when the file is created. In addition to logical file names, labels containing further identification can be associated with a file. The primary purpose of a label is to uniquely identify the file. Other uses depend on the type of label and the device on which the file is stored.

Currently, labels can be used only for magnetic tape files. All other files are considered unlabeled.

LABELS ON MAGNETIC TAPE

On magnetic tape, labels can be associated with each file recorded. These labels serve the following purposes:

1. Like the logical file name, the label can be used to identify a file. It also contains such information as the edition number and creation data to further differentiate one file from another in the system. Similar labels identify the volume (reel of tape) on which a labeled file resides.

2. A label can protect a file from accidental destruction by preventing a user from writing on the file until the expiration date specified on the label has elapsed.

USERS

The most predominant users of labeled files on tape are COBOL programmers. For this reason, the information and examples in this section are directed primarily toward them. Labels may also be used by those programming in other languages, such as FORTRAN and COMPASS. The procedure for labeling in FORTRAN is described in the FORTRAN Extended and FORTRAN RUN Reference Manuals.

STANDARD LABELS

Labels used most commonly for files processed under SCOPE 2 are written in the standard format. This format conforms to the American National Standards Institute (ANSI) Magnetic Tape Labels for Information Interchange X3.27-1969 Specifications. This standardization means that magnetic tape labels created in this format can be processed under many other computer systems. Standard labels are described further in Appendix C.

REQUESTING STANDARD LABELED TAPES

To specify that a label is to be generated on an output tape or that a label exists and is to be checked requires parameter specification on the STAGE or REQUEST statement for the file.

To specify creation of a new label, place N on the statement; to specify checking of an existing label, place E on the statement. If neither parameter is used, the file is assumed to be unlabeled. If E (existing label) is specified and the file is opened for output, a new label is created. N has no significance when prestaging.

```
               MT      N                                     MT       N,POST)
REQUEST(lfn,     ,...,  )          or      STAGE(lfn,          ,...,
               NT      E                                     NT       E)
```

Label writing (W or omitted) or reading (R) can, as an alternative, be supplied on a LABEL statement. This specification, if a LABEL statement is supplied, takes precedence over the N or E parameter on the STAGE/REQUEST statement. To specify creation of a new label, place W for write on the LABEL statement. For checking an existing label, place R for read on the statement. The statement must precede the first use of the file. Parameters after lfn describe label field values as explained in the following text.

```
           R
LABEL(lfn, ,...)
           W
```

PROVIDING STANDARD LABEL INFORMATION

The information used to generate or check the label is usually supplied through the source language program. If no information is provided by the user program or if the user wishes to override the information, he can supply a LABEL statement for the file.

NOTE

> The LABEL statement does not require the existence of a REQUEST or STAGE statement for the file. It can be used to label a blocked sequential mass storage file. However, permanent files at the 6000 Station cannot be labeled.

Specification of characters other than A through Z and 0 through 9 is possible by using $ delimiters. For the file identification field, for example, specifying L=value limits the characters in the string to A through Z and 0 through 9, but by using the form L=$value$, any of the 63 characters in the subset can be used. However, if a dollar sign is to be significant in the string, it must be represented as $$. To specify the character string *FILE%+FILE$*, delimit the string as follows in display code characters:

```
$*FILE%+FILE$$*$
```

Label Creation Information

The contents of label fields are generated as described in the following paragraphs using values supplied on the LABEL statement. An exception is the VSN field in the Volume Header Label, information for which is supplied on the STAGE or REQUEST statement.

File Identifier

If no file identifier is specified, the system uses the logical file name (lfn). To supply a file identifier, use the L=value parameter on the LABEL statement. The value is 1 to 17 alphanumeric characters. Remember to delimit characters other than A through Z and 0 through 9 with the $ delimiters.

Set Identification

If no set identification is specified, the system uses blanks. Specify a name for a multifile volume by using the parameter M=name, where name is 1 to 6 alphanumeric characters.

File Section Number

If no file section number is specified, the system uses 0001 for the first volume, 0002 for the second volume, etc. Setting this parameter on an output file would be highly irregular and would cause sequencing to begin at the specified value. The parameter that would reset the file section number is V=n, where n is 1 to 4 digits.

File Sequence Number

The file sequence number applies only for a multifile volume.

The system uses 0001 for the first file, 0002 for the second file, etc.,if no file sequence number is specified. Setting this parameter on an output file would be highly irregular and would cause sequencing to begin at the specified value. The parameter that would reset the file sequence is P=n, where n is 1 to 4 digits.

Generation Number

The default for the file generation number is 0001. Specify a new number through the parameter G=n, where n is 1 to 4 digits.

Generation Version Number

The default for the generation version number is 00. If you wish to use the field to designate successive iterations of a file, specify the E=n parameter where n is 00 to 99.

Creation Date

The default value for the creation data is the current date. Supplying some other date is unusual but can be done by supplying the parameter C=yyddd, where yyddd specifies the date in Julian format.

Expiration Date

The default date for the expiration date is the current date. A different date can be supplied as a number of days which will be converted into Julian form before it is placed in the label, or it can be supplied in Julian form. To supply a number, use the parameter T=n, where n is 1 to 4 digits. To supply a Julian date, use the parameter U=yyddd.

Example 11-1 illustrates a COBOL program that supplies label values for creation. Information for the label is obtained from the data card. Notice that the program sets the file identification field to LBTAPE. The date written field is zeroed, the reel number and edition number fields are set to 1, and the retention cycle field is 10. The user has overridden the file identification field by renaming the file NEWLIB and has set the retention cycle to 100 by specifying T=100 on the LABEL statement.

Label Checking Information

When reading a labeled input file, the user can supply information on a LABEL statement to be used for checking the label and for defining the initial volume to be read and the initial position on the volume currently mounted.

File Identifier

If no file identifier is specified, the file identifier is not checked. A file identifier can be supplied in the user program through the compiler language or COMPASS language program or can be specified on the LABEL statement in the form L=value. The value must match the character string used when the file was generated.

Set Identification

If no set identification is supplied by the user program or on the LABEL statement, no check is made. To specify the value on the LABEL statement, use M=name, where name must match the character string used when the file was created.

File Section Number

If no file section number is supplied by the user program or on the LABEL statement, processing of the file begins with volume 1. To specify a volume when you wish processing to begin at other than the first volume, use the V=n parameter on the LABEL statement. Remember, however, to also specify the correct beginning VSN on the corresponding STAGE or REQUEST statement. Both the file section number and the volume serial numbers must match those on the mounted tape.

File Sequence Number

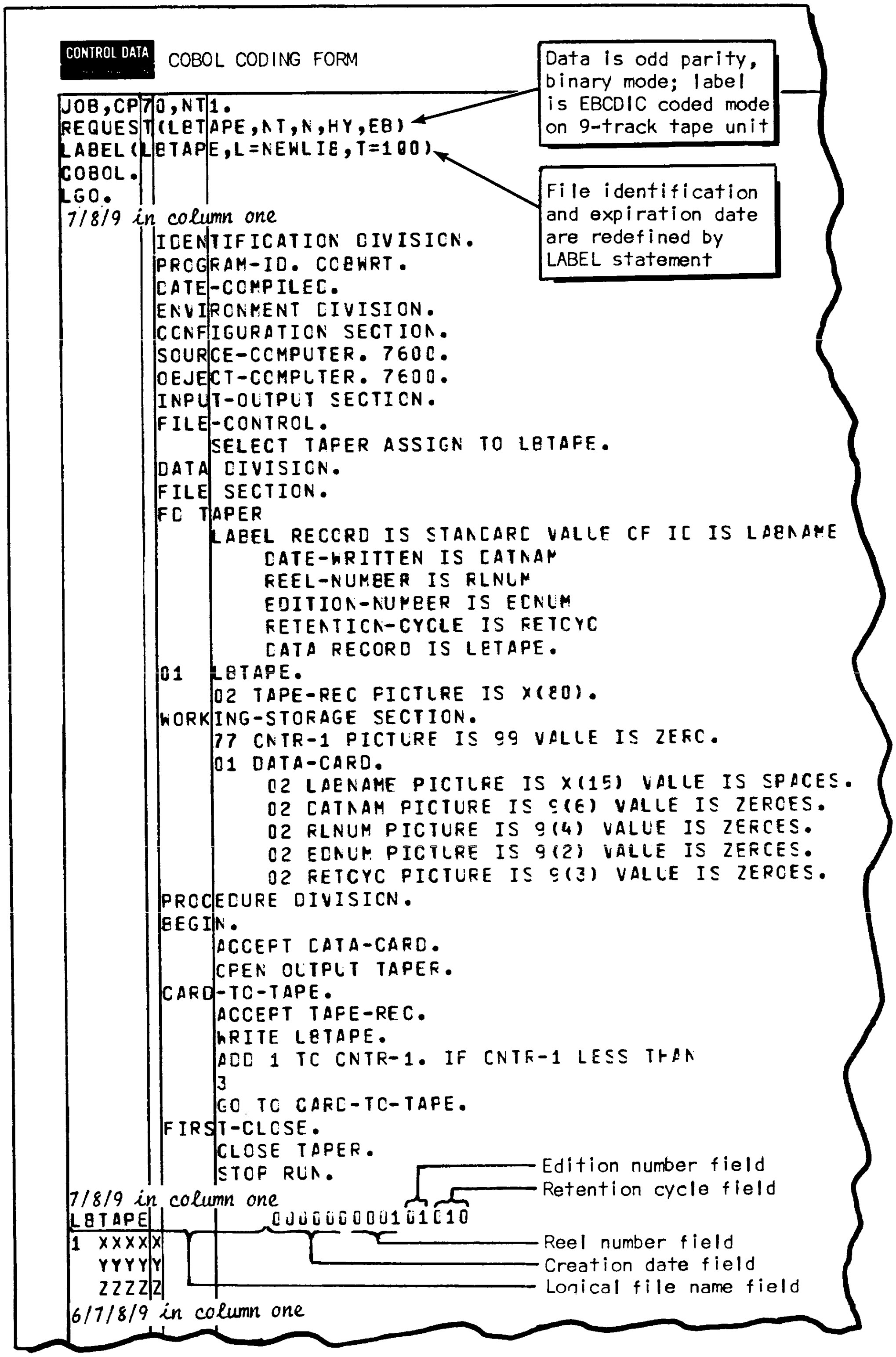
When no file sequence number is specified by the user program or on the LABEL statement, the first file on the volume of a multifile volume is processed. To specify that processing is to begin at other than the first file, use the P=n parameter on the LABEL statement.

Generation Number

In no generation number is specified by the user program or on a LABEL statement, no check is made of the field. To specify a generation number on the LABEL statement, use the G=n parameter where n must match the number specified when the file was generated.

Generation Version Number

The generation version number is always checked. If no number is specified for checking by the program or on a LABEL statement, the record manager uses 00. To specify a generation version number, use the E=n parameter where n must match the number specified when the file was generated.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70,NT1.
REQUEST(LBTAPE,NT,N,HY,EB)
LABEL(LBTAPE,L=NEWLIB,T=100)
COBOL.
LGO.
7/8/9 in column one
          IDENTIFICATION CIVISICN.
          PRCGRAM-ID. CCBWRT.
          CATE-COMPILEC.
          ENVIRONMENT CIVISION.
          CCNFIGURATICN SECTION.
          SOURCE-CCMPUTER. 760C.
          OBJECT-CCMPLTER. 7600.
          INPUT-OUTPUT SECTICN.
          FILE-CONTROL.
              SELECT TAPER ASSIGN TO LBTAPE.
          DATA CIVISICN.
          FILE SECTION.
          FC TAPER
              LABEL RECCRD IS STANCARC VALLE CF IC IS LABNAME
                  CATE-WRITTEN IS CATNAM
                  REEL-NUMBER IS RLNUM
                  EDITION-NUMBER IS ECNUM
                  RETENTICN-CYCLE IS RETCYC
                  CATA RECORD IS LBTAPE.
          01  LBTAPE.
              02 TAPE-REC PICTLRE IS X(80).
          WORKING-STORAGE SECTION.
              77 CNTR-1 PICTURE IS 99 VALLE IS ZERC.
              01 DATA-CARD.
                  02 LABNAME PICTLRE IS X(15) VALLE IS SPACES.
                  02 CATNAM PICTURE IS 9(6) VALLE IS ZERCES.
                  02 RLNUM PICTURE IS 9(4) VALUE IS ZERCES.
                  02 ECNUM PICTLRE IS 9(2) VALLE IS ZERCES.
                  02 RETCYC PICTURE IS 9(3) VALLE IS ZERCES.
          PROCEDURE DIVISICN.
          BEGIN.
              ACCEPT CATA-CARD.
              CPEN OLTPLT TAPER.
          CARD-TC-TAPE.
              ACCEPT TAPE-REC.
              WRITE LBTAPE.
              ADD 1 TC CNTR-1. IF CNTR-1 LESS THAN
              3
              GO TC CARC-TC-TAPE.
          FIRST-CLCSE.
              CLOSE TAPER.
              STOP RUN.
7/8/9 in column one
LBTAPE                000000000101010
1 XXXXX
  YYYYY
  ZZZZZ
6/7/8/9 in column one
```

Data is odd parity, binary mode; label is EBCDIC coded mode on 9-track tape unit

File identification and expiration date are redefined by LABEL statement

Edition number field

Retention cycle field

Reel number field

Creation date field

Logical file name field

Example 11-1. Using LABEL Statement for Label Generation With COBOL Program

Creation Date

If no creation date is specified by the user program or on a LABEL statement, no check is made of the field. Specification of a creation data for label checking is not conventional but can be specified through the C=yyddd parameter on the LABEL statement.

Expiration Date

If the T or U parameter is supplied for input, the expiration date is checked. If the file is opened for output, any existing label on the file is checked against the current date. If the date is unexpired, the tape cannot be written on without operator permission.

Other Fields

The record manager does not check any other fields. For example, it does not check the owner identification field or accessibility fields.

In Example 11-2, suppose that we want to use the labeled tape file created by the program in Example 11-1 as input to another COBOL program. For simplicity, let us assume that the sole purpose of the program is to print the contents of the file; the contents of the label will be checked when the file is opened. In most cases where labeled files are used, much more intricate or varied processing occurs; but the principles of referencing and checking labeled files remain the same. In this example, the tape created on-line is prestaged.

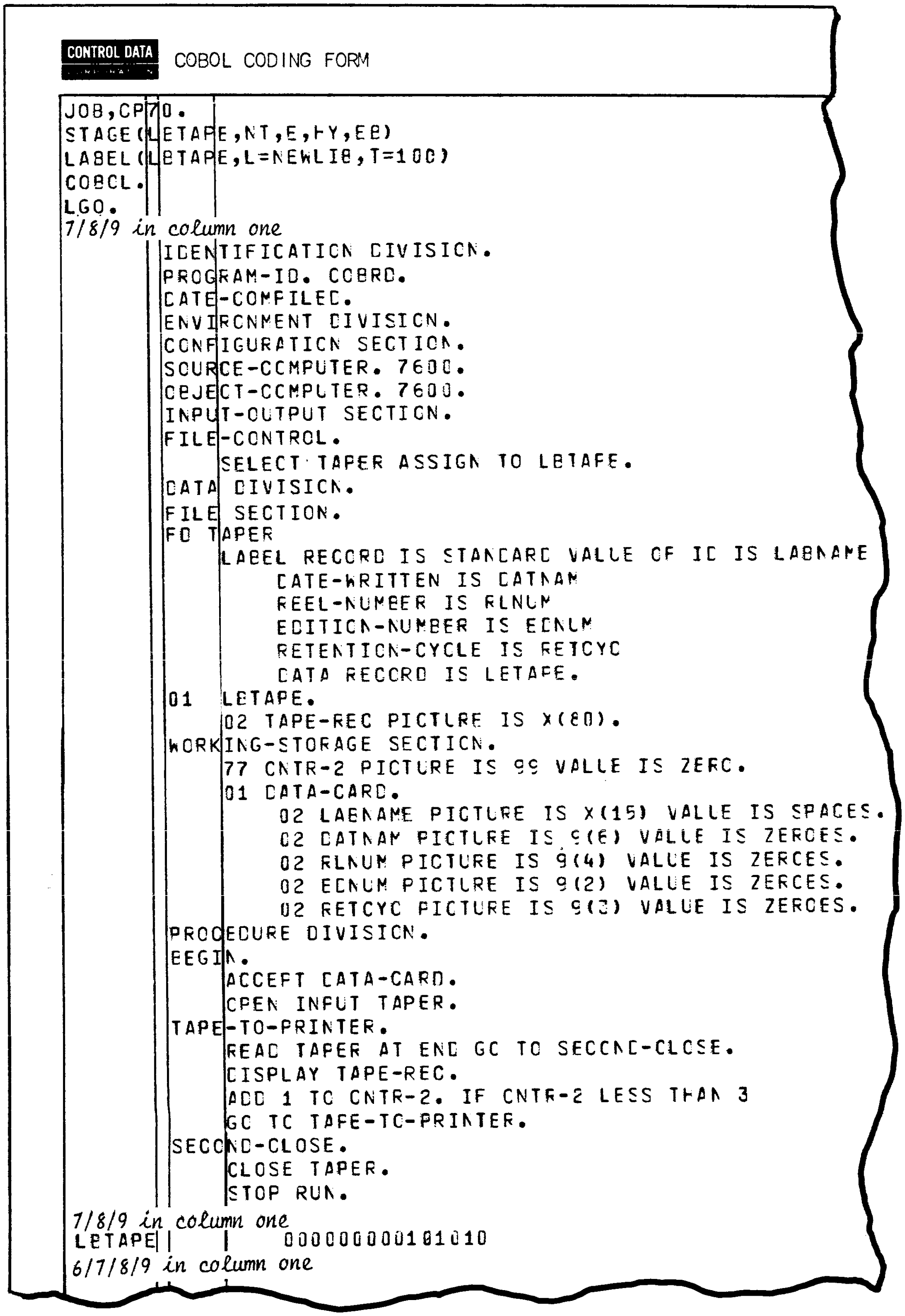
PROTECTION OF UNEXPIRED LABELED TAPES

Any output tape, in addition to requiring a write enable ring, cannot be written on without operator consent if it contains a label having an unexpired date. All tape drivers before writing on an output tape check the mounted tape for a label. If the tape is unlabeled, writing begins. If the tape is labeled and contains an unexpired date, the operator must respond by replacing the tape or by indicating that it can be written on before writing can begin.

COPYING LABELED TAPES

The copy routines (Section 10) can be used for copying labeled tapes as well as unlabeled tapes. If the input tape contains a label, the label is checked but is not copied to the output file. Copying begins after the HDR1 label and terminates before the EOF1 label. Labeling of the output file requires that the STAGE or REQUEST contain an N parameter or that the LABEL statement contain the W parameter. The output tape will be "blank-labeled" if the STAGE or REQUEST for the output tape specifies N and there is no accompanying LABEL statement. When the LABEL statement is present, the record manager uses it as the source of label information. When labeling is specified, the tape to be copied must not be a multifile volume. The copy routines cannot handle more than one labeled file on a volume.

If you want to copy the labels as well as the data on a file, or if you want to copy a multifile volume, you must declare both the input file and output file as unlabeled. In this case, the labels are treated as partitions of data. No label checking takes place. When using this technique, both files must be defined as record type U, block type K, and both labels and data must be recorded in even parity, coded mode.



CONTROL DATA COBOL CODING FORM

```
JOB,CP70.
STAGE(LBTAPE,NT,E,HY,EB)
LABEL(LBTAPE,L=NEWLIB,T=100)
COBOL.
LGO.
7/8/9 in column one
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COBRD.
       DATE-COMPILED.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. 7600.
       OBJECT-COMPUTER. 7600.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT TAPER ASSIGN TO LBTAPE.
       DATA DIVISION.
       FILE SECTION.
       FD TAPER
           LABEL RECORD IS STANDARD VALUE OF ID IS LABNAME
               DATE-WRITTEN IS DATNAM
               REEL-NUMBER IS RLNUM
               EDITION-NUMBER IS EDNUM
               RETENTION-CYCLE IS RETCYC
               DATA RECORD IS LBTAPE.
       01  LBTAPE.
           02 TAPE-REC PICTURE IS X(80).
       WORKING-STORAGE SECTION.
           77 CNTR-2 PICTURE IS 99 VALUE IS ZERO.
           01 DATA-CARD.
               02 LABNAME PICTURE IS X(15) VALUE IS SPACES.
               02 DATNAM PICTURE IS 9(6) VALUE IS ZEROES.
               02 RLNUM PICTURE IS 9(4) VALUE IS ZEROES.
               02 EDNUM PICTURE IS 9(2) VALUE IS ZEROES.
               02 RETCYC PICTURE IS 9(3) VALUE IS ZEROES.
       PROCEDURE DIVISION.
       BEGIN.
           ACCEPT DATA-CARD.
           OPEN INPUT TAPER.
       TAPE-TO-PRINTER.
           READ TAPER AT END GO TO SECOND-CLOSE.
           DISPLAY TAPE-REC.
           ADD 1 TO CNTR-2. IF CNTR-2 LESS THAN 3
           GO TO TAPE-TO-PRINTER.
       SECOND-CLOSE.
           CLOSE TAPER.
           STOP RUN.
7/8/9 in column one
LBTAPE          000000000101010
6/7/8/9 in column one
```

Example 11-2. Using LABEL Statement for Label Checking With COBOL Program

LABEL DENSITY

For an output tape, labels and data are always written at the same density. For a 9-track input tape, labels and data are always read at the same density because the 9-track tape units allow density selection at load point only. For a 7-track input tape, labels and data need not be at the same density. A tape driver attempts to read a label at all densities until it performs a successful read. It then reads the data at the density specified by the user.

For SCOPE 3.4, the LABEL statement includes a label density parameter. This parameter is not recognized by SCOPE 2.0.

LABEL PARITY AND CHARACTER CONVERSION

All labels are recorded in coded mode (with character conversion), regardless of the presence of the CM=YES parameter. CM=YES applies to data only. For 7-track tapes, labels are written (or when reading assumed to be written) in External BCD.

For 9-track tapes, the character set used is determined by whether or not EB is specified on the STAGE or REQUEST statement. If you recall the description of 9-track character conversion in Section 6, EB used in conjunction with CM=YES caused the data to be converted to EBCDIC rather than ASCII. For labels, the parameters work differently.

To specify that data on a 9-track tape is to be binary mode but that labels are to be in EBCDIC rather than ASCII (the default character set), specify CM=NO or omit the parameter from the FILE statement and specify EB on the FILE or STAGE statement. To specify that data and label are to be in coded mode, include the CM=YES parameter on the FILE statement. Similarly, specifying US (or omitting US) will effect conversion to or from ASCII. On input, any lower case letter is converted to upper case. Any other character not in the 63-character subset is interpreted as a blank.

ANALYTICAL AIDS 12

Hopefully, your program will run perfectly the first time it is compiled and executed. If your program should happen to terminate prematurely, however, you will want to make use of analytical aids available through the operating system. Options include the use of EXIT statements to allow a job to resume processing despite the occurrence of abort conditions, specifying that certain types of errors or conditions be ignored rather than resulting in job termination, generation of core memory dumps for analysis, and generation of listings of files for determining possible causes of program termination.

This section deals primarily with the use of SCOPE 2 features.

In addition to the listing of instructions in your program, which is automatically printed in source (compiler or assembler) language, other listings of these instructions in object code can be obtained as a compiler option. Compiler options are used primarily by experienced programmers to isolate specific errors in object-coded instructions.

The TRAP program is another aid available to you. It has two options, TRACK and FRAME. The TRACK option provides a printed analysis of program instructions in terms of storage references, operand references, and arithmetic register use over a user-specified range of instructions. The FRAME option provides printouts of selected areas of storage at the time specified instructions are executed. This routine gives you a picture of a small portion of SCM or LCM at any given moment. The TRAP program and its options (TRACK and FRAME) are not described in this guide but are described in the Loader Reference Manual.

CONTROLLING YOUR JOB WITH EXIT STATEMENTS

If an abnormal termination (abort) condition occurs during loading or execution of one of your job steps, all need not be lost. SCOPE allows you to specify through EXIT statements that you want control statement processing to resume despite an abort condition.

When an abort condition occurs in your job, SCOPE skips control statements until it encounters an EXIT statement. Figure 12-1 and Table 12-1 show whether SCOPE aborts the job when it encounters the EXIT statement or whether SCOPE resumes control statement processing (continues the job) with the statement following the EXIT statement. Statements following the EXIT statement might dump the SCM image of your job, save files, catalog or purge a permanent file, request an entirely different program sequence, etc.

The allowable forms of the EXIT statement are as follows:

EXIT.	Normal abort processing
EXIT(S)	Selective processing
EXIT(C)	Conditional processing
EXIT(U)	Unconditional processing

SCOPE 3.4 does not recognize EXIT(C) and EXIT(U) statements.

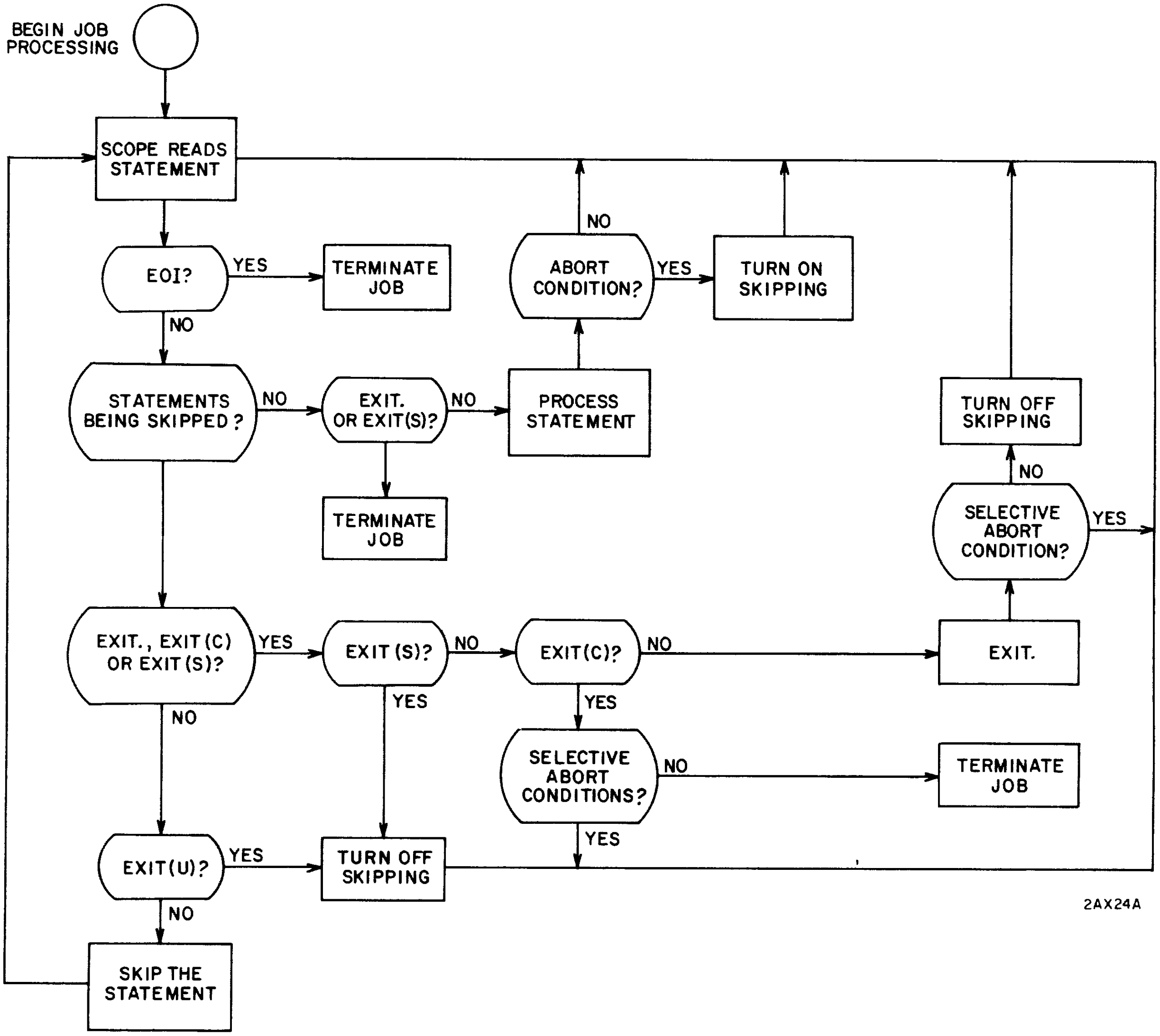
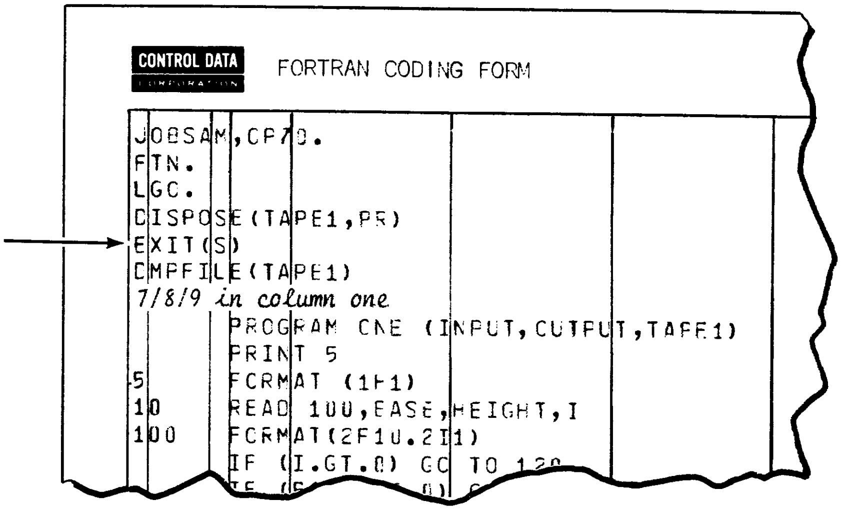


Figure 12-1. Flow Chart of EXIT Processing

TABLE 12-1. EXIT STATEMENT PROCESSING

	EXIT.	EXIT(S)	EXIT(U)	EXIT(C)
Normal job termination; no errors encountered	End job	End job	Continue job	Continue job
Abort condition resulting from: Program execution error (not overridden through MODE statement) Operator has issued DROP command at station Time, mass storage, or LCM limit exceeded Irrecoverable mass storage or tape parity error Control statement syntax error System aborts, for example, record manager errors	Continue job	Continue job	Continue job	End job
Selective abort condition resulting from: Compilation errors Number of on-line tapes scheduled has been exceeded SCM/LCM parity errors	Skip until end-of-job or EXIT(S)	Continue job	Skip until end-of-job or EXIT(S)	Skip until end-of-job or EXIT(S)
Irrecoverable abort condition resulting from operator issuing RERUN or KILL command	End job	End job	End job	End job

In Example 12-1, the EXIT statement allows job processing to continue following abnormal job termination. In this example, the DMPFILE statement requests a printout of the contents of file TAPE1.

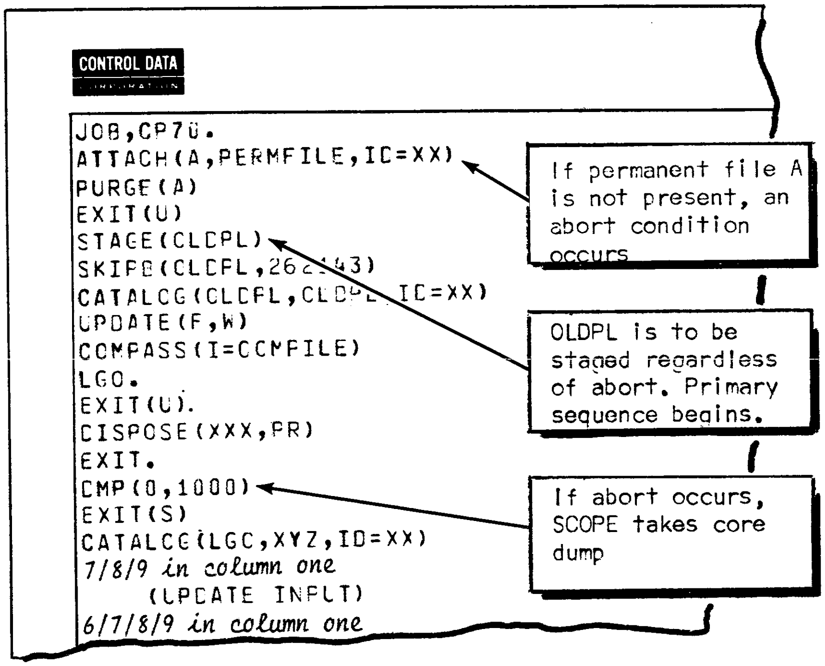


CONTROL DATA CORPORATION FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
LGC.
DISPOSE(TAPE1,PR)
EXIT(S)
DMPFILE(TAPE1)
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
```

Example 12-1. Selective Exit Processing

Example 12-2 illustrates a job that contains a combination of EXIT statements. The first thing the user wishes to do is to purge file A if it is still a cycle of a permanent file. If the file has already been purged, the attempt to attach it will cause an abort condition. The EXIT(U) allows processing to continue with the STAGE(OLDPL) statement. Following execution of the object program on LGO, a second unconditional EXIT statement assures that the contents of file XXX are printed regardless of whether or not the object program abnormally terminates. Then, if an abort condition does occur, SCOPE takes an SCM dump of the first 1000_8 words. Finally, if a selective condition occurs, the EXIT(S) assures that the LGO file is saved as a permanent file.



```
JOB,CP70.
ATTACH(A,PERMFILE,ID=XX)
PURGE(A)
EXIT(U)
STAGE(OLDPL)
SKIPB(OLDPL,262143)
CATALOG(OLDPL,OLDPL,ID=XX)
UPDATE(F,W)
COMPASS(I=COMFILE)
LGO.
EXIT(U).
DISPOSE(XXX,PR)
EXIT.
DMP(0,1000)
EXIT(S)
CATALOG(LGO,XYZ,ID=XX)
7/8/9 in column one
     (UPDATE INPUT)
6/7/8/9 in column one
```

If permanent file A is not present, an abort condition occurs

OLDPL is to be staged regardless of abort. Primary sequence begins.

If abort occurs, SCOPE takes core dump

Example 12-2. Combination of Exit Paths

SETTING ERROR CONDITIONS

SCOPE 2 allows you to override some of the error conditions that can occur during execution of your object program. The conditions over which you have some control are the following:

SCM direct range error	This condition results from an attempt to reference an operand outside your SCM field length.
Overflow error	This condition occurs if the floating point functional unit generated an infinite operand.
Indefinite operand error	This condition occurs if the floating point unit generated an indefinite operand.
Underflow error	This condition occurs if the floating point unit generated a smaller operand than it is possible to represent in floating point notation.

The underflow error condition is not detected on a 6000 Series or CDC CYBER 70/Model 72, 73, or 74 Computer System. For compatibility with SCOPE 3.4, the installation option for program error mode is usually set so that your program is abnormally terminated if any of the conditions other than underflow occurs. This is the same as MODE (7). In addition, underflow occurs often enough so that generally it should not be selected as an abort condition.

Error conditions over which you have no control and which always result in job termination are the following:

LCM direct range error — This condition results from an attempt to reference an operand outside the LCM field length.

SCM or LCM block range error — These conditions result from a block copy that references an address outside the respective field length.

Program range error — This condition results from an attempt to execute a word containing zero.

NOTE

> For any range error - including the SCM direct range error - the program takes the error exit. In the case of the SCM direct range error, however, error processing by SCOPE allows the error to be ignored through the use of the MODE statement.

If you want to continue program execution despite the occurrence of one of the error conditions (except where noted), or if you want to terminate execution if an underflow error occurs, you can place a MODE statement in the control statement section of your job deck (see Table 12-2 for MODE parameters). The MODE statement affects all subsequent job steps until another MODE statement is encountered or until job end.

TABLE 12-2. MODE STATEMENT PARAMETERS

MODE Parameter	Error Condition Causing Job Termination			
	Underflow (Bit 2^3)	Indefinite (Bit 2^2)	Overflow (Bit 2^1)	SCM Direct Range (Bit 2^0)
0				
1				X
2			X	
3			X	X
4		X		
5		X		X
6		X	X	
7		X	X	X
10	X			
11	X			X
12	X		X	
13	X		X	X
14	X	X		
15	X	X		X
16	X	X	X	
17	X	X	X	X

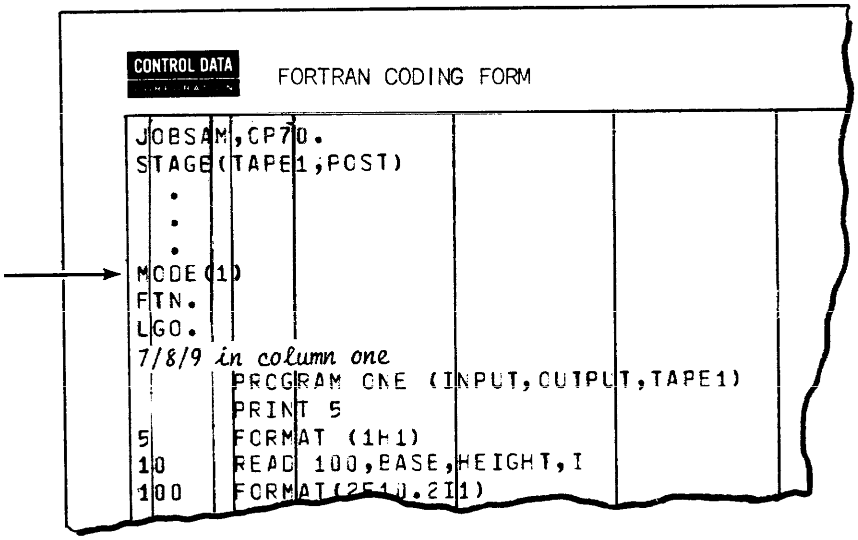
The format of the MODE statement is as follows:

```
MODE(n)
```

Example 12-3 illustrates the use of a MODE(1) statement to designate that job termination is to occur only if an SCM direct range error occurs. MODE statements are used primarily as an aid while debugging your program.

NOTE

> The MODE(0) statement is usually used as a last resort when debugging a program that does not run to completion for no apparent reason. The MODE(0) statement forces the job to run to completion.



```
JOBSAM,CP70.
STAGE(TAPE1;PCST)
   .
   .
   .
MODE(1)
FTN.
LGO.
7/8/9 in column one
      PRCGRAM CNE (INPUT,CUTPUT,TAPE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FCRMAT(2F10.2I1)
```

Example 12-3. Using the MODE Statement

SETTING LOADER ERROR OPTIONS

SCOPE 2 allows you to override some of the error conditions that can occur during loading of your object program. When the loader detects an error during the load sequence, it takes the following action depending on the severity of the error.

Terminal errors — Immediately terminate loading and terminate the job

Fatal errors — Complete the load sequence and then terminate the job without executing the loaded program

Nonfatal errors — Continue normal processing of the job but issue an informative message

The LDSET ERR option lets you specify job termination for terminal errors only, or for any error, despite its severity. The SCOPE 2 loader also detects normal conditions that result in informative messages but do not cause a job to terminate. Such conditions are not considered error conditions.

Placing the following control statement in the load sequence causes the loader to issue an error message and continue job processing for either fatal or nonfatal errors. A terminal error causes immediate job termination.

```
LDSET(ERR=NONE)
```

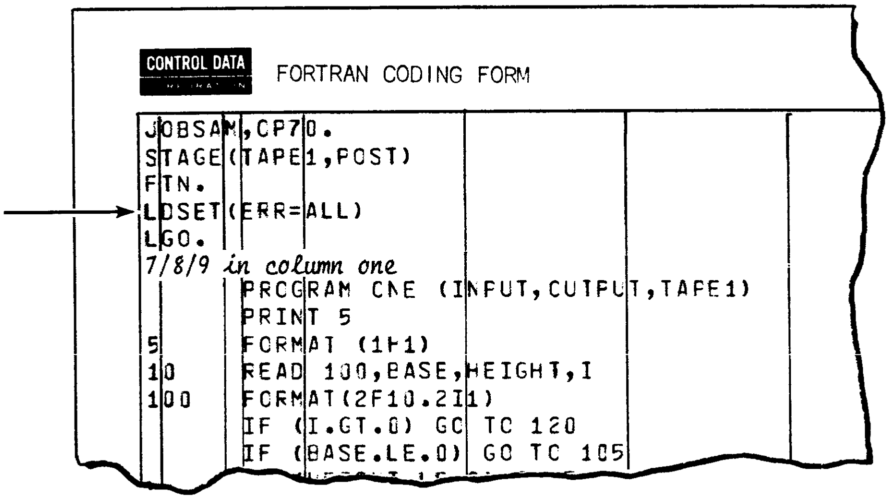
The following option causes the loader to terminate the job even for errors that are normally nonfatal.

```
LDSET(ERR=ALL)
```

The following ERR option is equivalent to the normal default mode.

```
LDSET(ERR=FATAL)
```

In Example 12-4, the LDSET (ERR=ALL) causes execution of LGO to be prevented even though the loader detected an error that was normally nonfatal, for example, if the loader failed to satisfy all externals.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
STAGE(TAPE1,POST)
FTN.
LDSET(ERR=ALL)
LGO.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO TO 105
```

Example 12-4. Requesting No Program Execution For Any Loader Error

OBTAINING PROGRAM DUMPS

If an error prematurely terminates your job, SCOPE 2 automatically generates a Standard Dump of SCM and writes it on the OUTPUT file. This dump (Figure 12-2) includes:

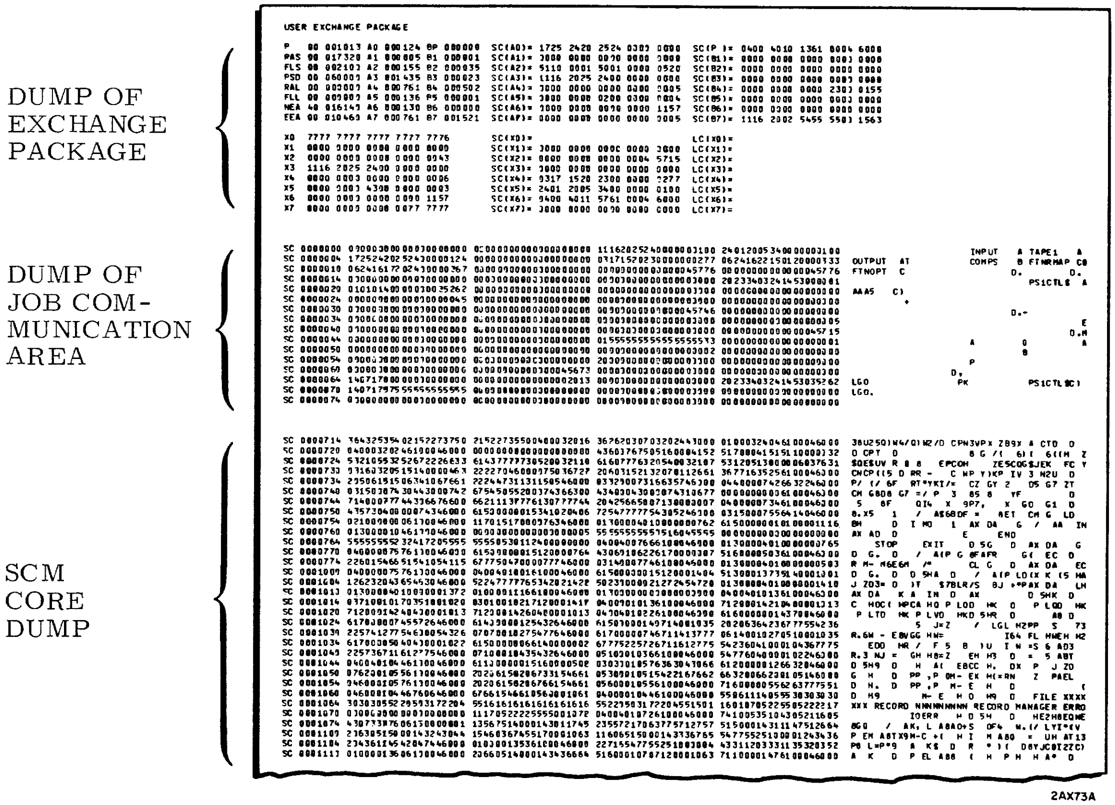
- Contents of the exchange package used for communication between your job and SCOPE, including the current contents of all arithmetic registers in the central processor unit

- Assuming that the arithmetic registers contain addresses, the dump includes a listing of the contents of the addresses referenced in the registers.

- Contents of the first 210_8 locations in the SCM field (RAS+0 to RAS+207). The first 100_8 locations are the job communications area.

- Contents of the 77_8 locations immediately preceding the location at which execution abnormally terminated and the 77_8 locations immediately subsequent to the termination point, as well as the contents of the termination address

Figure 12-2 illustrates a sample standard dump.


DUMP OF EXCHANGE PACKAGE
DUMP OF JOB COMMUNICATION AREA
SCM CORE DUMP
2AX73A

```
USER EXCHANGE PACKAGE

P   00 001013 A0 000124 B0 000000  SC(A0)= 1725 2420 2524 0303 0000  SC(P )= 0400 4010 1361 0004 6000
RAS 00 017320 A1 000005 B1 000001  SC(A1)= 0000 0000 0070 0000 0000  SC(B1)= 0000 0000 0000 0003 0000
FLS 00 002103 A2 000155 B2 000035  SC(A2)= 5110 0001 5001 0000 0520  SC(B2)= 0000 0000 0000 0000 0000
PSD 00 060003 A3 001435 B3 000023  SC(A3)= 1116 2025 2400 0000 0000  SC(B3)= 0000 0000 0000 0003 0000
RAL 00 000003 A4 000761 B4 000502  SC(A4)= 0000 0000 0000 0000 0005  SC(B4)= 0000 0000 0000 2303 0155
FLL 00 000003 A5 000136 B5 000001  SC(A5)= 0000 0000 0200 0000 0004  SC(B5)= 0000 0000 0000 0003 0000
NEA 40 016143 A6 000130 B6 000000  SC(A6)= 0000 0000 0070 0000 1157  SC(B6)= 0000 0300 0000 0000 0000
EEA 00 010463 A7 000761 B7 001521  SC(A7)= 0000 0000 0000 0000 0005  SC(B7)= 1116 2002 5455 5503 1563

X0  7777 7777 7777 7777 7776       SC(X0)=                           LC(X0)=
X1  0000 0000 0000 0000 0000       SC(X1)= 0000 0000 000C 0000 0000  LC(X1)=
X2  0000 0000 0000 0000 0043       SC(X2)= 0000 0000 0000 0004 5715  LC(X2)=
X3  1116 2025 2400 0000 0000       SC(X3)= 0000 0000 0000 0000 0000  LC(X3)=
X4  0000 0003 0000 0000 0006       SC(X4)= 0317 1520 2300 0000 0277  LC(X4)=
X5  0000 0003 4300 0000 0003       SC(X5)= 2401 2005 3400 0000 0100  LC(X5)=
X6  0000 0003 0000 0090 1157       SC(X6)= 0400 4011 5761 0004 6000  LC(X6)=
X7  0000 0003 0000 0077 7777       SC(X7)= 0000 0000 0000 0000 0000  LC(X7)=


SC 0000000 00000000000000000000 00000000000000000000 11162025240000003100 24012005340000003100                  INPUT   A TAPE1   A
SC 0000004 17252420252400000124 00000000000000000000 03171520230000000277 06241622150120000333 OUTPUT   AT      COMPS   B FTNRMAP C0
SC 0000010 06241617202400000367 00000000000000000000 00000000000000045776 00000000000000045776 FTNOPT   C               D.        D.
SC 0000014 00000000000000000000 00000000000000000000 00000000000000003000 20233403241453000001                              PS1CTL$  A
SC 0000020 01010140000000035262 00000000000000000000 00000000000000003000 00006000000000000000 AAA5    C)
SC 0000024 00000000000000000045 00000000000000000000 00000000000000003000 00000000000000000300           +
SC 0000030 00000000000000000000 00000000000000000000 00000000000000045746 00000000000000000000
SC 0000034 00000000000000000000 00000000000000000000 00000000000000003000 00000000000000000305                         D.-
SC 0000040 00000000000000000000 00000000000000000000 00000000000000003000 00000000000000045715                                   E
                                                                                                                                 D.M
SC 0000044 00000000000000000000 00000000000000000000 01555555555555555533 00000000000000000001                 A        0        A
SC 0000050 00000000000000000000 00000000000000000000 00000000000000000002 00000000000000000000                          0
SC 0000054 00000000000000000000 00000000000000000000 20000000000000003000 00000000000000000300                 P
SC 0000060 00000000000000000000 00000000000000045673 00000000000000003000 00000000000000000300               D,
SC 0000064 14071700000000000000 00000000000000002013 00000000000000003000 20233403241453035262 LGO            PK          PS1CTL$C)
SC 0000070 14071757555555555555 00000000000000000000 00000000000000003000 00000000000000000000 LGO.
SC 0000074 00000000000000000000 00000000000000000000 00000000000000003000 00000000000000000000


SC 0000714 36432535402152273750 21522735500400032016 36762030703202443000 01000324046100046000 36U25Q)N4/Q)N2/D CPN3VPX Z89X A CTO D
SC 0000720 04000320246100046000 00000000000000000000 43603767505160004152 51700041515110000332 D CPT  D        8 G /(  6)(  6((H  Z
SC 0000724 53210553252672226633 61437777305200032110 61607776320540032107 53120513000006037631 $OE$UV R 8 8   EPCOH    ZESCOG$JEK  FC Y
SC 0000730 03160320515140000463 22227046000075036727 20603152132070112661 36771635256100046300 CNCP((5 D RR -   C NP Y)KP IV 3 N2U  D
SC 0000734 23506151506341067661 22244731131150546000 03325007316635746300 04400007426632246300 P/ (/ 6F  RT*YKI/=  CZ GY 2    D5 G7 ZT
SC 0000740 03150307430443000742 67545055200374366300 43400043000074310677 00000000006100046300 CM G8D8 G7 =/ P  3  85 8   YF        D
SC 0000744 71400077744336676600 66211137776130777744 20425665007130000007 04000007346100046300  5   6F    QI4  X  9P7,   X  G0  G1  D
SC 0000750 43573040000074346000 61530000015341020406 72547777754305246300 03150007556414046000 8.X5   1    /  A$68DF =   8ET  CM G  LD
SC 0000754 02100000006130046000 11701517000376346000 01300004010000000762 61500000010100001116 BH     D  I N0   1  AX DA   G  /  AA  IN
SC 0000760 01300010461730046000 00000000000000000005 55555555555516045555 00000000000000000000 AX AD  D            E     END
SC 0000764 55555555232417205555 55550530112400000000 04004007666100046000 01300004010000000765    STOP     EXIT    D 5G   D  AX DA   G
SC 0000770 04000007576130046030 61530000151200000764 43060106226170000307 51600005036100046300 D G.  D    /  A(P G 0FAFR     G(  EC  D
SC 0000774 22601546651541054115 67775047000077746000 03140007746100046000 01300004010000000503 R M- M6E6M  /*       CL G   D  AX DA   EC
SC 0001000 04000007576130046000 04004010016100046000 61500000015120001404 51300013735140001001 D G.  D  D 5HA  D   /  A(P LD(X K (5 MA
SC 0001004 12623204365453046000 52247777765342021422 50231000021272454720 01300004010000001410 J Z03= D  )T   $7BLR/S  8J +*PAX DA   LH
SC 0001010 01300004010030001372 01000011166100046000 01300000000000003000 04004010136100046300 AX DA    K A  IN  D  AX         D 5HK  D
SC 0001014 03710010170351001020 03010010217120001417 04007010136100046000 71200014210400001313 C  HOC( HPCA HO P LOD  HK   0   P LOD  HK
SC 0001020 71200014240430001013 71200014260400001013 04304010226100046300 61600000014370046000  P LTD  HK P LVO  HKD 5HR  0       A0 D
SC 0001024 61700000745572646000 61430000125432646000 61500000140714001035 20206364236777554236            5  J=Z   /  LGL H2PP  S   73
SC 0001030 22574127754630054326 07070010275477646000 61700000746711413777 06140010270510001035 R.6W - E8VGG HW=          I64 FL HWEH H2
SC 0001034 61700005040430001022 61500000066140000002 67775225726711612775 54236041000104367775    ED0  HR /  F 5  8  )U  I W =S 6 AD3
SC 0001040 22573671161277546000 07100010435432646000 05100010366100046000 54776040000102246300 R.3 NJ =  GH H8=Z   EH H3  D  =  5 ABT
SC 0001044 04004010446130046000 61100000015160000502 03030010576363047066 61200000126632046000 D 5H9  D   H  A(  E8CC H,  DX  P  J Z0
SC 0001050 07620010556130046000 20206150206733154661 05300010515422167662 66320066230105146000 G  H   D  PP ,P 0M- EX H(=RN   Z  PAEL
SC 0001054 04600010576130046000 20206150206766154661 05600010556100046000 71600000556263777551 D  H.  D  PP ,P  M- E  H   D           (
SC 0001060 04600010446760646000 67661546610560001061 04000010446100046000 55861114055530303030 D  H9        M- E  H 0  H9  D  FILE XXXX
SC 0001064 30303055220553172204 55161616161616161616 55227503172204551501 16010705225505222217 XXX RECORD NNNNNNNN RECORD MANAGER ERRO
SC 0001070 00000000000000000000 11170522225555001072 04004010726100046000 74100535104305211605               IOERR  H D 5H  D  HE2H8EQNE
SC 0001074 43073376061300000001 13567514000143011745 23557217063775712757 51500014311147512664 8G0  /  AK, L A8AO+S  OF4  W.(/ LYI*(V
SC 0001100 23630515000143243044 15460367455170001063 11606515000143336765 54775525100001243436 P EM A8TX9M-C +(  H I  M A80  =  UH AT13
SC 0001104 23436112454204744600 01030013536100046000 22715547755251003004 43311203331135320352 P0 L=P*9  A  K$  D  R  * )(  D8YJC0I2ZC)
SC 0001110 01000013606130046000 23660514000143436664 51600010707120001063 71100001476100046000 A  K   D  P EL A88  (  H  P H  H A*  D
```

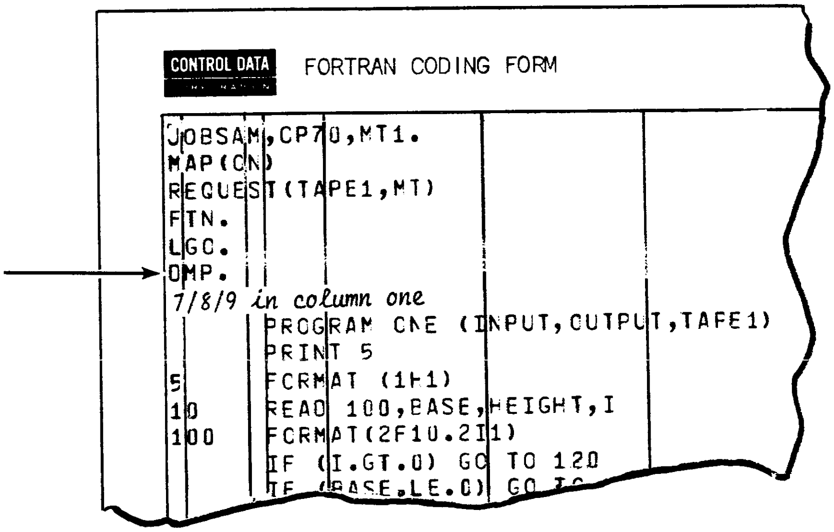
Figure 12-2. Standard Dump

REQUESTING A STANDARD DUMP

If you want a standard dump even if no error has occurred, you can request it through the DMP statement.

```
DMP.
```

As shown in Example 12-5, the request is generally written after the request that executes your program.



CONTROL DATA
FORTRAN CODING FORM

```
JOBSAM,CP70,MT1.
MAP(ON)
REQUEST(TAPE1,MT)
FTN.
LGO.
DMP.
7/8/9 in column one
      PROGRAM ONE (INPUT,OUTPUT,TAPE1)
      PRINT 5
5     FORMAT (1H1)
10    READ 100,BASE,HEIGHT,I
100   FORMAT(2F10.2I1)
      IF (I.GT.0) GO TO 120
      IF (BASE.LE.0) GO T
```

Example 12-5. Request for Standard Dump

REQUESTING SCM DUMPS

Depending on the nature of your job, you may want a dump to indicate the contents of locations other than those normally shown on a standard dump. You can specify more or fewer locations to be printed. For example, you can request a dump that will show the contents of all words from the beginning of your SCM field (RAS) to any specific address beyond RAS. To dump from RAS to a specified address, use the statement

```
DMP(lwa)
```

where lwa is the octal address of the last location you want listed. You can also specify the beginning relative address as well as the ending relative address for the dump. To do so, use the following form of the DMP statement:

```
DMP(fwa,lwa)
```

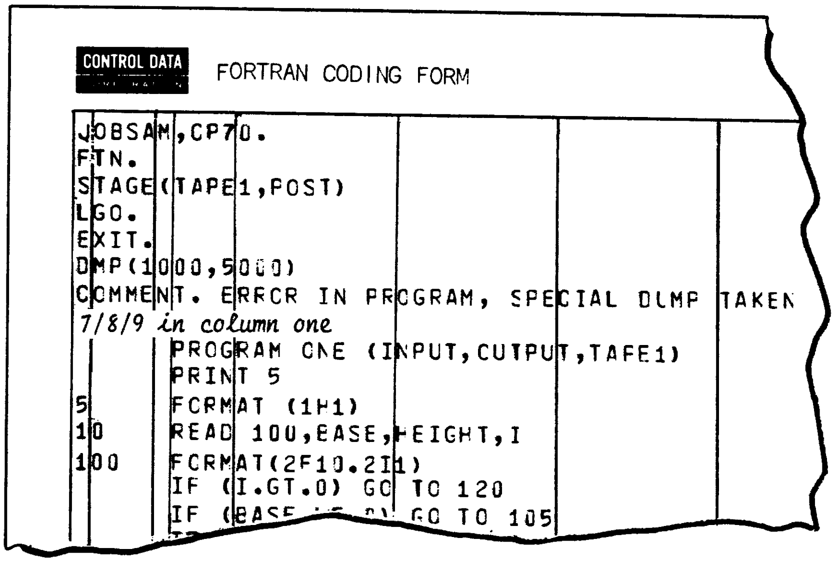
Thus, if you wished a dump of locations 200$_8$ through 500$_8$, you would use

```
DMP(200,500)
```

You can place the DMP statement after an EXIT statement if you wish the dump taken following abnormal job termination. Example 12-6 illustrates a job in which an SCM dump is generated during EXIT processing.

If you should happen to specify fwa the same as lwa, the system generates a dump as if you had simply specified DMP.

Unlike 6000 SCOPE, 7000 SCOPE 2 does not provide a means of obtaining a dump of SCM using absolute rather than relative addresses.



CONTROL DATA FORTRAN CODING FORM

```
JOBSAM,CP70.
FTN.
STAGE(TAPE1,POST)
LGO.
EXIT.
DMP(1000,5000)
COMMENT. ERRCR IN PRCGRAM, SPECIAL DLMP TAKEN
7/8/9 in column one
      PROGRAM CNE (INPUT,CUTPUT,TAFE1)
      PRINT 5
5     FCRMAT (1H1)
10    READ 100,EASE,HEIGHT,I
100   FCRMAT(2F10.2I1)
      IF (I.GT.0) GC TC 120
      IF (EASE         GO TO 105
```

Example 12-6. SCM Dump Taken Within An Exit Path

REQUESTING LCM DUMPS

The standard dump does not include a listing of any of the contents of the user LCM field. SCOPE 2 provides two control statements, DMPL and DMPECS, which you can use to obtain listings of locations in LCM. Except for statement name, the control statements are identical.

To request a dump that will show the contents of all words from the beginning of your LCM field (RAL) to any specific address beyond RAL, use one of the following statements.

```
DMPL(lwa)          or          DMPECS(lwa)
```

where lwa is the octal address of the last LCM location you want listed.

You can also specify the beginning relative address as well as the ending relative address for the dump. To do so, use the following form of the DMPL or DMPECS statement:

```
DMPL(fwa,lwa)          or          DMPECS(fwa,lwa)
```

Specifying LCM Dump Format

By default, the LCM dump is printed in a format similar to that used for the SCM dump shown in Figure 12-2. That is, SCOPE lists the LCM address followed by four groups of 20 octal digits indicating the contents of the address and the next three locations. To the right of the octal listing is a display code interpretation of each two octal digits (00 is interpreted as blank). You can request that words be listed only two at a time rather than four at a time. If you are listing octal instructions, you may prefer that the contents of the two words be listed in 15-bit groups (4 to a word) rather than having all 20 digits in a group. However, if you are listing data, you may feel that the listing is easier to read if the word contents are broken into five 12-bit bytes.

To obtain any of these options, specify the format parameter as the third parameter on the DMPL or DMPECS statement:

```
DMPL(fwa,lwa,f)          or          DMPECS(fwa,lwa,f)
```

Specify f as follows:

null, 0 or 1	List 4 words per line in 4 groups of 20 octal digits accompanied by character interpretation
2	List 2 words per line in 4 5-digit groups accompanied by character interpretation
3	List 2 words per line in 5 4-digit groups acconpanied by character interpretation
4	List 2 words per line in 2 groups of 20 octal digits accompanied by character interpretation

Specifying LCM Dump File

For LCM dumps, you also can specify a file other than OUTPUT to receive the dump. Remember, if you specify a file other than OUTPUT you are responsible for saving the file or routing it to a printer through the DISPOSE statement. Specify a file as the fourth parameter on the DMPL or DMPECS statement:

```
DMPL(fwa,lwa,f,lfn)          or          DMPECS(fwa,lwa,f,lfn)
```

The comma terminating the f field must be present even when f is null.

OBTAINING LOAD MAPS

Each time the loader is called for an object module (that is, relocatable load) you have the option of requesting a listing that describes where each module is loaded and what entry points and external symbols were used for loading. This listing is called a load map. You can control the contents of the map and can designate the file to receive it. A load map optionally contains the following items (Item 1 through 15 are illustrated in Figure 12-3. Items 16 through 18 are illustrated in Figure 12-4).

1. Identifies the output as a load map

2. Identifies which version of the LOADER is being used

3. Page number

4. Present only for overlay generation; identifies the overlay for which this is a map

5. Present only for overlay generation; indicates where overlay will be loaded when called (octal value)

6. Symbolic name for point of entry

7. Absolute value for point of entry in octal

8. Program or overlay length (SCM) in octal

9. Program or overlay length (LCM) in octal

10. A block name within the program or overlay, where // means blank common

11. Absolute beginning address of block, where an L following address indicates an LCM address

12. Length of the block

13. A list of entry points occurring in the program or overlay

14. A list of subroutines that references these entry points

15. The absolute address within the subroutine where the entry point is referenced

16. A list of unsatisfied externals

17. A list of subroutines in which the unsatisfied externals occur

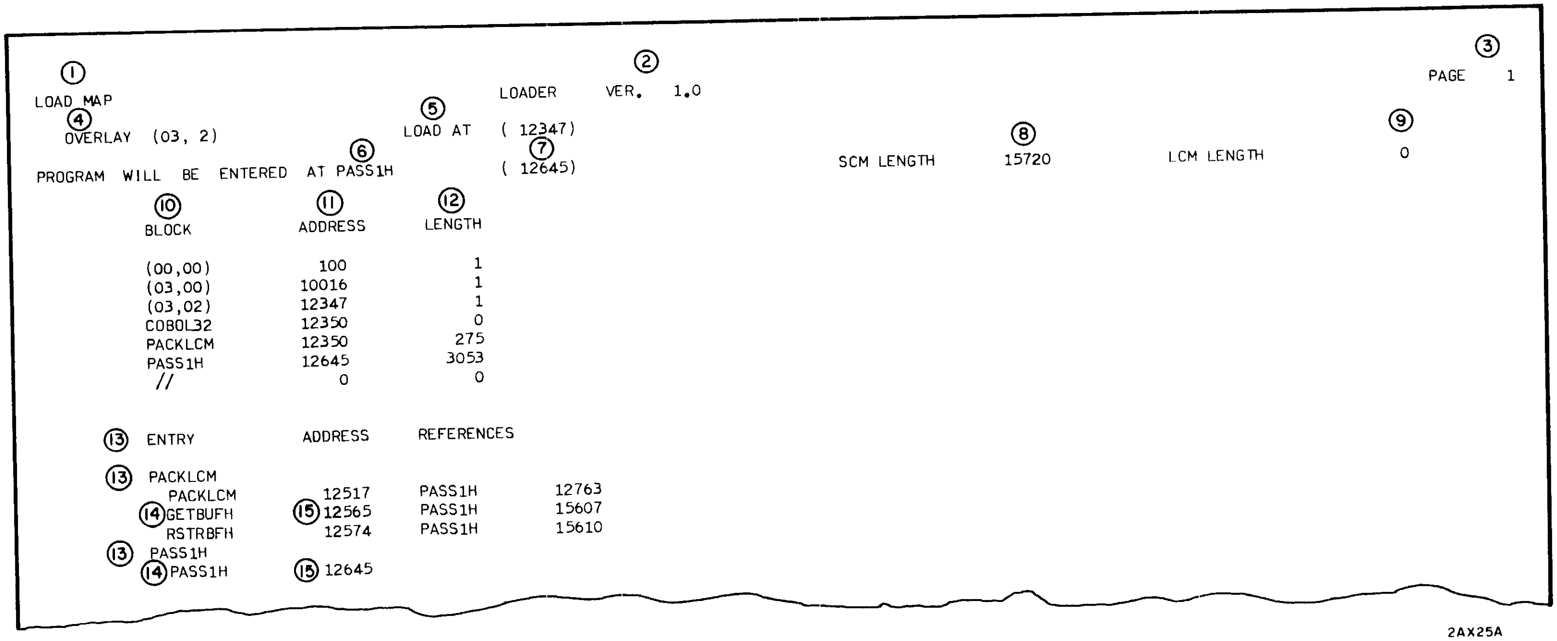
18. The absolute address within the subroutines where the unresolved external was referenced

A system parameter determines whether or not you receive a map with each load. This parameter can be set to specify no map, a partial map, or a full map. You may want to check with a systems analyst to learn what the default is for your site. For this discussion, let us assume that the system default specifies no map.

You can change this default for the duration of your job by using a MAP control statement, or you can change it for a load sequence through the use of the LDSET MAP option. Options also apply for overlay and segment generation.

MAP Statement

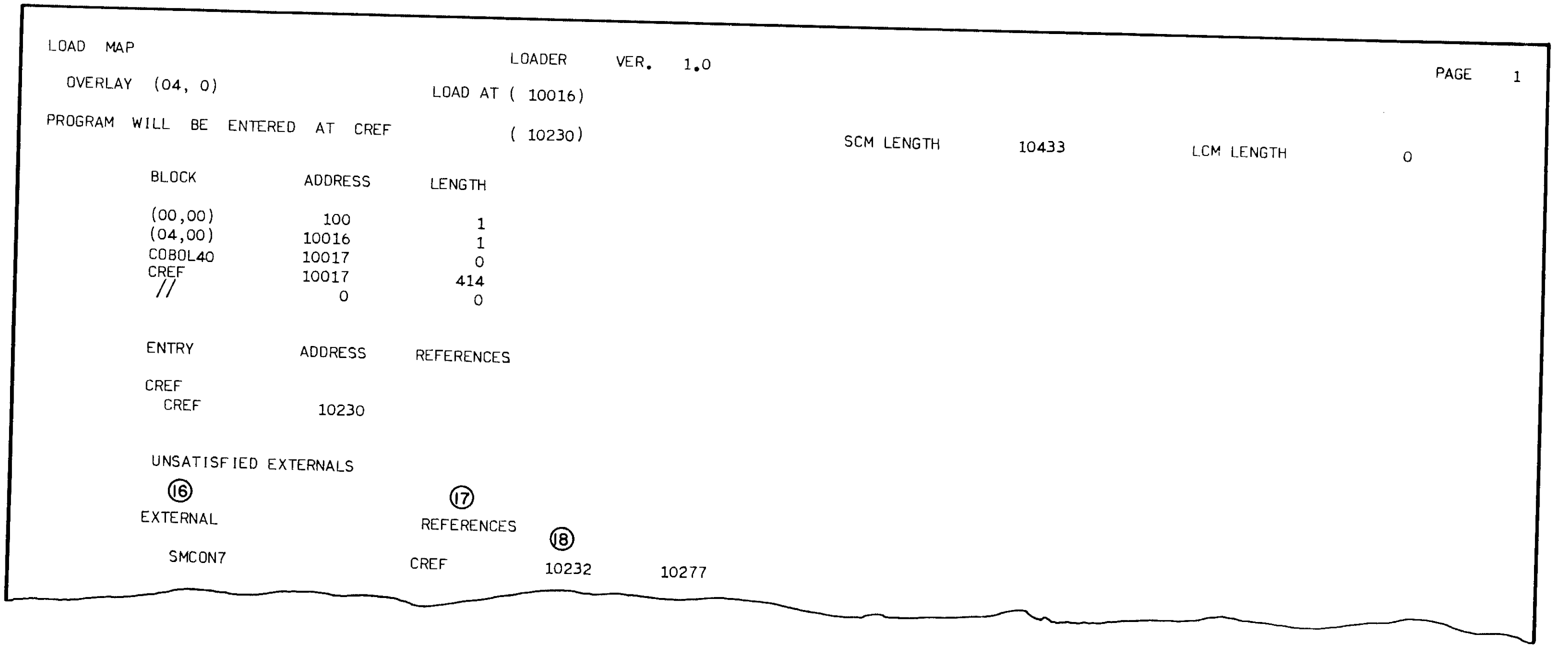
To change the map default for the job, place a MAP statement in your control statement section before the loads that are to be affected. Although it is a SCOPE control statement and not a loader control statement, MAP can occur within a load sequence for compatibility with previous systems (see Table 12-3 for MAP parameters).



```
LOAD MAP                                  LOADER     VER.   1.0                                                         PAGE    1
  OVERLAY  (03, 2)                 LOAD AT   ( 12347)
PROGRAM  WILL  BE  ENTERED  AT PASS1H        ( 12645)            SCM LENGTH      15720           LCM LENGTH          0
              BLOCK            ADDRESS      LENGTH

             (00,00)              100           1
             (03,00)            10016           1
             (03,02)            12347           1
             COBOL32            12350           0
             PACKLCM            12350         275
             PASS1H             12645        3053
              //                    0           0

           ENTRY               ADDRESS     REFERENCES

           PACKLCM
              PACKLCM             12517     PASS1H        12763
             GETBUFH              12565     PASS1H        15607
              RSTRBFH             12574     PASS1H        15610
           PASS1H
             PASS1H               12645
```

2AX25A

Figure 12-3. Loader Map-Part 1



```
LOAD  MAP                                        LOADER    VER.  1.0
 OVERLAY  (04, 0)                           LOAD AT ( 10016)                                                          PAGE   1
PROGRAM  WILL  BE  ENTERED  AT  CREF             ( 10230)             SCM LENGTH     10433        LCM LENGTH      0

              BLOCK          ADDRESS      LENGTH

              (00,00)          100           1
              (04,00)        10016           1
              COBOL40        10017           0
              CREF           10017         414
               //                0           0

              ENTRY          ADDRESS      REFERENCES

              CREF
                CREF          10230

              UNSATISFIED EXTERNALS

              EXTERNAL                    REFERENCES

                SMCON7                    CREF        10232       10277
```

16
17
18
2AX26A

Figure 12-4. Loader Map-Part 2

LDSET Option

To override the system default when there is no MAP statement or to override the MAP statement on a temporary basis, use an LDSET loader control statement with the MAP option specified (see Table 12-3 for options).

TABLE 12-3. MAP OPTIONS

MAP Contents	MAP Statement Parameter	LDSET Option
No map	OFF	MAP=0 or MAP=Ø
Partial map (items 1-9)	---	MAP=S
Partial map (items 13-15 omitted)	PART	MAP=B
Partial map (items 14, 15 omitted)	---	MAP=E
Full map	ON	MAP=X

In addition to allowing you greater control of map contents, the LDSET option also lets you determine the file to receive the map. You can specify a file by specifying the parameter as MAP=p/lfn where lfn is the name of the file to receive the map and p is the map option, or can change the file without changing the current map default by simply specifying MAP=/lfn. The default lfn is OUTPUT.

Example 12-7 illustrates user control of load maps.

OBTAINING FILE DUMPS

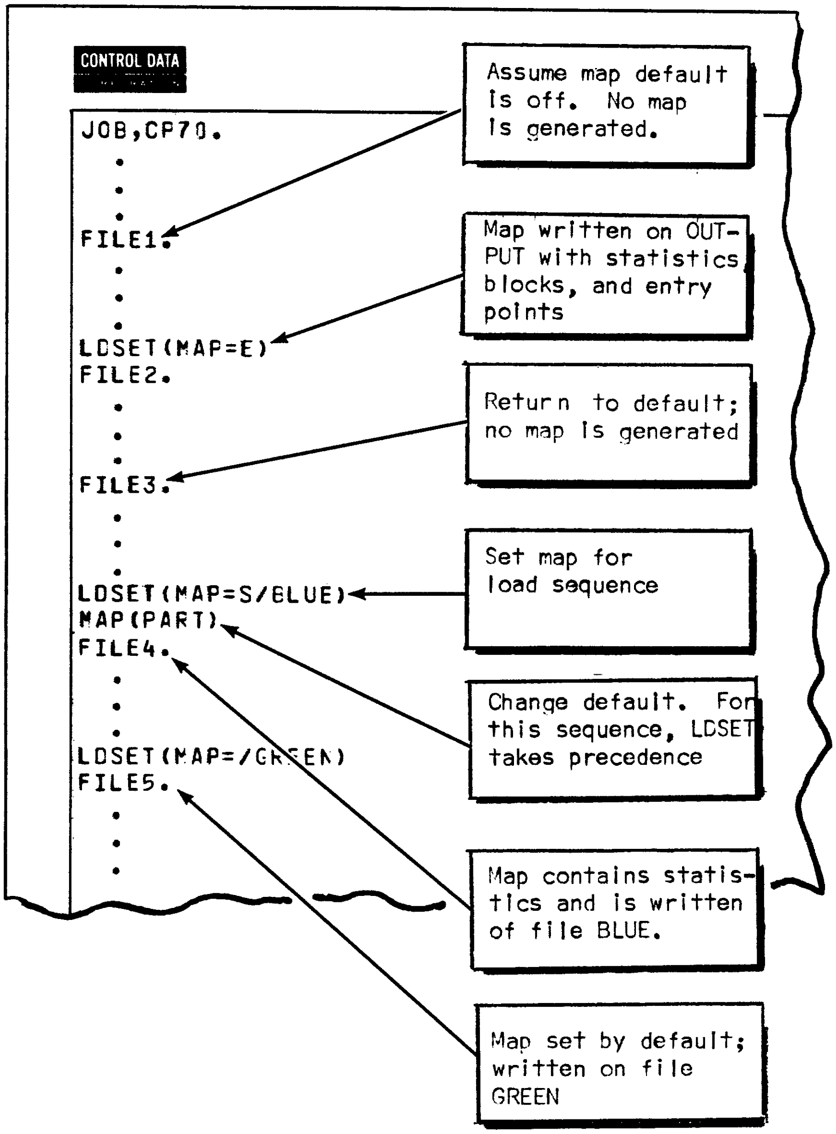
As an analytical aid, SCOPE 2 provides a utility routine that lists portions of a file. This routine is called with the DMPFILE statement and includes a number of options.

The dump format (Figure 12-5) resembles the standard dump format in that it lists the contents of the file in groups of four words accompanied by a display code representation of the contents. The number of words listed in each record and the number of records, sections, and partitions listed are user options. A file word address precedes each listing of the contents of a record. Because DMPFILE uses the record manager, any parity errors are noted but do not cause DMPFILE termination. The definition of a logical record is determined by record type. If you want to list all of the contents of the file, including W control words, I blocking control words, and recovery control words, or want to list the 48-bit appendages for S or Z record types, you can redefine the file as record type U, unblocked, before using DMPFILE. A blocked file must be closed before it can be redefined with a different file type. If the file is post-staged or is an on-line tape, you will not be able to redefine it as unblocked.

REQUESTING A DUMP OF ENTIRE FILE

Use the following control statement to list an entire file on the OUTPUT file:

```
DMPFILE(lfn)
```



```
JOB,CP70.
  .
  .
  .
FILE1.
  .
  .
  .
LDSET(MAP=E)
FILE2.
  .
  .
  .
FILE3.
  .
  .
  .
LDSET(MAP=S/BLUE)
MAP(PART)
FILE4.
  .
  .
  .
LDSET(MAP=/GREEN)
FILE5.
  .
  .
  .
```

Assume map default is off. No map is generated.

Map written on OUTPUT with statistics blocks, and entry points

Return to default; no map is generated

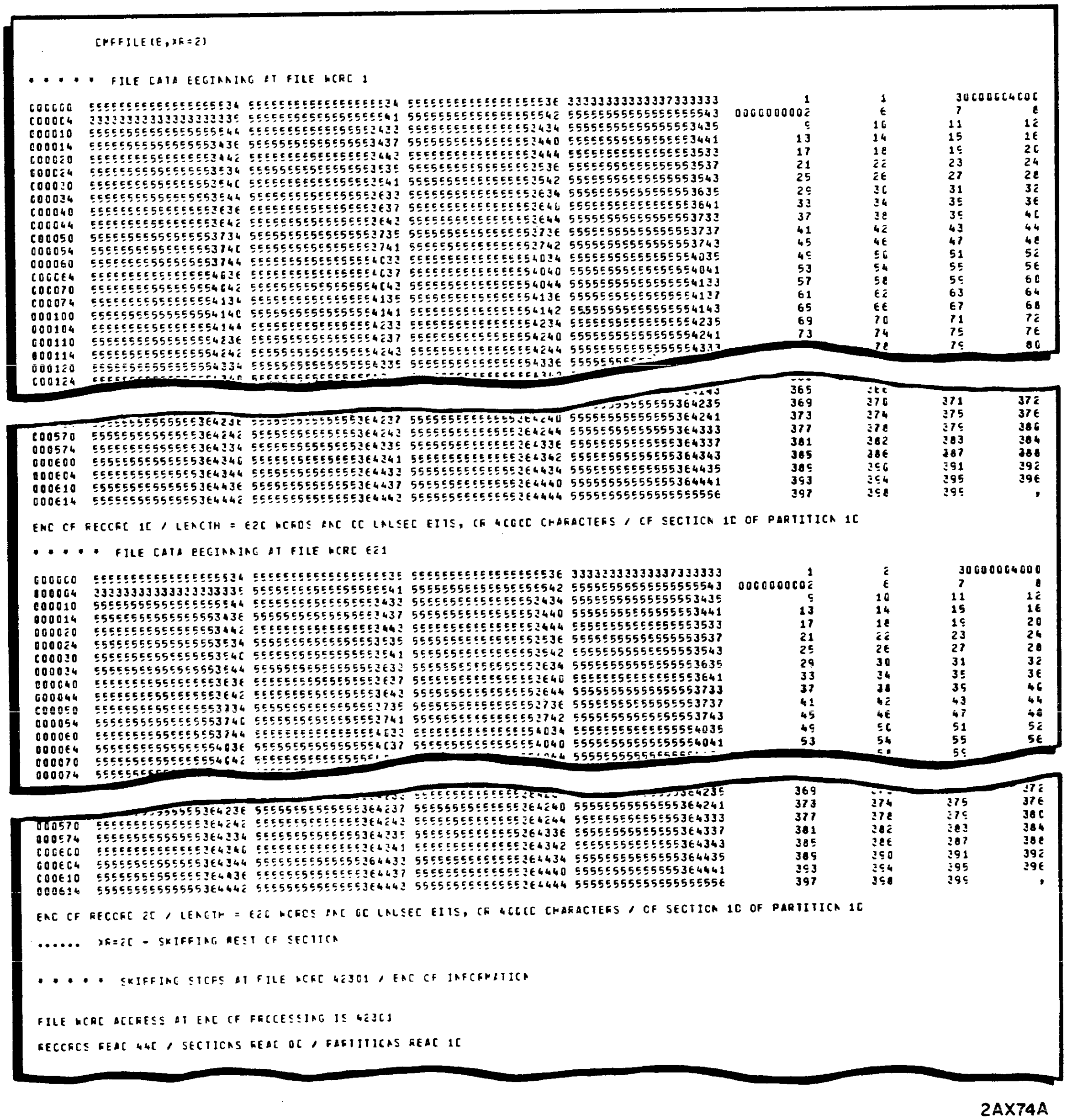
Set map for load sequence

Change default. For this sequence, LDSET takes precedence

Map contains statistics and is written of file BLUE.

Map set by default; written on file GREEN

Example 12-7. User Control of Load Map



```
        DMPFILE(E,NR=2)

* * * * *  FILE DATA BEGINNING AT FILE WORD 1

000000  55555555555555555534 55555555555555555524 55555555555555555536 33333333333333373333            1         1    3000004000
000004  33333333333333333335 55555555555555555541 55555555555555555542 55555555555555555543   0000000002         6         7         8
000010  55555555555555555544 55555555555555553433 55555555555555553434 55555555555555553435            9        10        11        12
000014  55555555555555553436 55555555555555553437 55555555555555553440 55555555555555553441           13        14        15        16
000020  55555555555555553442 55555555555555553443 55555555555555553444 55555555555555553533           17        18        19        20
000024  55555555555555553534 55555555555555553535 55555555555555553536 55555555555555553537           21        22        23        24
000030  55555555555555553540 55555555555555553541 55555555555555553542 55555555555555553543           25        26        27        28
000034  55555555555555553544 55555555555555553633 55555555555555553634 55555555555555553635           29        30        31        32
000040  55555555555555553636 55555555555555553637 55555555555555553640 55555555555555553641           33        34        35        36
000044  55555555555555553642 55555555555555553643 55555555555555553644 55555555555555553733           37        38        39        40
000050  55555555555555553734 55555555555555553735 55555555555555553736 55555555555555553737           41        42        43        44
000054  55555555555555553740 55555555555555553741 55555555555555553742 55555555555555553743           45        46        47        48
000060  55555555555555553744 55555555555555554033 55555555555555554034 55555555555555554035           49        50        51        52
000064  55555555555555554036 55555555555555554037 55555555555555554040 55555555555555554041           53        54        55        56
000070  55555555555555554042 55555555555555554043 55555555555555554044 55555555555555554133           57        58        59        60
000074  55555555555555554134 55555555555555554135 55555555555555554136 55555555555555554137           61        62        63        64
000100  55555555555555554140 55555555555555554141 55555555555555554142 55555555555555554143           65        66        67        68
000104  55555555555555554144 55555555555555554233 55555555555555554234 55555555555555554235           69        70        71        72
000110  55555555555555554236 55555555555555554237 55555555555555554240 55555555555555554241           73        74        75        76
000114  55555555555555554242 55555555555555554243 55555555555555554244 55555555555555554333                     78        79        80
000120  55555555555555554334 55555555555555554335 55555555555555554336 5555555555
000124  55555555555555554340 5555555555555555          55555555554342
                                                                   4143          365       366
                                                          5555555555364235          369       370       371       372
        55555555555555364236 55555555555555364237 55555555555555364240 55555555555555364241          373       374       375       376
000570  55555555555555364242 55555555555555364243 55555555555555364244 55555555555555364333          377       378       379       380
000574  55555555555555364334 55555555555555364335 55555555555555364336 55555555555555364337          381       382       383       384
000600  55555555555555364340 55555555555555364341 55555555555555364342 55555555555555364343          385       386       387       388
000604  55555555555555364344 55555555555555364433 55555555555555364434 55555555555555364435          389       390       391       392
000610  55555555555555364436 55555555555555364437 55555555555555364440 55555555555555364441          393       394       395       396
000614  55555555555555364442 55555555555555364443 55555555555555364444 55555555555555555556          397       398       399         ,

END OF RECORD 10 / LENGTH = 620 WORDS AND 00 UNUSED BITS, OR 40000 CHARACTERS / OF SECTION 10 OF PARTITION 10

* * * * *  FILE DATA BEGINNING AT FILE WORD 621

000000  55555555555555555534 55555555555555555535 55555555555555555536 33333333333333373333            1         2    3000004000
000004  33333333333333333335 55555555555555555541 55555555555555555542 55555555555555555543   0000000002         6         7         8
000010  55555555555555555544 55555555555555553433 55555555555555553434 55555555555555553435            9        10        11        12
000014  55555555555555553436 55555555555555553437 55555555555555553440 55555555555555553441           13        14        15        16
000020  55555555555555553442 55555555555555553443 55555555555555553444 55555555555555553533           17        18        19        20
000024  55555555555555553534 55555555555555553535 55555555555555553536 55555555555555553537           21        22        23        24
000030  55555555555555553540 55555555555555553541 55555555555555553542 55555555555555553543           25        26        27        28
000034  55555555555555553544 55555555555555553633 55555555555555553634 55555555555555553635           29        30        31        32
000040  55555555555555553636 55555555555555553637 55555555555555553640 55555555555555553641           33        34        35        36
000044  55555555555555553642 55555555555555553643 55555555555555553644 55555555555555553733           37        38        39        40
000050  55555555555555553734 55555555555555553735 55555555555555553736 55555555555555553737           41        42        43        44
000054  55555555555555553740 55555555555555553741 55555555555555553742 55555555555555553743           45        46        47        48
000060  55555555555555553744 55555555555555554033 55555555555555554034 55555555555555554035           49        50        51        52
000064  55555555555555554036 55555555555555554037 55555555555555554040 55555555555555554041           53        54        55        56
000070  55555555555555554042 5555555555555555                   4044 5555555555555555                      58        59
000074  555555
                                   233 5555555555555555364           5555364235          369                           372
        55555555555555364236 55555555555555364237 55555555555555364240 55555555555555364241          373       374       375       376
000570  55555555555555364242 55555555555555364243 55555555555555364244 55555555555555364333          377       378       379       380
000574  55555555555555364334 55555555555555364335 55555555555555364336 55555555555555364337          381       382       383       384
000600  55555555555555364340 55555555555555364341 55555555555555364342 55555555555555364343          385       386       387       388
000604  55555555555555364344 55555555555555364433 55555555555555364434 55555555555555364435          389       390       391       392
000610  55555555555555364436 55555555555555364437 55555555555555364440 55555555555555364441          393       394       395       396
000614  55555555555555364442 55555555555555364443 55555555555555364444 55555555555555555556          397       398       399         ,

END OF RECORD 20 / LENGTH = 620 WORDS AND 00 UNUSED BITS, OR 40000 CHARACTERS / OF SECTION 10 OF PARTITION 10

......  NR=20 - SKIPPING REST OF SECTION

* * * * *  SKIPPING STOPS AT FILE WORD 42301 / END OF INFORMATION

FILE WORD ADDRESS AT END OF PROCESSING IS 42301

RECORDS READ 440 / SECTIONS READ 00 / PARTITIONS READ 10
```

2AX74A

Figure 12-5. Sample of DMPFILE Output

SPECIFYING A LIST FILE

If you want your DMPFILE output placed on a file other than OUTPUT, specify a list file as L=$\mathrm{lfn}_{\mathrm{out}}$.

$\mathrm{DMPFILE}(\mathrm{lfn}_{\mathrm{in}}, \mathrm{L}{=}\mathrm{lfn}_{\mathrm{out}})$

Remember that if you specify a list file other than OUTPUT, you are responsible for saving the file contents.

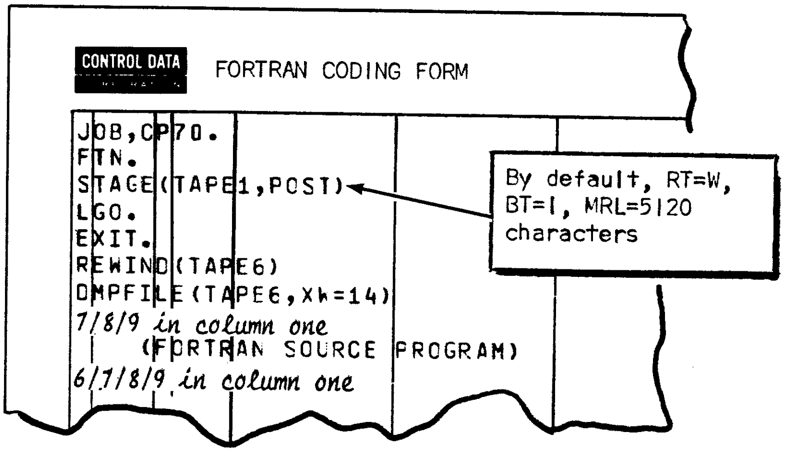
SPECIFYING DUMP LIMITS

When listing a file with very long records, you may want to sample only the beginning of each record. Similarly, if your file is very large, you may want to list only so many sections of each partition or so many partitions on the entire file. Several X parameters permit you to specify dump limits. These parameters and the L=parameter can be in any order after the comma for the input file field.

The allowable parameters are as follows:

XW=n	n is a decimal count of the number of words in each record to be dumped. If n is 0, DMPFILE lists the size and number but not the contents of the records in the file.
XR=n	n is a decimal count of the number of records in each section to be dumped.
XS=n	n is a decimal count of the number of sections in each partition to be dumped.
XP=n	n is a decimal count of the number of partitions in the file to be dumped.

In Example 12-8, the user requests a dump of file TAPE6 following abnormal job termination. The statement requests the first 14 words of each of the records on the file. The listing is written on OUTPUT.



CONTROL DATA
FORTRAN CODING FORM

```
JOB,CP70.
FTN.
STAGE(TAPE1,POST)
LGO.
EXIT.
REWIND(TAPE6)
DMPFILE(TAPE6,XW=14)
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
6/7/8/9 in column one
```

By default, RT=W, BT=I, MRL=5120 characters

Example 12-8. Requesting a File Dump

OBTAINING DAYFILE SUMMARIES

To analyze the performance of specific job steps, you can obtain intermediate accounting information at a specific point in your job. Place a SUMMARY statement in your job deck after the step you want analyzed. When SCOPE encounters the statement, it gives an accounting for the job up to that point. SUMMARY has no parameters.

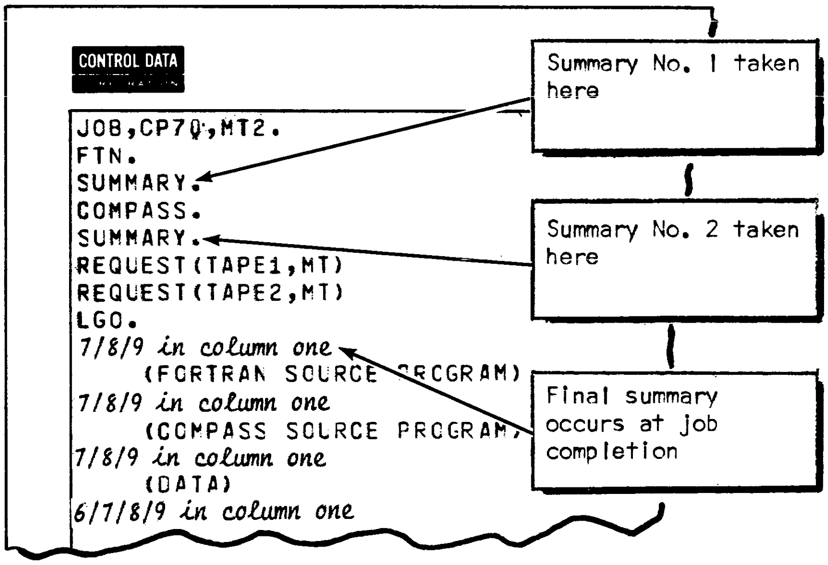
Example 12-9 illustrates a job in which the user wants information about the compilation and the assembly. By comparing these two intermediate summaries with the job completion summary, he can determine the resources used for each job step.

Here are some simple guidelines for reducing the use of resources while running a job.

1. Use dynamic memory allocation. If you must use user-controlled memory allocation, don't increase your field size until just before the job step that requires the memory, and then return to automatic mode immediately after the job step.

2. Don't acquire a resource before you need it. In general, this means that you should place statements such as REQUEST and ATTACH before the job step that uses them. For example, if your FORTRAN job uses an on-line tape named TAPE1, the REQUEST (TAPE1,MT) statement should lie between the FTN. and LGO. statements, not before the FTN statement.

3. Relinquish a resource as soon as you no longer need it. This means that you should return any mass storage file or unload any on-line tape when you are through using it without waiting for job termination. This includes post-staged tapes and disposed files.



```
JOB,CP70,MT2.
FTN.
SUMMARY.
COMPASS.
SUMMARY.
REQUEST(TAPE1,MT)
REQUEST(TAPE2,MT)
LGO.
7/8/9 in column one
     (FORTRAN SOURCE PROGRAM)
7/8/9 in column one
     (COMPASS SOURCE PROGRAM)
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

Example 12-9. Intermediate Accounting Information

A

CDC CYBER 70 STANDARD CHARACTER SET

CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code	CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code
:†	:	00†	8-2	00	8-2	3A	6	6	41	6	06	6	36
A	A	01	12-1	61	12-1	41	7	7	42	7	07	7	37
B	B	02	12-2	62	12-2	42	8	8	43	8	10	8	38
C	C	03	12-3	63	12-3	43	9	9	44	9	11	9	39
D	D	04	12-4	64	12-4	44	+	+	45	12	60	12-8-6	2B
E	E	05	12-5	65	12-5	45	–	–	46	11	40	11	2D
F	F	06	12-6	66	12-6	46	*	*	47	11-8-4	54	11-8-4	2A
G	G	07	12-7	67	12-7	47	/	/	50	0-1	21	0-1	2F
H	H	10	12-8	70	12-8	48	(	(	51	0-8-4	34	12-8-5	28
I	I	11	12-9	71	12-9	49	)	)	52	12-8-4	74	11-8-5	29
J	J	12	11-1	41	11-1	4A	$	$	53	11-8-3	53	11-8-3	24
K	K	13	11-2	42	11-2	4B	=	=	54	8-3	13	8-6	3D
L	L	14	11-3	43	11-3	4C	blank	blank	55	no punch	20	no punch	20
M	M	15	11-4	44	11-4	4D	, (comma)	, (comma)	56	0-8-3	33	0-8-3	2C
N	N	16	11-5	45	11-5	4E	. (period)	. (period)	57	12-8-3	73	12-8-3	2E
O	O	17	11-6	46	11-6	4F	≡	#	60	0-8-6	36	8-3	23
P	P	20	11-7	47	11-7	50	[	[	61	8-7	17	12-8-2	5B
Q	Q	21	11-8	50	11-8	51	]	]	62	0-8-2	32	11-8-2	5D
R	R	22	11-9	51	11-9	52	%††	%	63	8-6	16	0-8-4	25
S	S	23	0-2	22	0-2	53	≠	" (quote)	64	8-4	14	8-7	22
T	T	24	0-3	23	0-3	54	→	_ (underline)	65	0-8-5	35	0-8-5	5F
U	U	25	0-4	24	0-4	55	∨	!	66	11-0 or	52	12-8-7 or	21
V	V	26	0-5	25	0-5	56				11-8-2†††		11-0†††	
W	W	27	0-6	26	0-6	57	∧	&	67	0-8-7	37	12	26
X	X	30	0-7	27	0-7	58	↑	' (apostrophe)	70	11-8-5	55	8-5	27
Y	Y	31	0-8	30	0-8	59	↓	?	71	11-8-6	56	0-8-7	3F
Z	Z	32	0-9	31	0-9	5A	<	<	72	12-0 or	72	12-8-4 or	3C
0	0	33	0	12	0	30				12-8-2†††		12-0†††	
1	1	34	1	01	1	31	>	>	73	11-8-7	57	0-8-6	3E
2	2	35	2	02	2	32	≤	@	74	8-5	15	8-4	40
3	3	36	3	03	3	33	≥	\	75	12-8-5	75	0-8-2	5C
4	4	37	4	04	4	34	¬	^ (circumflex)	76	12-8-6	76	11-8-7	5E
5	5	40	5	05	5	35	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	3B

† For 6000 SCOPE, 12 or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

†† In installations using the CDC 63-graphic set, display code 00 has no associated graphic or Hollerith code; display code 63 is the colon (8-2 punch).

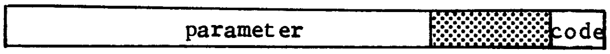
††† The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

JOB COMMUNICATION AREA B

	59	55	36	29	18	13	06	00
RA(S)					a	g	ss	sl
RA(S)+1	User/System Interface							
RA(S)+2 • • • RA(S)+53_8	Parameters from the program call statement (Available to user during job execution)							
RA(S)+54_8 • • • RA(S)+63_8	Used for absolute program loading							
RA(S)+64_8	name				np			
RA(S)+65_8	lwaec(lcm)		▨		ℓ	lwacm(scm)		
RA(S)+66_8	x	fwaec(lcm)	▨		d	fwacm(scm)		
RA(S)+67_8	Reserved for loader			c	Reserved for loader			
RA(S)+70_8 • • • RA(S)+77_8	Control statement image							

Word	Bits	Field	Significance
RA(S)	59-18	none	Reserved
	17	a	Abort flag (SCOPE 3.4 only)
	16-13	none	Reserved
	12	g	Go/pause flag 0 Go 1 Pause; wait for go
	11-06	ss	Sense switches
	05-00	sl	Sense lights
RA(S)+1	59-00	user/ system interface	Reserved for use during execution (SCOPE 3.4 only)
RA(S)+2 through RA(S)+63_8	59-00	params	Parameters from the program call card; available to user during execution

A keyword or value occupies bits 59-18. A delimiter is converted to a code and placed in bits 03-00.



Code	Delimiter	Character
00	Continuation	None
01	comma	,
02	Equal sign	=
03	Slash	/
04	Left parenthesis	(
05	Plus sign	+
06	Minus sign	-
07	Blank	
10	Semicolon	;
11	Reserved	
12		
13		
14		
15		
16	Other	
17	Termination	. or)

Word	Bits	Field	Significance
RA(S)+64_8	59-18	name	Name of file or program from control card or macro
	17-00	np	Number of parameters

Word	Bits	Field	Significance
RA(S)+65_8	59-36	lwaec(lcm)	Last word address +1 of loadable area in ECS/LCM of most recently completed load operation.
	35-19	none	Unused; zero
	18	ℓ	Library flag†: 0 Name is a file name 1 Name is a library name
	17-00	lwacm(scm)	Last word address +1 of loadable area in CM/SCM of most recently completed load operation.
RA(S)+66_8	59	x	Set to 1 if CPU program† can issue XJ instruction.
	58-36	fwaec(lcm)	First word address of loadable area in ECS/LCM of most recently completed load operation.
	35-19	none	Unused; zero
	18	d	RSS flag†: 0 Not in RSS mode 1 RSS mode while using DIS
	17-00	fwacm(scm)	First word address of loadable area in CM/SCM of most recently completed load operation.
RA(S)+67_8	59-30	Reserved	Used by loader
	29	c	LDV completion flag; set by LDV upon completion of execution. Required for SCOPE 3.3 compatibility.
	28-00	Reserved	Used by loader
RA(S)+70 through RA(S)+77_8	59-00	card image	Control card image

† SCOPE 3.4 only.

STANDARD LABELS C

STANDARD LABEL TYPES

The four types of labels in the Standard are identified by the first four characters of the label. Contents of labels are described later.

Type	Identifier	Significance
Volume Header	VOL1	Beginning of volume
File Header	HDR1	Beginning of information
End of Volume	EOV1	End of volume
End of File	EOF1	End of information

LABEL GROUPS

Each occurrence of one or more of the labels is called a group. The following groups are possible:

Volume/header group composed of a VOL1 label and a HDR1 label separated from each other by an interrecord gap and from the beginning of information on the file by tapemark.

End-of-file group composed of an EOF1 label separated from the end of information on the preceding file by a tapemark. If the volume contains only one file, the EOF1 label is followed by a double tapemark. If the volume contains multiple files, the EOF1 label is separated from the next volume/header group by a tapemark.

End-of-volume group composed of a EOV1 label separated from the information by a tapemark and followed by a double tapemark.

Several different combinations of files and tapes are possible. First consider the simplest case, the single volume of tape containing only one file. This volume has label groups as shown in Figure C-1.

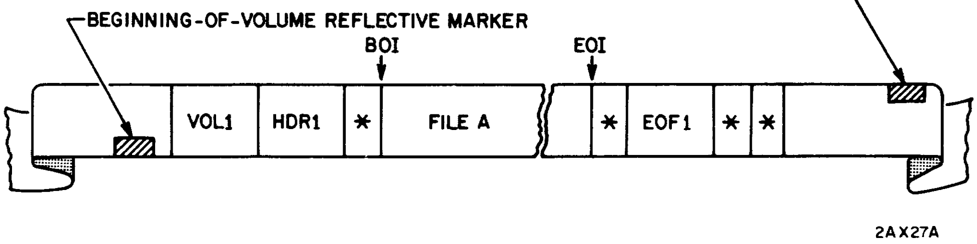


Figure C-1. Single Volume File

Notice that the tape cannot be read beyond the double tapemark. If the information on the tape is extended beyond the current end-of-information, the EOF1 label and double tapemark are overwritten and a new label and double tapemark are written following the data.

Notice also that the terminating label group consists of an EOF1 label, not an EOV1 label. An EOV1 label is written only when data must be continued on a subsequent volume. In other words, the EOV1 label tells you that file information is coming from another reel. As an example, consider the multivolume file illustrated in Figure C-2.

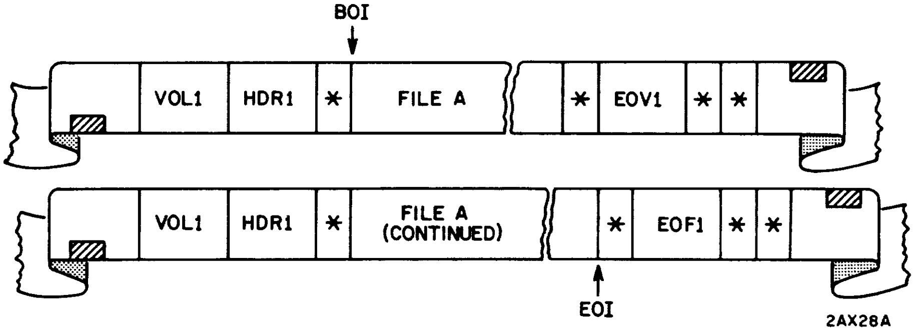


Figure C-2. Multivolume File

Notice that when a file is continued on a second reel (Figure C-2), it is headed again by a volume/header group. On the final volume for the file, it is terminated by an end-of-file group.

Both tapes in the preceding illustrations terminate with double tapemarks. This is standard for all SCOPE 2 tapes. Figure C-3 illustrates a volume containing multiple files. Here, a file is not terminated by a double tapemark when it is followed by another file. This format is not currently supported by SCOPE 2.

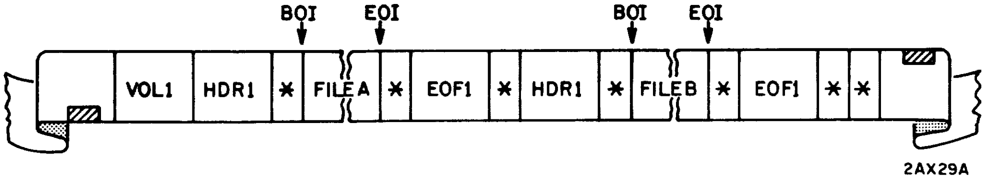


Figure C-3. Multifile Volume

Let us also consider the multivolume multifile set composed of several labeled files occupying several volumes (Figure C-4). Although this combination is allowed under 6000 SCOPE, it is not allowed under SCOPE 2.

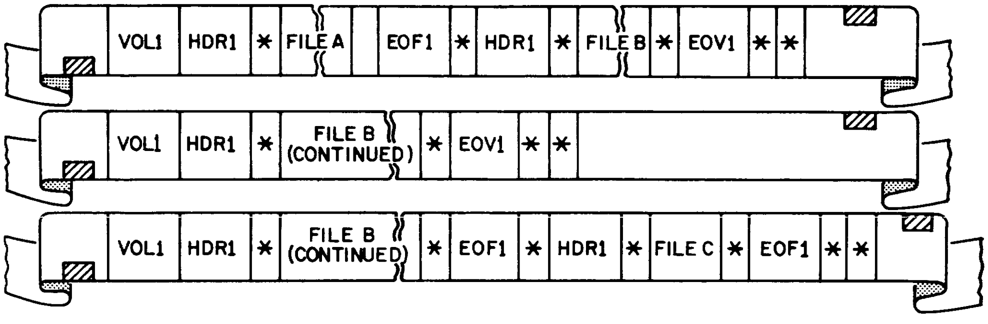


Figure C-4. Multivolume Multifile

On rare occasions, the end-of-volume reflective marker and the end-of-information may coincide. When this occurs, the labeling that terminates a reel is somewhat different from the labeling previously described. For example, Figure C-5 shows that the end-of-volume marker is reached at the same point that file A concludes. A tapemark is written followed by an end-of-volume label and a double tapemark.

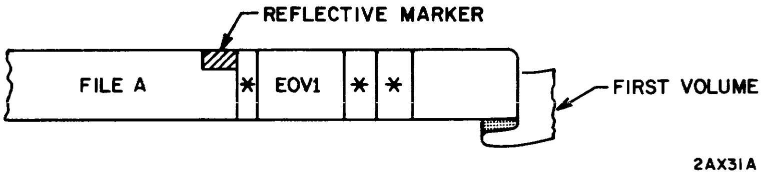


Figure C-5. Simultaneous EOI and EOV Marker

In this case, the end-of-file label must be recorded on the next volume. Figure C-6 illustrates the format used on the second reel:

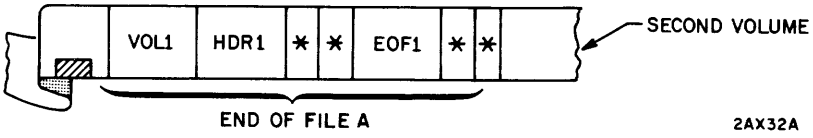


Figure C-6. Continuation of EOF Labeling

The double tapemark between the volume/header group and the end-of-file trailer group indicates that no further data appears within file A. Since this condition constitutes a multivolume set, file A cannot be followed by a second file for SCOPE 2 usage. That is, the presence of file B would produce a multivolume multifile set.

LABEL CONTENT

In the following description, if a field is optional, it contains either the designated information or blanks. Digits are 0 through 9. A character is any of the characters listed for the 63-character subset. (See Label Parity and Character Conversion.)

Volume Header Label

The volume header label is an 80-character block that must appear as the first block of every tape volume (reel).

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	Must be VOL
4	Label number	1	Must be 1
5-10	Volume serial number	6	Six characters permanently assigned by the owner to identify this physical volume and supplied on STAGE or REQUEST statement
11	Accessibility	1	A character that indicates any restrictions on who may have access to the information in the volume. A blank means unlimited access. Any other character means special handling in the manner agreed between the interchange parties.
12-37	Reserved for future standardization	26	Must be blanks
38-51	Owner identification	14	Any characters identifying the owner of the physical volume
52-79	Reserved for future standardization	28	Must be blanks
80	Label standard level	1	1 means the labels and data formats on this volume conform to the requirements of this standard. Blank means the labels and data formats on this volume require the agreement of the interchange parties. Record manager uses 1.

File Header Label

The file header label must be the first 80-character block of a file. When a file is the first on a volume or is continued on more than one volume, the file header label must be repeated after the volume header label on each new volume for the file.

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	Must be HDR
4	Label number	1	Must be 1
5-21	File identifier	17	Any characters agreed on between the originator and the recipient
22-27	Set identification	6	Any characters to identify the set in which this file is included. This identification must be the same for all files of a multifile set.
28-31	File section number	4	0001 for the first header label of a file. This value is incremented by 1 for each continuation of the file on a new volume.
32-35	File sequence number	4	Four digits denoting the sequence of files within the set: the first file is 0001, the second 0002, etc. In all labels for a single file, this field will contain the same number.
36-39	Generation number (optional)	4	Four digits denoting the current stage in the succession of one file being generated by an update to a previous file. When a file is first created, its generation number is 0001.
40-41	Generation version number (optional)	2	Two digits distinguishing successive iterations of the same generation. The generation version number of the first attempt to produce a file is 00.
42-47	Creation data in Julian format	6	A space followed by two digits for the year, followed by three digits (001-366) for the day.
48-53	Expiration date in Julian format	6	Same as the preceding field. The file is regarded as expired when today's date is equal to or later than the expiration date. When the file is expired, the remainder of this volume may be overwritten. Thus, to be effective on multivolumes, the expiration date of a file must be less than or equal to the expiration date of all previous files on the volume. SCOPE 2 checks only the first file expiration date.

Character Position	Field Name	Length in Characters	Contents
54	Accessibility	1	A character that denotes any restrictions on who may have access to the information in this file. A blank means unlimited access. Any other character means special handling in a manner agreed between the interchange parties. Record manager uses blank.
55-60	Block count	6	Must be zeros
61-73	System code (optional)	13	Thirteen characters identifying the operating system that recorded this file. Record manager uses blanks.
74-80	Reserved for future standardization	7	Must be blanks

End-of-File Label

The 80-character end-of-file label is the last block of a file.

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	Must be EOF
4	Label number	1	Must be 1
5-21	File identifier	17	Same as the corresponding fields in the first file header label for this file
22-27	Set identification	6	
28-31	File section number	4	
32-35	File sequence number	4	
36-39	Generation number (optional)	4	
40-41	Generation version number (optional)	2	
42-27	Creation date	6	
48-53	Expiration date	6	
54	Accessibility	1	
55-60	Block count	6	Six digits denoting the number of data blocks (exclusive of labels and tapemarks) since the preceding HDR label group.

Character Position	Field Name	Length in Characters	Contents
61-73	System code (optional)	13	Same as the corresponding field in the first file header label for this file
74-80	Reserved for future standardization	7	Must be blanks

End-of-Volume Label

The 80-character end-of-volume label appears at the end of all but the last volume in a set.

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	Must be EOV
4	Label number	1	Must be 1
5-21	File identifier	17	Same as the corresponding fields in the first file header label of the last file begun on this volume
22-27	Set identification	6	
28-31	File section number	4	
32-35	File sequence number	4	
36-39	Generation number (optional)	4	
40-41	Generation version number (optional)	2	
42-47	Creation date	6	
48-53	Expiration date	6	
54	Accessibility	1	
55-60	Block count	6	Six digits denoting the number of data blocks (exclusive of labels and tapemarks) since the preceding HDR label group
61-73	System code	13	Thirteen characters identifying the operating system that recorded this file. Record manager uses blanks.
74-80	Reserved for future standardization	7	Must be blanks

SUMMARY OF FILE FORMATS D

This appendix summarizes record types and block types supported by the record manager.

W RECORDS

The W record type is the most important record type in the SCOPE 2 system. It is the only record type recognized by the loader and is the only record type that can be disposed, that is, printed or punched. It is also used for the INPUT file. Refer to Section 9 for descriptions of printer input, and punched card input and output.

In the SCOPE 2 system, the default record type is W. A notable exception is COBOL. For W to be the default within COBOL requires a combination of variable length records according to the RECORD CONTAINS clause, and the BLOCK CONTAINS clause. Otherwise, the user can explicitly set the record type to W by using a FILE statement with RT=W.

On an output request (for example, a FORTRAN WRITE statement) the record manager expands the data to the nearest full-word boundary and prefixes the data with a W control word (Figure D-1). Thus, in its recorded form, the W record consists of a W control word followed by an integral number of 60-bit words. The last word may be partially unused, that is, a field in the W control word indicates how many bits of the last word in the record do not contain data. The unused bits are zeroed.

NOTE

If you want your program to be able to accept W or S records, always read or write full words.

A W record is considered "zero-length" when it consists of a W control word only and is not accompanied by data. Zero-length records serve as partition and section delimiters.

On an input request (for example, a FORTRAN READ statement) the record manager removes the W control word and places the data in the user buffer.

On magnetic tape, W records can be recorded in binary mode only. When using formatted reads and writes, the maximum record size is limited to 150 by the FORTRAN object-time routines.

S RECORDS

S records are also known as 6000 SCOPE Standard and 7600 SCOPE 1.1 I-Mode. This is the only record type recognized by both SCOPE 3.3 and SCOPE 2.

Only two record types, S and X are defined in terms of physical blocks. The S record (Figure D-2) consists of blocks of data terminated by a short block. For S records, blocking is set to C-type by the system BT=C is not required. The short block has a 48-bit level number appended to it where level is $0\text{-}16_8$. If the data portion of the record terminates on a block boundary, the terminating short block is considered as zero-length because it consists of the 48-bit level number only. Another kind of zero-length block, the partition delimiter where level is 17_8, can also occur.

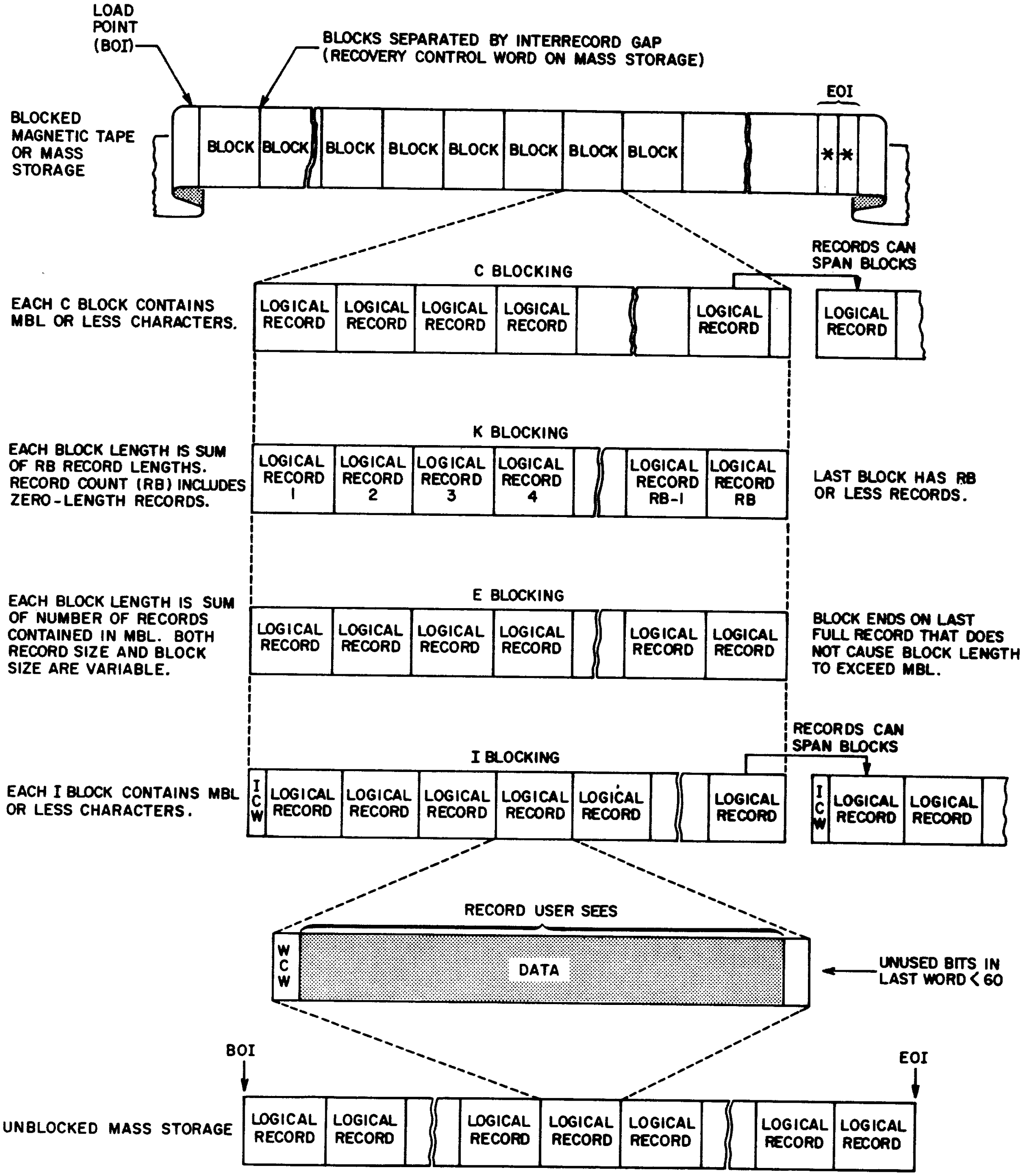

LOAD POINT (BOI)
BLOCKS SEPARATED BY INTERRECORD GAP (RECOVERY CONTROL WORD ON MASS STORAGE)
BLOCKED MAGNETIC TAPE OR MASS STORAGE
BLOCK
EOI
*
C BLOCKING
RECORDS CAN SPAN BLOCKS
EACH C BLOCK CONTAINS MBL OR LESS CHARACTERS.
LOGICAL RECORD
K BLOCKING
EACH BLOCK LENGTH IS SUM OF RB RECORD LENGTHS. RECORD COUNT (RB) INCLUDES ZERO-LENGTH RECORDS.
LOGICAL RECORD 1
LOGICAL RECORD 2
LOGICAL RECORD 3
LOGICAL RECORD 4
LOGICAL RECORD RB-1
LOGICAL RECORD RB
LAST BLOCK HAS RB OR LESS RECORDS.
E BLOCKING
EACH BLOCK LENGTH IS SUM OF NUMBER OF RECORDS CONTAINED IN MBL. BOTH RECORD SIZE AND BLOCK SIZE ARE VARIABLE.
BLOCK ENDS ON LAST FULL RECORD THAT DOES NOT CAUSE BLOCK LENGTH TO EXCEED MBL.
I BLOCKING
RECORDS CAN SPAN BLOCKS
EACH I BLOCK CONTAINS MBL OR LESS CHARACTERS.
ICW
RECORD USER SEES
WCW
DATA
UNUSED BITS IN LAST WORD < 60
BOI
EOI
UNBLOCKED MASS STORAGE
2AX33A

Figure D-1. W Record Control Word Format

In the user buffer, the S record consists of the data without the 48-bit appendage. On an output request (for example, a FORTRAN WRITE or BUFFER OUT), the record manager blocks the data and adds the 48-bit level number. On output, the level number is always zero for records.

On an input request (for example, a FORTRAN READ or BUFFER IN statement), the record manager deblocks the data, removes the 48-bit level number, and returns end-of-record status. End-of-record status is passed to the user. The level number is maintained in the FIT. An S record must be a multiple of 10 characters (full 60-bit words).

To specify record type S, set RT=S on the FILE statement. On magnetic tape, S records can be recorded in binary mode only. When using formatted reads and writes, the maximum record size is limited to 150 by the FORTRAN object-time routines.

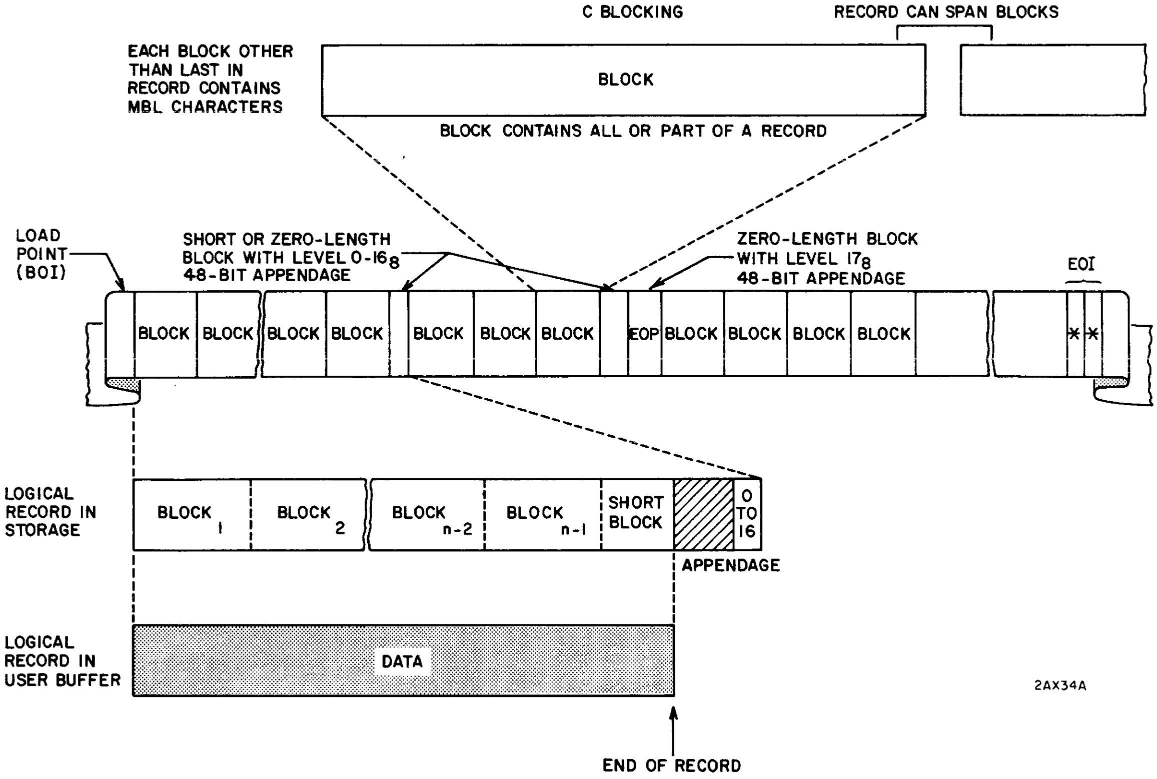

C BLOCKING
RECORD CAN SPAN BLOCKS
EACH BLOCK OTHER THAN LAST IN RECORD CONTAINS MBL CHARACTERS
BLOCK
BLOCK CONTAINS ALL OR PART OF A RECORD
LOAD POINT (BOI)
SHORT OR ZERO-LENGTH BLOCK WITH LEVEL 0-16₈ 48-BIT APPENDAGE
ZERO-LENGTH BLOCK WITH LEVEL 17₈ 48-BIT APPENDAGE
EOI
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
EOP
BLOCK
BLOCK
BLOCK
BLOCK
*
*
LOGICAL RECORD IN STORAGE
BLOCK 1
BLOCK 2
BLOCK n-2
BLOCK n-1
SHORT BLOCK
0 TO 16
APPENDAGE
LOGICAL RECORD IN USER BUFFER
DATA
2AX34A
END OF RECORD

Figure D-2. S SCOPE Logical Record Format

X RECORDS

SCOPE 2 allows reading of files consisting of X records. It does not allow a user to create files with X records. X record type is the same as X-mode format for 6000

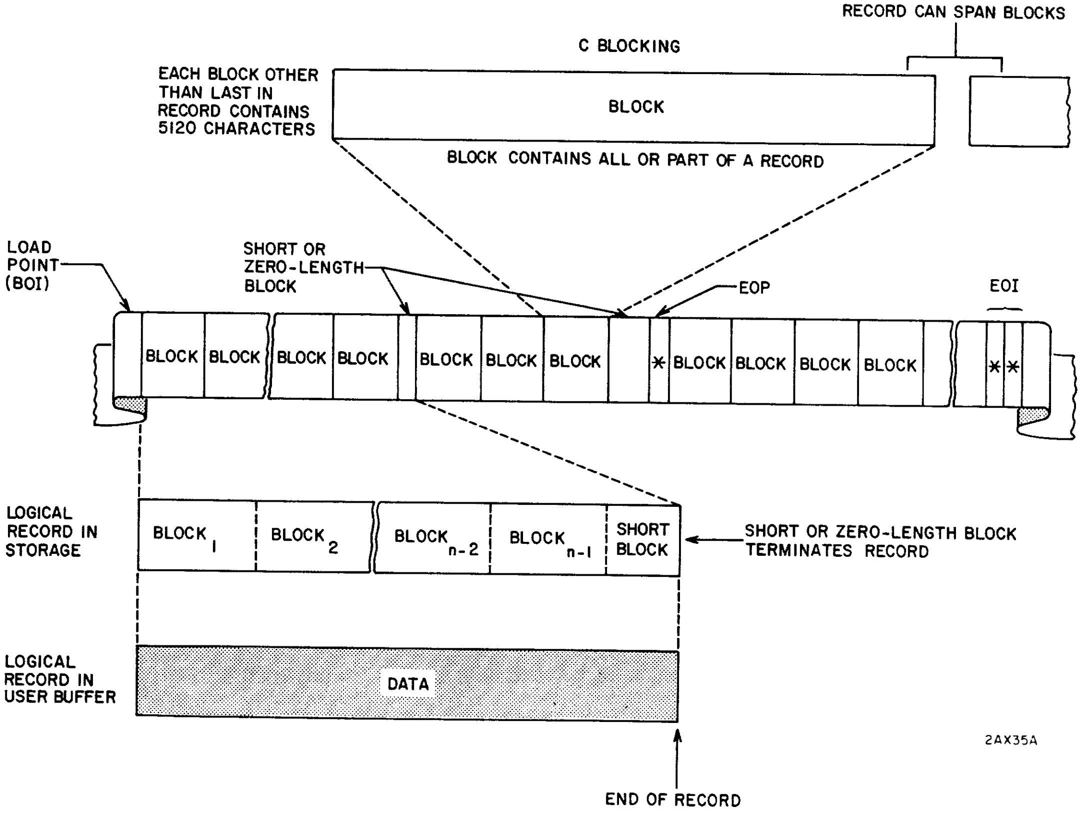
SCOPE 3.2 and 7600 SCOPE 1.1. A logical record is divided into fixed-length blocks containing 60-bit words. Each block contains 5120 characters. A short block marks end-of-record.

If the length is an exact multiple of blocks, an additional short block, sometimes called a zero-length block because it contains no data, is necessary. On input, the record manager then deletes this zero-length block consisting of 48 bits of zero.

X records differ from S records in that the 48-bit appendage appears only when necessary, and there is no level number associated with the record.

An X record must be a multiple of 10 characters (full 60-bit words). To specify type X, set RT=X on a FILE statement.

Conventionally, end-of-information is not defined for X-mode tapes. It is the user's responsibility to know the structure of the tape (that is, number of partitions as indicated by tapemarks). 6000 SCOPE 3.2 and 7600 SCOPE 1.1 append four and two tapemarks respectively, at the end of data.



RECORD CAN SPAN BLOCKS
C BLOCKING
EACH BLOCK OTHER THAN LAST IN RECORD CONTAINS 5120 CHARACTERS
BLOCK
BLOCK CONTAINS ALL OR PART OF A RECORD
LOAD POINT (BOI)
SHORT OR ZERO-LENGTH BLOCK
EOP
EOI
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
*
BLOCK
BLOCK
BLOCK
BLOCK
*
*
LOGICAL RECORD IN STORAGE
BLOCK 1
BLOCK 2
BLOCK n-2
BLOCK n-1
SHORT BLOCK
SHORT OR ZERO-LENGTH BLOCK TERMINATES RECORD
LOGICAL RECORD IN USER BUFFER
DATA
2AX35A
END OF RECORD

Figure D-3. X X-Mode Logical Record Format

Z RECORDS

The Z (zero byte) record type is commonly used in the 6000 SCOPE systems and the 7600 SCOPE 1.1 system for print or card files.

On a file, the Z record consists of an integral number of 60-bit words of data in which the last word has the low-order 12 bits set to zero (contains a zero byte).

In the user form, the Z record consists of a fixed number of characters of data. The user does not see the zero bytes.

Consider first what happens on an output request such as a FORTRAN WRITE or BUFFER OUT statement. Referring to Figure D-4, the record manager takes a specified number of characters (FL) from the buffer and either (A) removes trailing blanks to the nearest word boundary minus 2 characters and inserts 12 bits of zero in the low-order position of the word, or (B) adds blanks to the nearest word boundary minus 2 characters and appends a 12-bit zero byte. For example, if FL is 80 and you supply 62 characters of data followed by 18 blanks, the record manager removes 12 blanks to make 68 characters and adds a zero byte to make a full 7 words. If, on the other hand, you had supplied 79 characters, the record manager would add 9 blanks making 88 characters which, with the addition of the zero byte, makes the Z record 9 words.

Now consider what happens on an input request such as a FORTRAN READ or BUFFER IN statement. When reading Z records from a file, the record manager takes full words of data until it detects a word with a zero byte in the low order position of the word. It removes the zero byte and places the data in the user buffer. If the number of characters of data is less than FL, the record manager adds blanks to FL characters. This means if FL is 80 and the record manager reads a 68-character record (discounting the zero byte), the data is expanded to 80 characters. On input, when the record manager reads 88 characters (of which 9 are blanks added on the write), the Z record exceeds the FL specified. This does not result in an error since the record manager allows a Z record to exceed FL by as much as one word as long as the characters are all blanks.

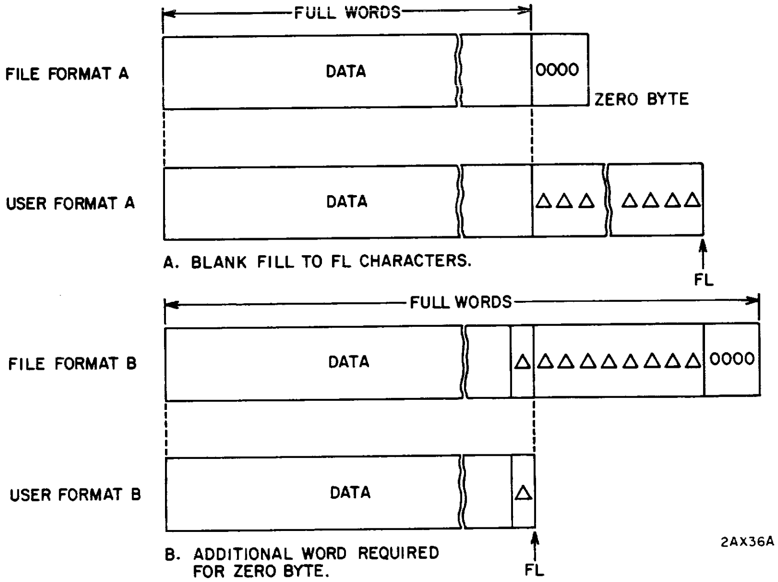


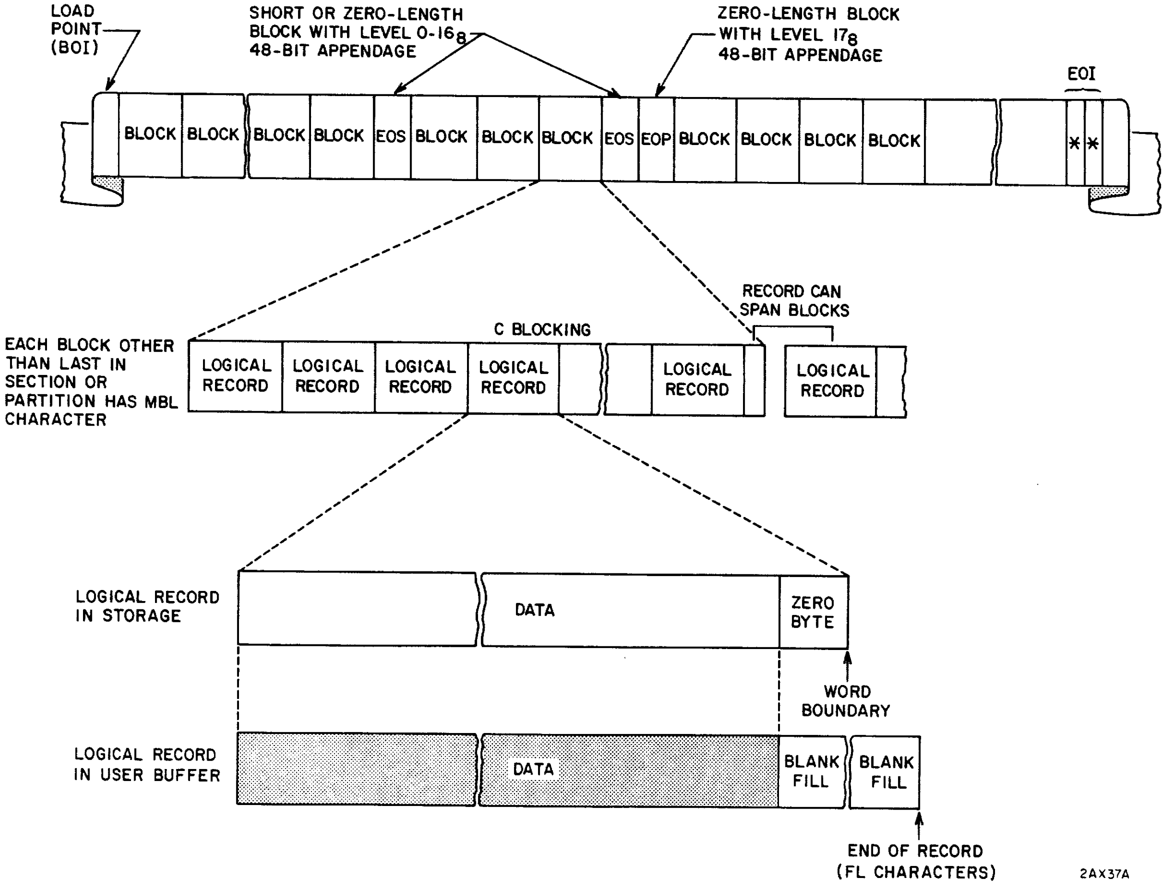
Figure D-4. Padding of Z Records

Z-record specification requires two parameters on the FILE statement, the RT=Z parameter and the FL parameter. FL supplies the decimal count of the fixed length of the user record in characters. It must be large enough to accommodate the largest record on the file.

Usually, the FL parameter will be equivalent to a coded card and (FL=80) or to a print line (FL=137). FL must be large enough to accommodate the largest record.

When Z records are blocked according to block types E and K, partition delimiters are in the form of tapemarks; section delimiters are not possible. When Z records are C-blocked, a section delimiter consists of a short or zero-length block appended by a 48-bit level number where level is $0\text{-}16_8$. A partition delimiter consists of a 48-bit level 17_8 appendage.

When generating Z records, it is the user's responsibility to assure that two display code colons do not occur in the lowest order character positions of a word causing a superfluous record delimiter. If the application has records with many trailing blanks, Z records can provide savings in storage because less data is stored when blanks are removed. However, processing of Z records is slower than processing of W records. On magnetic tape, Z records can be recorded in binary mode only. When using formatted reads and writes, the maximum record size is set to 150 by the FORTRAN object-time routines.



LOAD POINT (BOI)
SHORT OR ZERO-LENGTH BLOCK WITH LEVEL $0\text{-}16_8$ 48-BIT APPENDAGE
ZERO-LENGTH BLOCK WITH LEVEL 17_8 48-BIT APPENDAGE
EOI
BLOCK
BLOCK
BLOCK
BLOCK
EOS
BLOCK
BLOCK
BLOCK
EOS
EOP
BLOCK
BLOCK
BLOCK
BLOCK
*
*
RECORD CAN SPAN BLOCKS
C BLOCKING
EACH BLOCK OTHER THAN LAST IN SECTION OR PARTITION HAS MBL CHARACTER
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD IN STORAGE
DATA
ZERO BYTE
WORD BOUNDARY
LOGICAL RECORD IN USER BUFFER
DATA
BLANK FILL
BLANK FILL
END OF RECORD (FL CHARACTERS)
2AX37A

Figure D-5. Z Zero Byte Records with C Blocking

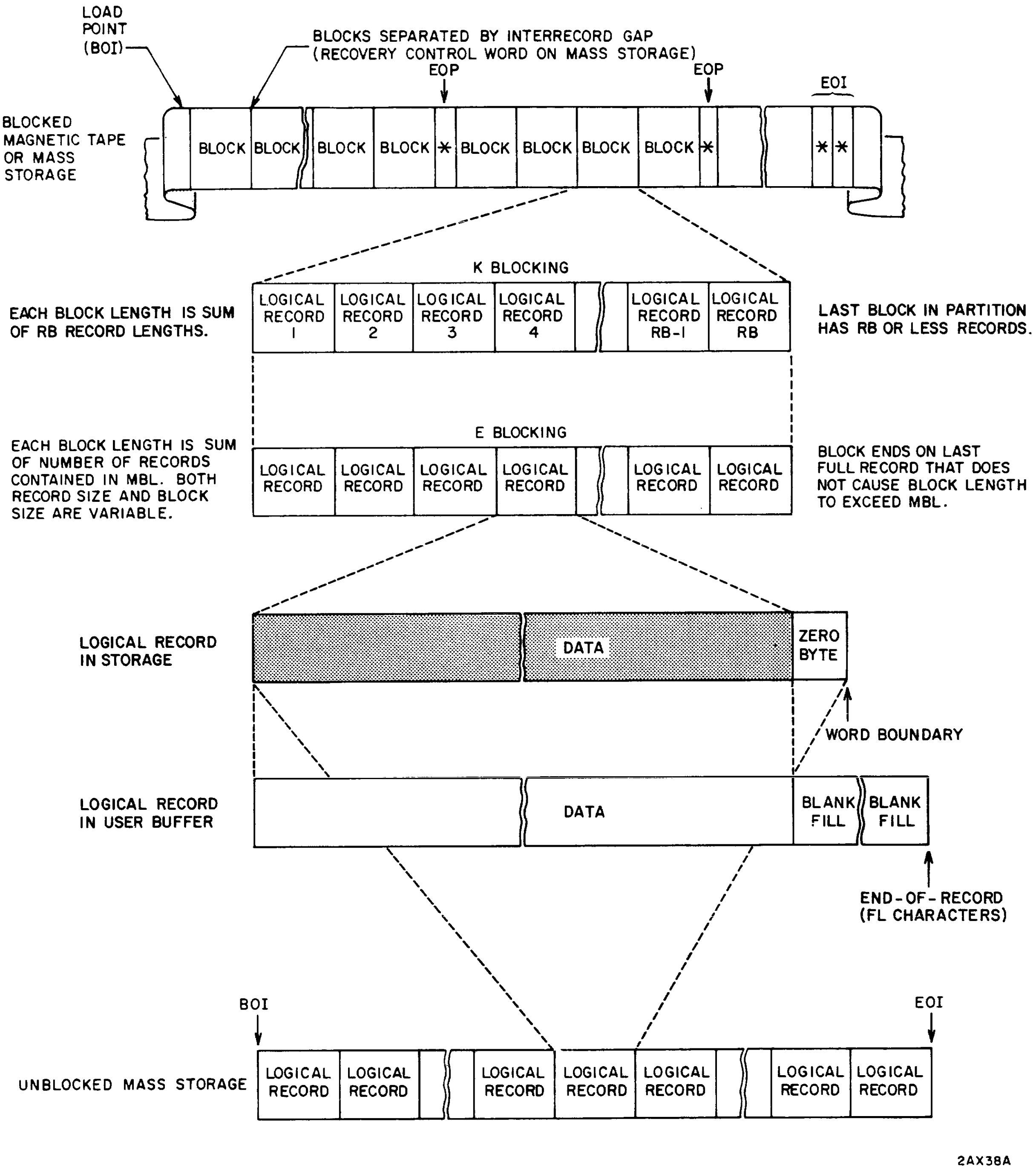
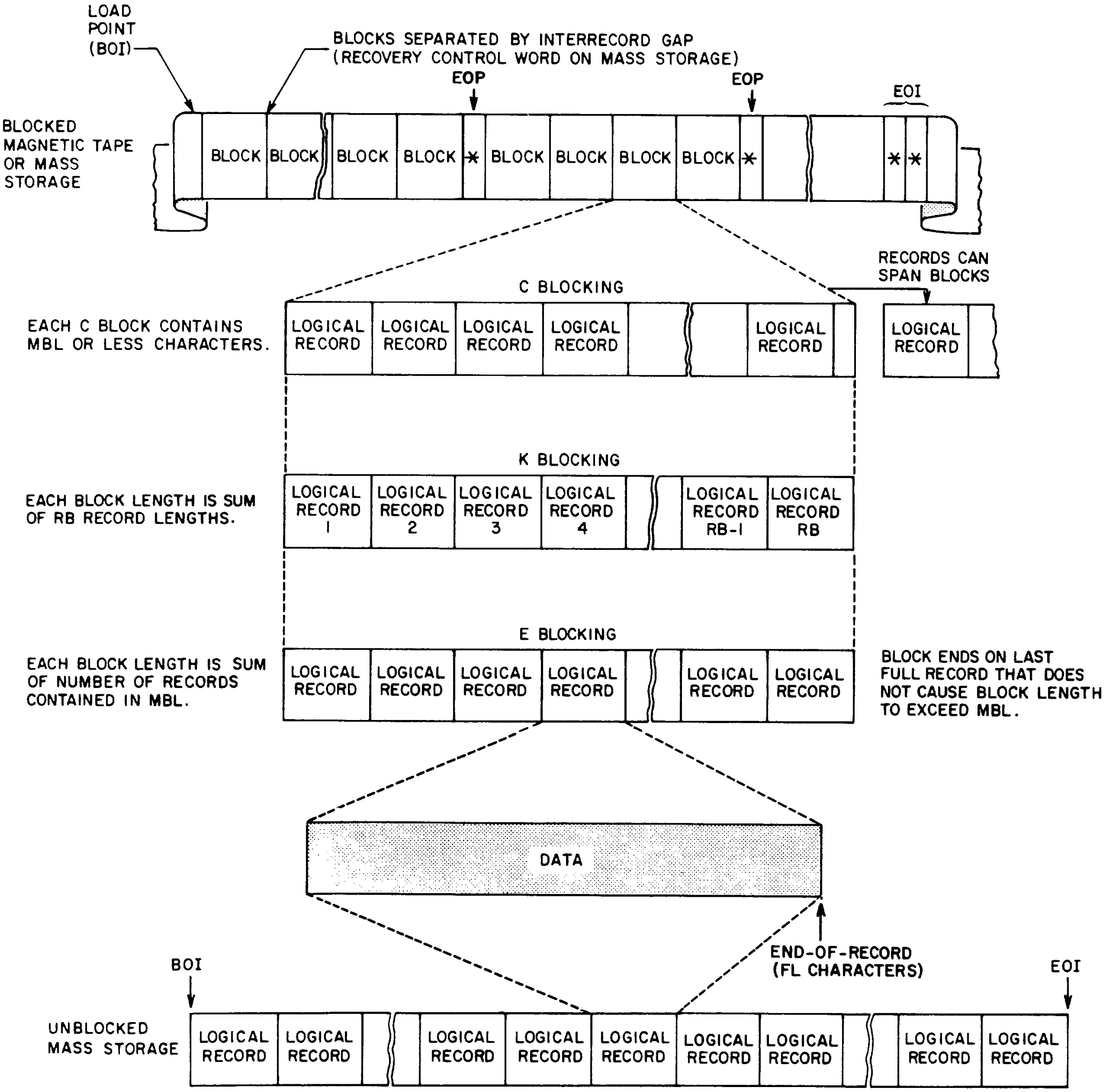

LOAD POINT (BOI)
BLOCKS SEPARATED BY INTERRECORD GAP (RECOVERY CONTROL WORD ON MASS STORAGE)
EOP
EOP
EOI
BLOCKED MAGNETIC TAPE OR MASS STORAGE
BLOCK
BLOCK
BLOCK
BLOCK
*
BLOCK
BLOCK
BLOCK
BLOCK
*
*
*
K BLOCKING
EACH BLOCK LENGTH IS SUM OF RB RECORD LENGTHS.
LOGICAL RECORD 1
LOGICAL RECORD 2
LOGICAL RECORD 3
LOGICAL RECORD 4
LOGICAL RECORD RB-1
LOGICAL RECORD RB
LAST BLOCK IN PARTITION HAS RB OR LESS RECORDS.
E BLOCKING
EACH BLOCK LENGTH IS SUM OF NUMBER OF RECORDS CONTAINED IN MBL. BOTH RECORD SIZE AND BLOCK SIZE ARE VARIABLE.
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
BLOCK ENDS ON LAST FULL RECORD THAT DOES NOT CAUSE BLOCK LENGTH TO EXCEED MBL.
LOGICAL RECORD IN STORAGE
DATA
ZERO BYTE
WORD BOUNDARY
LOGICAL RECORD IN USER BUFFER
DATA
BLANK FILL
BLANK FILL
END-OF-RECORD (FL CHARACTERS)
BOI
EOI
UNBLOCKED MASS STORAGE
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
2AX38A

Figure D-6. Z Zero Byte Records Unblocked or with K and E Blocking

F RECORDS

Fixed length (F) records are commonly used in the computer industry and are the most efficient for COBOL to handle in terms of CPU utilization and throughput.

F records (Figure D-7) are the same as 7600 SCOPE 1.1 E-mode files. F record type is required for the SCOPE 2 direct (DR) file organization. On an output request such as a FORTRAN WRITE or BUFFER OUT, the record manager takes a fixed number of characters (FL) from the user buffer and writes them on the file. On an input request such as a FORTRAN READ or BUFFER IN, the record manager reads the next FL characters on the file and places them in the user buffer.



LOAD POINT (BOI)
BLOCKS SEPARATED BY INTERRECORD GAP (RECOVERY CONTROL WORD ON MASS STORAGE)
EOP
EOP
EOI
BLOCKED MAGNETIC TAPE OR MASS STORAGE
BLOCK
BLOCK
BLOCK
BLOCK
*
BLOCK
BLOCK
BLOCK
BLOCK
*
*
*
RECORDS CAN SPAN BLOCKS
C BLOCKING
EACH C BLOCK CONTAINS MBL OR LESS CHARACTERS.
LOGICAL RECORD
K BLOCKING
EACH BLOCK LENGTH IS SUM OF RB RECORD LENGTHS.
LOGICAL RECORD 1
LOGICAL RECORD 2
LOGICAL RECORD 3
LOGICAL RECORD 4
LOGICAL RECORD RB-1
LOGICAL RECORD RB
E BLOCKING
EACH BLOCK LENGTH IS SUM OF NUMBER OF RECORDS CONTAINED IN MBL.
BLOCK ENDS ON LAST FULL RECORD THAT DOES NOT CAUSE BLOCK LENGTH TO EXCEED MBL.
DATA
END-OF-RECORD (FL CHARACTERS)
BOI
EOI
UNBLOCKED MASS STORAGE
LOGICAL RECORD

2AX39A

Figure D-7. F Fixed Length Record Format

F records are generated by COBOL object time routines according to the File Definition entry.

Specification of F records requires the RT and FL parameters on the FILE statement. FL supplies a decimal count of the fixed length in characters.

On magnetic-tape, F records can be recorded in either binary or coded mode.

Section delimiters are not possible; partition delimiters are equivalent to tapemarks on blocked files.

D RECORDS

Decimal count (D) records are provided primarily for COBOL usage.

D records (Figure D-8) are handled by the record manager as follows. On an output request (for example, a FORTRAN WRITE or BUFFER OUT) the record manager extracts the decimal character count from a length field placed in the data by the user, and writes that number of characters on the file.

On an input request, the record manager again looks at the length field and places the specified number of characters in the user buffer.

To use D records, you must specify the following FILE statement parameters:

RT=D   Specifies record type as D

LL=m   m is the length in characters (1 to 6) of the length field in each record. The maximum record size allowed is determined by the LL parameter. If LP is 0, no data precedes length field. If LL=6, the maximum record size is 999,999. If it is 5, the maximum record size is 99,999, etc. However, MRL takes precedence if it is less than the maximum allowed for the field.

LP=n   n is the beginning position in characters of the length field. The first character in the record is numbered zero. LP+LL must not exceed the record length.

The decimal size must be in the length field right-justified in display code with display code zero fill. The record size includes the length field.

Section delimiters are not possible; partition delimiters are equivalent to tapemarks on blocked files.

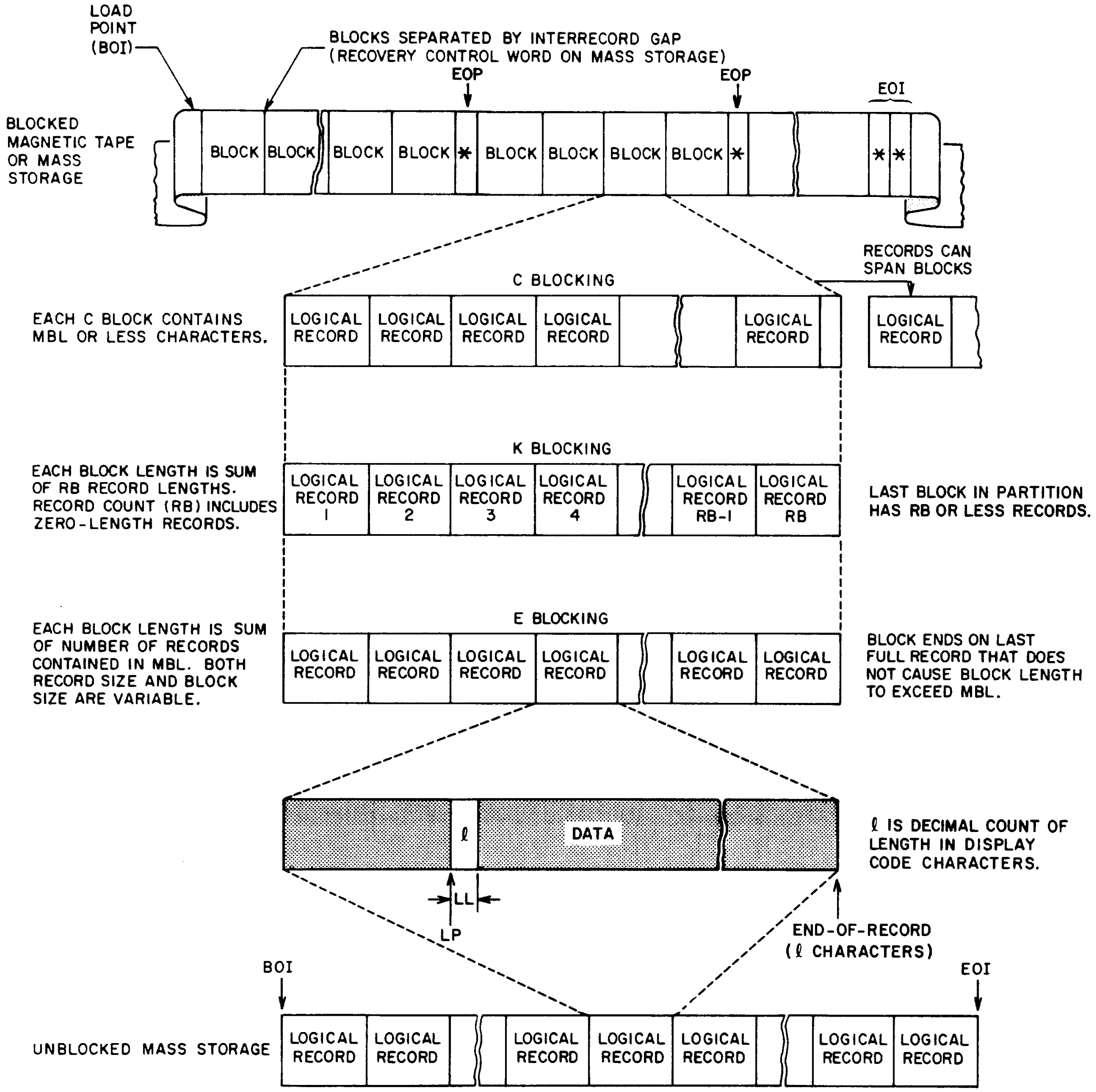

LOAD POINT (BOI)
BLOCKS SEPARATED BY INTERRECORD GAP (RECOVERY CONTROL WORD ON MASS STORAGE)
EOP
EOP
EOI
BLOCKED MAGNETIC TAPE OR MASS STORAGE
BLOCK
BLOCK
BLOCK
BLOCK
*
BLOCK
BLOCK
BLOCK
BLOCK
*
*
*
RECORDS CAN SPAN BLOCKS
C BLOCKING
EACH C BLOCK CONTAINS MBL OR LESS CHARACTERS.
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
K BLOCKING
EACH BLOCK LENGTH IS SUM OF RB RECORD LENGTHS. RECORD COUNT (RB) INCLUDES ZERO-LENGTH RECORDS.
LOGICAL RECORD 1
LOGICAL RECORD 2
LOGICAL RECORD 3
LOGICAL RECORD 4
LOGICAL RECORD RB-1
LOGICAL RECORD RB
LAST BLOCK IN PARTITION HAS RB OR LESS RECORDS.
E BLOCKING
EACH BLOCK LENGTH IS SUM OF NUMBER OF RECORDS CONTAINED IN MBL. BOTH RECORD SIZE AND BLOCK SIZE ARE VARIABLE.
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
BLOCK ENDS ON LAST FULL RECORD THAT DOES NOT CAUSE BLOCK LENGTH TO EXCEED MBL.
ℓ
DATA
ℓ IS DECIMAL COUNT OF LENGTH IN DISPLAY CODE CHARACTERS.
LL
LP
END-OF-RECORD (ℓ CHARACTERS)
BOI
EOI
UNBLOCKED MASS STORAGE
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
LOGICAL RECORD
2AX40A

Figure D-8. Decimal Count Record Format

R RECORDS

Record mark (R) records (Figure D-9) are a COBOL record type. On output, the record manager takes the characters up to and including the record mark character from the user buffer and writes them on the file. On input, the record manager reads the record including the record mark character and places it in the user buffer.

R-type records are variable length. The largest must not exceed the maximum record length (MRL) specified in the FIT.

The conventional record mark character is] . This is the default for COBOL and for 6000 record manager. 7600 record manager, however, defaults to a colon, which is 00 in display code.

If you are a COBOL programmer and wish to change the record mark character, or if you are a FORTRAN programmer and wish to read a file that has a record mark character other than colon, you can use the RMK parameter on the FILE statement.

NOTE

> If the FILE statement changes the character, the data name RECORD-MARK cannot be used to move the character to an output record area.

In the RMK specification, use the decimal or octal equivalent (octal value is suffixed with B) of the display code value for the desired record mark character. Thus, to specify the] character, use either RMK=50 or RMK=62B. Use RT=R to specify R record type. When using R records, it is the user's responsibility to assure that the record mark character does not occur elsewhere in the record.

When using R type records, it is possible to define a minimum record length (MNR=n characters) which allows the record mark character to occur before the minimum character position without being detected by the record manager. Using MNR also speeds up processing because all the records are greater than a certain number of characters. Record manager begins searching for the record mark character at the character following the nth character. If no MNR is specified, record manager begins searching with the first character in each record, or in the case of COBOL programs, according to the RECORD CONTAINS clause.

R type records can be recorded in either binary or coded mode on magnetic tape.

Section delimiters are not possible; partition delimiters are equivalent to tapemarks on blocked files.

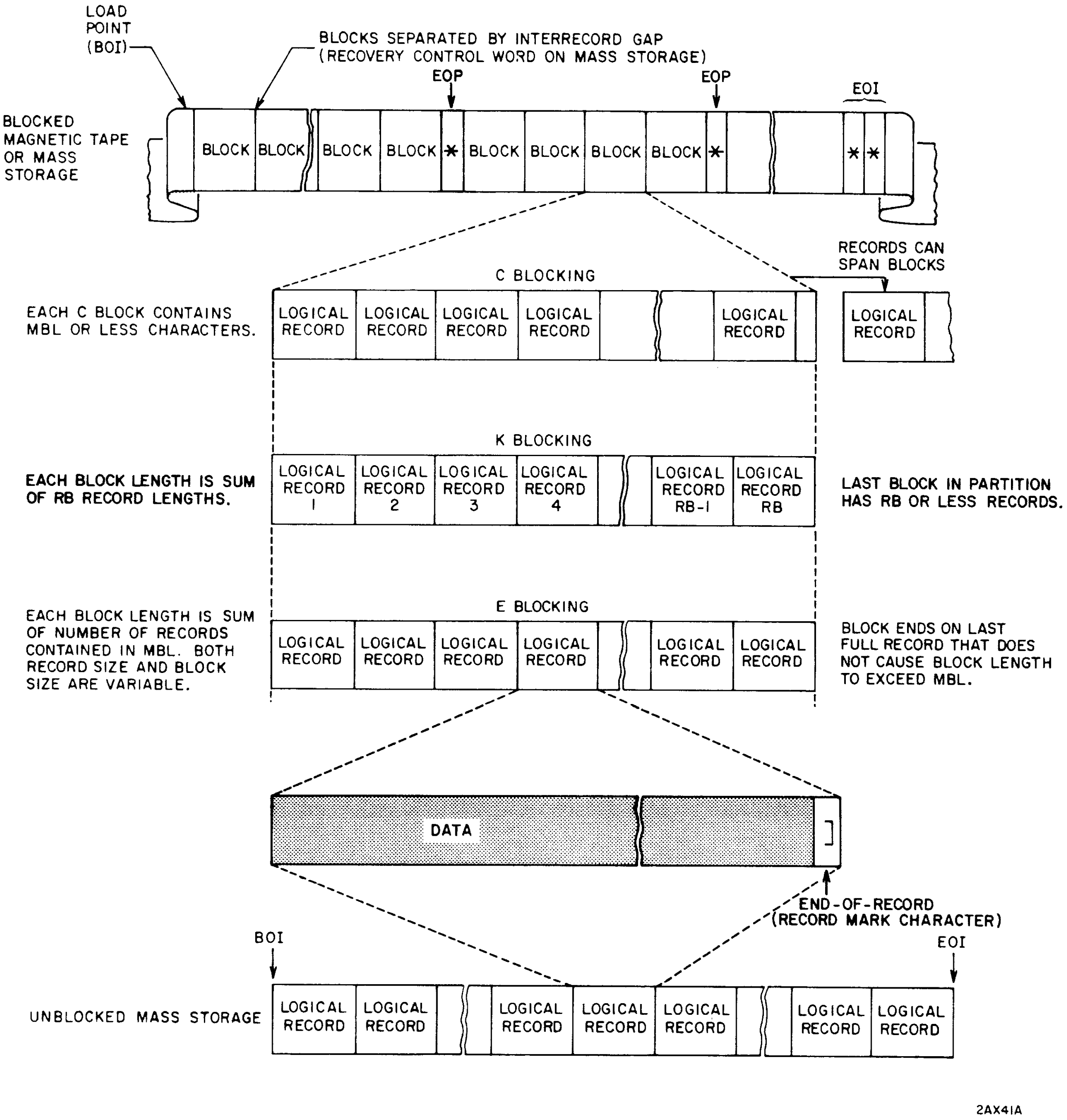


Figure D-9. R Record Mark Character Record Format

T RECORDS

Trailer (T) records (Figure D-10) are the most complex record type. T records are specified by COBOL programs as shown in Table 3-2.

FORTRAN programs using T-type records require a file statement with the following parameters specified.

RT=T Specifies T record type

HL=hl Length of header in characters

TL=tl Length of each trailer in characters

CL=cl Length of count field in characters (1 to 6)

CP=cp Beginning character position in header of count field. CP+CL must not exceed the header length. The first character is numbered 0.

No data precedes the trailer count field if CP=0. The number of fixed length trailers must be inserted into the trailer count field in display coded decimal, right-justified with display code zero fill. 6000 record manager allows blank fill; 7600 record manager does not.

The size of the record is determined by the header length plus the sum of the trailer lengths. This value must not exceed the maximum allowed for a record (MRL) set in the FIT.

On an output request, the record manager determines where the count field is from the FIT, takes the header length plus n times the trailer length characters, and writes them onto the file.

On input, it performs a similar operation to read the record and places it in the user buffer. Section delimiters are not possible; partition delimiters are equivalent to tapemarks on blocked files.

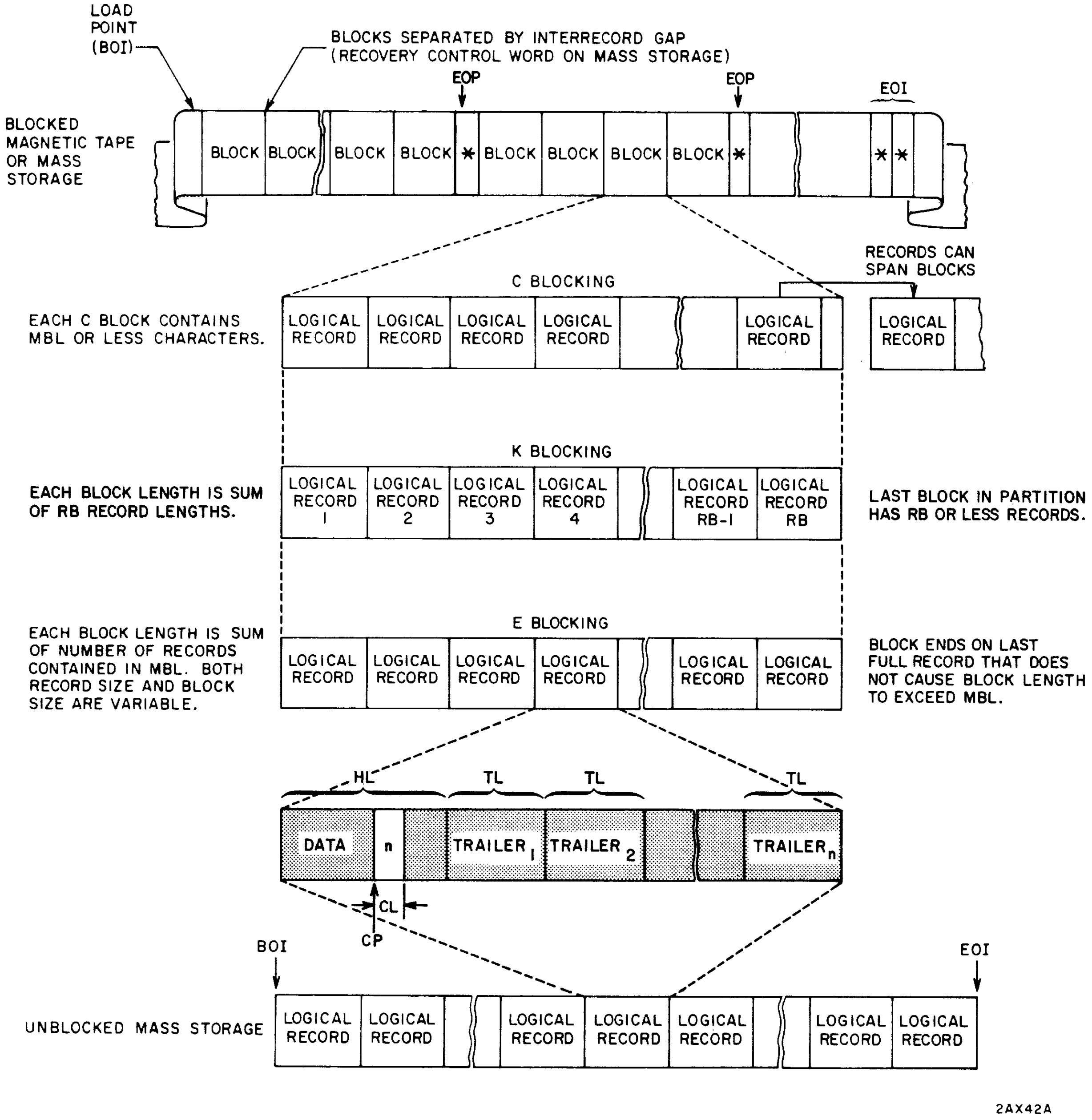


Figure D-10. T Trailer Count Record Format

U RECORDS

The undefined record type (U) is commonly used in the computer industry. It is sometimes referred to as "universal" format.

For U format records (Figure D-11), the record manager receives no definition of what to interpret as a record. Thus, on an unblocked, C blocked, or E blocked file, the entire file consists of a single U record on input.

In the special case of K blocking with one record per block, the record manager uses block delimiters as end-of-record delimiters thus giving the U records some definition. U records can be generated in a COBOL program depending on the RECORD CONTAINS and BLOCK CONTAINS clauses.

To specify record type as U, set the RT=U parameter on the FILE statement.

A file is often defined as U when no other definition is applicable. For example, the COPYXS routine uses a U description to convert X mode tapes to S type records. For U-type records, C blocking and E blocking are possible when accessed through a COMPASS language program. These combinations are not illustrated.

Section delimiters are not possible; partition delimiters are equivalent to tapemarks on blocked files.

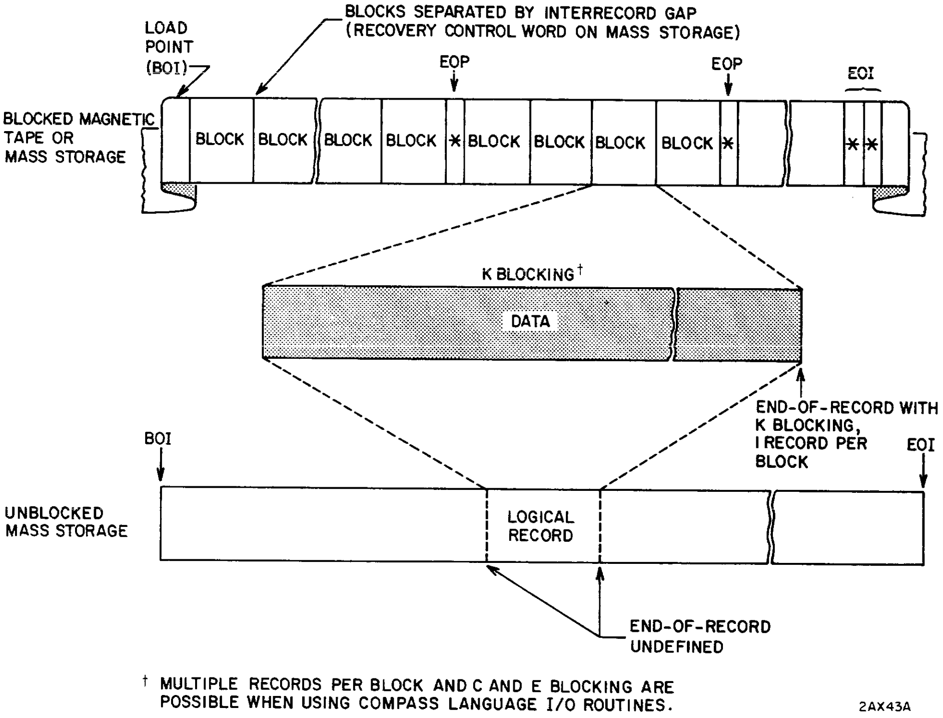

BLOCKS SEPARATED BY INTERRECORD GAP
(RECOVERY CONTROL WORD ON MASS STORAGE)
LOAD POINT (BOI)
EOP
EOP
EOI
BLOCKED MAGNETIC TAPE OR MASS STORAGE
BLOCK
BLOCK
BLOCK
BLOCK
*
BLOCK
BLOCK
BLOCK
BLOCK
*
* *
K BLOCKING†
DATA
END-OF-RECORD WITH K BLOCKING, I RECORD PER BLOCK
BOI
EOI
UNBLOCKED MASS STORAGE
LOGICAL RECORD
END-OF-RECORD UNDEFINED
† MULTIPLE RECORDS PER BLOCK AND C AND E BLOCKING ARE POSSIBLE WHEN USING COMPASS LANGUAGE I/O ROUTINES.
2AX43A

Figure D-11. U Undefined Record Format

USING RECORD MANAGER FOR FILE FORMAT CONVERSION E

A file format conversion, that is, change in record type, block type, or file organization, occurs when the description for the input file for a copy differs from the description for the output file. This feature is not compatible with SCOPE 3.4 for which the copy utilities do not recognize FILE statements.

HOW COPY CONVERTS FILES

Any of the COPY statements cause the copy routines to be loaded and executed. Neither the input file nor the output file is repositioned before its use. The copy routine opens the files and establishes a buffer in SCM to use for the copy, as described in Section 10. Assuming that automatic core memory is in effect, the size of the buffer is by default 1000_8 words (5120 characters) or is determined by whichever MRL or FL is larger for the two files. The MRL may be the maximum length of a partial record rather than the entire record. Setting MRL, except when using R-type records or when converting from S, X, Z, or R record types to W record types, does not limit the maximum size of the records that can be handled by the copy. If output record type is W, the copy must be able to determine full record size before writing out any of the record because the W control word precedes the record. Thus, if input record type is S, X, Z, or R, the MRL must be large enough to accommodate the largest record on the input file. If the buffer is inadequate for the copy, the job is terminated accompanied by the message EXCESS DATA in the dayfile.

The copy routine gets all or part of a record from the input file and places the data in the SCM buffer. Any recovery control words, internal control words, W control words, zero bytes, or level number appendages are removed. That is, normal record processing performed by the record manager takes place.

The copy routine checks status to determine whether EOR, EOS, EOP, or EOI has occurred. If EOR has not occurred, the partial record is put in the output file. In writing out the record, the record manager adds any recovery control words, internal control words, W control words, zero bytes, or level number appendages required by the format defined for the output file. In the case of D, T, and R output records, the count field or record mark character must be in the input data; it is not generated or altered by the copy routine. The record manager uses the count or record mark character to determine how much data to write.

Upon encountering EOR, the copy routine writes the remainder of the record and increments the record count. If the copy is COPYR, the copy terminates after n records have been copied or if a higher-order delimiter is encountered on input. If the copy is COPYS, COPYBR, or COPYCR, the copy terminates after n sections have been copied or a higher order delimiter is encountered. A COPYS of S record considers each record a section. If the copy is COPYP, COPYCF, or COPYBF, the copy terminates when n partitions or an EOI is encountered. Upon completion of the copy, the copy routine writes an EOS if copying sections or an EOP if copying partitions.

PROCEDURE FOR CONVERTING FILE FORMATS

1. Describe your input file using a FILE statement shown in the accompanying tables, if necessary.

2. Describe your output file using a FILE statement shown in the accompanying tables, if necessary. Record type cannot be X.

3. The size of the buffer used for the copy/conversion is determined by whichever maximum record length (MRL) or fixed length (FL) is larger for the two files. The default MRL is 5120 characters (1000_8 words) if neither of the files specifies MRL or FL. The buffer size is a factor in the following cases.

   a. If input or output record type is T or D, MRL must be large enough to include the count field.

   b. If output record type is W, the copy must be able to determine the record size before writing the record. This means that MRL must be large enough to incorporate the largest record on the input file when the input record type is S, X, Z, or R.

   c. If only one file specifies MRL or FL, that value sets the size of the buffer for both files.

NOTE

If input file is INPUT and output FL/MRL is 80, input data may exceed MRL due to JANUS processing as described in Section 9.

4. Select the copy routine you wish to use as you would for an exact copy.

W RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	None; defaults all apply	1,2,3,9
	I	FILE(lfn,BT=I)	1,4,10
	C	FILE(lfn,BT=C)	1,5,10
	K	FILE(lfn,BT=K)	1,6,10
	E	FILE(lfn,BT=E)	1,5,10
WA	Unblocked	FILE(lfn,FO=WA)	1,3,7
LB	Unblocked	FILE(lfn,FO=LB)	1,3,7,8,9

Notes and Rules

1. The following delimiters are recognized on input and recreated on output. W control words are removed from the data before it is placed in the buffer and are added to data when output is W record type.

EOR	Type 0 W control word (neither flag bit nor delete bit set)
EOS	Type 3 W control word (flag bit and delete bit both set)
EOP	Type 2 W control word (flag bit set; delete bit not set)
EOI	Input on magnetic tape: a pair of tapemarks or a tapemark, EOF1 label, and a double tapemark Output: a double tapemark

Section delimiters will be lost if input type is W and output type does not support sections (not W or not Z with C blocking).

Partition delimiters will be lost if input type is W and output file type does not support partitions (that is, output type not W or not blocked sequential).

Usually, W records cannot be converted to S records because S records must be full words of data.

If input file contains on S, X, Z, or R record that exceeds 5120 characters, the FILE statement for input file or output file must specify MRL to guarantee that the buffer for the copy is large enough to hold the record. Maximum record size allowed is determined by amount of SCM available.

If input record type is D or T, MRL must be large enough to encompass the count field.

The largest record that can be converted to W format is 2,621,420 characters.

Recovery control words are removed from blocked files on input and reinserted on output if the output file is to be blocked.

If input file is W records, deleted W records (Type 1 W control word) are not copied. On a W-to-W copy, control words may not be exactly duplicated.

2. The unblocked W record file is the most important format in the SCOPE 2 system. It is the only format recognized by the loader and is the only format that can be used for unit record files.

   When input to the loader originates on magnetic tape or when you wish to go tape-to-punch or tape-to-print, you must copy the tape file to an unblocked W file which can then be loaded, punched, or printed.

3. This file type cannot be used with a STAGE or magnetic tape REQUEST statement. For SQ files, the file will become blocked (see I blocked file). For WA or LB files, blocking is illegal.

4. BT=I is not required if REQUEST MT or STAGE is used. This is the only file description using I blocking. Default block size is 5120 characters and is the only size allowed by SCOPE 3.4. Maximum block length (MBL) must be a multiple of 10 characters (full words).

5. The default maximum block length (MBL) is 5120 characters. MBL must be a multiple of 10 characters (full words). K and E blocking are not commonly used with W records because neither allows records to span blocks.

6. Default records per block (RB) is 1; default maximum record length is 5120 characters. MBL must be a multiple of 10 characters (full words); MBL=MRL x RB. Zero-length W control words are also counted as records (EOS and EOP).

7. An attempt to copy/convert a WA or LB file results in an informative message. Since deleted records are removed, an index for a word addressable file will be invalidated by the copy when records are deleted.

8. This is the only file type using FO=LB.

9. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

10. On magnetic tape, W records can be recorded in binary mode only.

S RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	C	FILE(lfn,RT=S)	See below

Notes and Rules

Default blocking type is always C when record type is S.

S records are also known as 6000 SCOPE standard and 7600 SCOPE 1.1 I-mode. An S record must be a multiple of 10 characters. Copy aborts if output is S and input record is not a multiple of 10 characters. Thus, copying from INPUT to a file with record type S is not possible.

Default block size (MBL) is 5120 characters. This is the equivalent of a 6000 SCOPE physical record unit (PRU) for S/L devices. MBL must be a multiple of 10 characters (full words).

If the file is being converted to W format, either the input file or the output file must specify MRL to guarantee that the SCM buffer for the copy is large enough to hold the record. Maximum record size allowed is determined by amount of SCM available.

An S record that contains display code with zero-byte delimiters can be redefined as Z record type. On magnetic tape, S records can be recorded in binary mode only (odd parity).

When COPYS or its equivalent (COPYBR/COPYCR) is used, each S record is interpreted as a section. For all other copies, each S record is interpreted as a record. That is, for all but COPYS, COPYBR, and COPYCR, the following delimiters are accepted on S record input.

EOR	A short or zero-length block with a level 0 through 16_8 48-bit appendage
EOP	A zero-length block with a level 17_8 48-bit appendage
EOI	A pair of tapemarks or a tapemark, EOF1 label, and two tapemarks

For S record output, the following delimiters are generated.

EOR	A short or zero-length block with a level 0 to 16_8 48-bit appendage. Levels 1 to 16_8 are not lost in copying S and Z records.
EOP	A zero-length block with a level 17_8 48-bit appendage
EOI	Two tapemarks

On mass storage delimiter, information is maintained in recovery control words. Recovery control words and 48-bit appendages are not transferred to the copy buffer. They are removed from input or inserted on output. Corresponding status is returned to the copy routines.

X RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	C	FILE(lfn, RT=S)	See Notes and Rules

Notes and Rules

X records can be input only. You cannot copy to X. Default blocking type is always C when record type is X.

X records are also known as 6000 X-MODE and 7600 SCOPE 1.1 X Format. They are not supported on SCOPE 3.4.

An X record must be a multiple of 10 characters; otherwise COPY aborts. Default block size (MBL) is 5120 characters. This is equivalent to a 6000 SCOPE physical record unit for an S/L device. MBL must be a multiple of 10 characters.

If the file is being converted to W format, either the input file or the output file must specify MRL to guarantee that the SCM buffer for the copy is large enough to hold the record. Maximum record size allowed is determined by the amount of SM available.

The following delimiters are recognized on input.

EOR A short or zero-length block

EOP A single tapemark

EOI A pair of tapemarks or a tapemark, EOF1 label, and two tapemarks

Partition delimiters on the input file will be lost if the output file type does not support partitions (not W or not blocked sequential). On blocked mass storage, tapemarks are maintained in recovery control words.

SCOPE 2 includes a utility named COPYXS designed solely to convert X-mode tapes to S record type.

COPYXS(lfn$_{in}$, lfn$_{out}$, n)

Parameter n is a count of the partitions to be copied. A tapemark is interpreted as an end-of-partition.

COPYXS assumes that the input file is described as follows:

```
FILE(TAPE1, RT=U)
```

By default, then, BT=K, RB=1, and MBL=5120.

The input file can be renamed on the COPYXS statement; otherwise, any change to the description causes the job to abort. COPYXS assumes that the output file is described as follows:

```
FILE(TAPE2, RT=S)
```

The file can be renamed on the COPYXS statement; use a FILE statement to alter any other characteristics.

Following the copy, both files are closed.

Z RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	FILE(lfn, RT=Z, FL=fl)	1, 2
	C	FILE(lfn, RT=Z, FL=fl, BT=C)	1, 3, 4, 8
	K	FILE(lfn, RT=Z, FL=fl, BT=K)	1, 5, 6, 8
	E	FILE(lfn, RT=Z, BT=E)	1, 4, 6, 8
WA	Unblocked	FILE(lfn, FO=WA, RT=Z, FL=fl)	1, 7

Notes and Rules

1. Fixed length in characters (FL) is required. There is no default. On Z record input, the zero bytes and recovery control words are removed. The data is filled to FL characters with display code blanks.

   On Z record output, the record manager removes trailing blanks from the data until the record is full words of data with a 12-bit zero-byte in the least significant 2 characters of the last word. If FL consists of data with one or no trailing blanks, the record manager adds a word of blanks to accommodate the zero byte. To avoid truncation, FL must be greater than or equal to the input MRL or FL. Truncation results in an informative message. Recovery control words are added on output if the output file is blocked.

2. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

3. Z records with C blocking are a special case and can be redefined as S record type. The following delimiters are recognized on input.

EOR	12 low-order bits of a word are zero.
EOS	A short or zero-length block with a level 0 through 16_8 48-bit appendage
EOP	A zero-length block with a level 17_8 48-bit appendage
EOI	A pair of tapemarks or a tapemark, EOF1 label and two tapemarks

   The following delimiters are generated on output.

EOR	12 low-order bits of a word are zero
EOS	A short or zero-length block with level 0 to 16_8 48-bit appendage. Levels 1 through 16_8 are not lost.
EOP	A zero-length block with a level 17_8 48-bit appendage
EOI	Two tapemarks

   On blocked mass storage, these delimiters are maintained in the form of recovery control words.

4. The maximum block length (MBL) is 5120 by default. It must be a multiple of 10 characters (full words).

5. BT=K is not required if REQUEST MT or STAGE is used.

   Default records per block (RB) is 1; if default MBL is RL x RB. Record length (RB) is usually determined from FL. MBL must be a multiple of 10 characters (full words).

6. The following delimiters are recognized on input and recreated on output.

EOR	12 low-order bits of a word are zero	
EOP	A single tapemark	
EOI	Input:	Two tapemarks, or tapemark, EOF1 label and two tapemarks
	Output:	Two tapemarks

Section delimiters will be lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

On blocked mass storage, tapemarks are maintained in recovery control words.

7. An attempt to convert/copy a WA file results in an informative message.

8. On magnetic tape, Z records can be recorded in binary mode (odd parity) only.

F RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	FILE(lfn,RT=F,FL=fl)	1,2,7
	C	FILE(lfn,RT=F,FL=fl,BT=C)	1,3,5
	K	FILE(lfn,RT=F,FL=fl,BT=K)	1,4,5
	E	FILE(lfn,RT=F,FL=fl,BT=E)	1,3,5
WA	Unblocked	FILE(lfn,FO=WA,RT=F,FL=fl)	1,2,6

Notes and Rules

1. Fixed length in characters (FL=fl) is required. There is no default. For output, fl must be greater than or equal to input MRL or FL to avoid truncation. Truncation results in an informative message. When converting to F, if fl on output specifies a record longer than the input record (for example, a W record), the record is filled with blanks if CM=YES and with zeroes if CM=NO.

   F-type records can be converted to S-type records if fl is a multiple of 10 characters.

2. This file type cannot be used with a STAGE or magnetic tape REQUEST statement. For SQ files, the file will become blocked (see K blocked file). For WA files, blocking is illegal.

3. The default maximum block length (MBL) is 5120 characters.

4. BT=K is not required if REQUEST MT or STAGE is used. Default records per block (RB) is 1; default maximum record length is 5120 characters. MBL=MRL x RB.

5. The following delimiters are recognized on input and recreated on output.

EOR	fl characters read/written	
EOP	Single tapemark	
EOI	Input:	Two tapemarks, or tapemark, EOF1 label and two tapemarks
	Output:	Two tapemarks

Section delimiters will be lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

On blocked mass storage, tapemarks are maintained in recovery control words.

6. An attempt to convert/copy a WA file results in an informative message.

7. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

D RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	FILE(lfn, RT=D, LL=m, LP=n)	1, 6
	C	FILE(lfn, RT=D, LL=m, LP=n, BT=C)	1, 2, 3
	K	FILE(lfn, RT=D, LL=m, LP=n, BT=K)	1, 2, 4
	E	FILE(lfn, RT=D, LL=m, LP=n, BT=E)	1, 2, 3
WA	Unblocked	FILE(lfn, FO=WA, RT=D, LL=m, LP=n)	1, 5

Notes and Rules

1. The parameters specifying length of the decimal count field (LL) and position of the field (LP) are required. There are no defaults. The length field length (LL=m) can be 1 to 6 characters. This indirectly limits the size of records. If LL=1, record length can be 1 to 9. If LL is 2, record length can be 1 to 99, etc., to a maximum of 999999 when LL is 6. The decimal count field contains the record length in display code decimal.

   LP + LL must be less than or equal to MRL (the buffer size).

   When reading D records, the record manager gets as many characters from the file as specified by the length field. When generating D records, the record manager writes the number of characters specified by the contents of the length field.

   The LP and LL parameters used to describe the position and size of the length field for the output file, actually describe the location and size of the decimal count field on the input file, regardless of the record type of the input file.

   If the input file is W records, conversion consists of removing the W control words. The input file is basically a D record type file over which the W record structure has been superimposed. If the input file is Z records, conversion consists of removing the zero bytes and padding the records to FL characters and writing out the number of characters indicated by LL. If the LL field specifies a record longer than the input record, the record is filled with whatever is in the buffer.

   If the LL field specifies a record shorter than the input record, truncation occurs accompanied by an informative message.

2. The following delimiters are recognized on input and recreated on output.

   EOR (LL) characters read/written

   EOP Single tapemark

EOI Input: Two tapemarks, or a tapemark, EOF1 label, and two tapemarks
Output: Two tapemarks

On blocked mass storage, a tapemark is maintained in a recovery control word.

Section delimiters are lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

3. Default maximum block length (MBL) is 5120 characters.

4. BT=K is not required if REQUEST MT or STAGE is used. Default records per block (RB) is 1. Default MRL is 5120 characters. MBL = RB x MRL.

5. An attempt to convert/copy a WA file results in an informative message.

6. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

R RECORDS CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	FILE(lfn,RT=R,RMK=char)	1,6
	C	FILE(lfn,RT=R,RMK=char,BT=C)	1,2,3
	K	FILE(lfn,RT=R,RMK=char,BT=K)	1,2,4
	E	FILE(lfn,RT=R,RMK=char,RT=E)	1,2,3
WA	Unblocked	FILE(lfn,FO=WA,RT=R,RMK=char)	1,5

Notes and Rules

1. RMK is usually specified. If not specified, it defaults to 00 which is a display code colon. The conventional character is] , specified in the form RMK=62B. The] is the standard default for 6000/7000 COBOL and for 6000 record manager under SCOPE 3.4.

   When the input file is R record type all the characters from the beginning of the record up to and including the record mark character comprise the record. MRL must be large enough to encompass the entire record or the message UT202 F/R/Z DATA TRUNCATED is issued.

   SCM buffer size is set to the larger MRL for the two files.

   When the output file is R record type, all the characters in the buffer up to and including the record mark character are written out. The record mark character must be in the input record. Truncation of the input record may occur. For example, when converting in the 71st character position, the R record output will consist of 71-character records.

2. The following delimiters are recognized on input and recreated on output.

   EOR Characters up to and including the record mark character are read/written

   EOP Single tapemark

EOI Input: Two tapemarks, or a tapemark, EOF1 label, and two tapemarks
Output: Two tapemarks

On blocked mass storage, a tapemark is maintained in a recovery control word.

Section delimiters will be lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

3. Default maximum block length (MBL) is 5120 characters.

4. BT=K is not required if REQUEST MT or STAGE is used. Default records per block (RB) is 1. Default MRL is 5120 characters. MBL=RB x MRL.

5. An attempt to convert/copy a WA file results in an informative message.

6. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

T RECORD CONVERSIONS

FO	BT	FILE Statement	Notes
SQ	Unblocked	FILE(lfn, RT=T, CP=cp, CL=cl, HL=hl, TL=tl)	1, 6
	C	FILE(lfn, RT=T, CP=cp, CL=cl, HL=hl, TL=tl, BT=C)	1, 2, 3
	K	FILE(lfn, RT=T, CP=cp, CL=cl, HL=hl, TL=tl, BT=K)	1, 2, 4
	E	FILE(lfn, RT=T, CP=cp, CL=cl, HL=hl, TL=tl, BT=E)	1, 2, 3
WA	Unblocked	FILE(lfn, FO=WA, RT=T, CP=cp, CL=cl, HL=hl, TL=tl)	1, 5

Notes and Rules

1. CP, CL, HL, and TL are required. There are no defaults. The count position (CP), starting with 0, plus the count field length (CL) must be less than the header length in characters (HL).

The count field contains the number of trailers in the record. The size of the record is determined from the header length plus the contents of the count field times the trailer length. Thus, if conversion is to W record type, the count field must be in the first portion of the input record placed in the SCM buffer.

When the output file is T record type, the CP, CL, HL, and TL parameters actually describe the count field, trailer length and header length on the input file, regardless of the record type of the input file.

In other words, the input file is basically a T record type file over which some other record structure has been superimposed. For example, if the input file is W records, conversion consists of removing the W control words. If the input file is Z records, conversion consists of padding the records to FL characters, removing the zero bytes, and writing out the number of characters computed as the record size. If the computed size exceeds the input record size, the record is filled with whatever is in the buffer. If the computed size is less than the input record size, truncation occurs.

2. The following delimiters are recognized on input and recreated on output.

   EOR Header length character count plus n time the trailer length characters have been read/written. n is the number of trailers.

   EOP Single tapemark

   EOI Input: Two tapemarks, or a tapemark, EOF1 label, and two tapemarks

   Output: Two tapemarks

   On blocked mass storage, a tapemark is maintained in a recovery control word.

   Section delimiters will be lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

3. Default maximum block length (MBL) is 5120 characters.

4. BT=K is not required if REQUEST MT or STAGE is used. Default records per block (RB) is 1. Default MRL is 5120 characters. MBL=RB x MRL

5. An attempt to convert/copy a WA file results in an informative message.

6. Unblocked sequential files are not supported by 6000 SCOPE 3.4.

U RECORD CONVERSIONS

FO	BT	FILE Statement	Notes and Rules
SQ	Unblocked	FILE(lfn, RT=U, BT)	1
	K	FILE(lfn, RT=U, BT=K)	2

Notes and Rules

1. Unblocked sequential files are not supported by 6000 SCOPE 3.4. The BT parameter can be omitted if the file is already unblocked. Default MRL is 5120 characters.

2. BT=K is not required if REQUEST MT or STAGE is used. Records per block must be 1. This is the default for RB. Default MBL is RB x MRL. Default MRL is 5120 characters. The following delimiters are recognized on input and recreated on output.

   EOR Data between two interrecord gaps, that is, one block

   EOP Single tapemark

   EOI Input: Two tapemarks, or a tapemark, EOF1 label, and two tapemarks

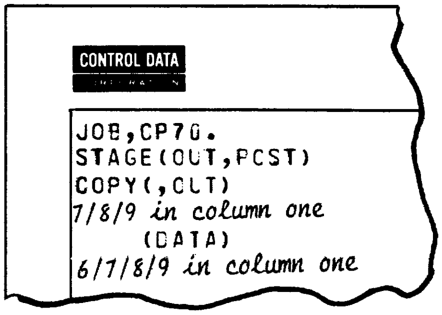
   Output: Two tapemarks

   On blocked mass storage, a tapemark is maintained as a recovery control word.

   Section delimiters are lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

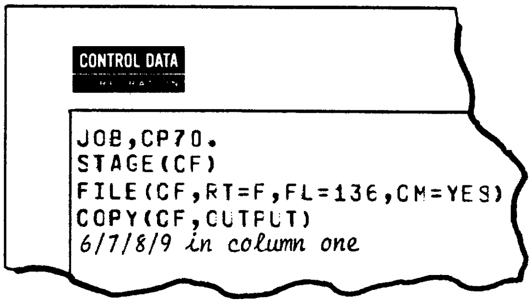
CONVERSION EXAMPLES

1. A very common application of COPY is to block or deblock W record files. This example illustrates a job that copies data on INPUT to a magnetic tape file (it performs a card-to-tape operation). Neither file requires a FILE statement because defaults apply. The only difference between file descriptions is that the input file is unblocked whereas the output file is I-blocked because of the STAGE statement. Remember, however, that the size of each W record depends on whether the cards in the deck are Hollerith, SCOPE binary, or free-form binary.



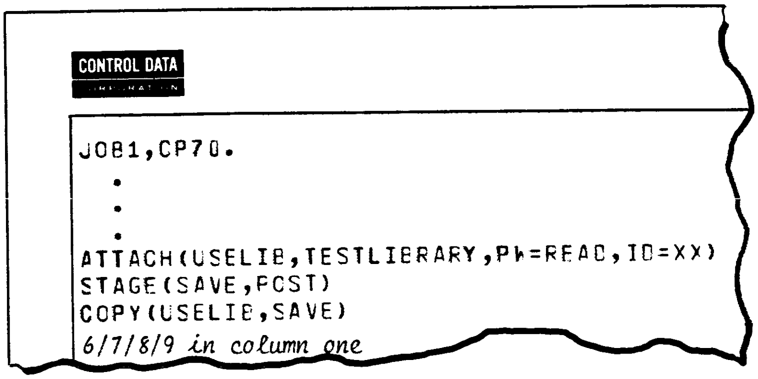
```
JOB,CP70.
STAGE(OUT,POST)
COPY(,OUT)
7/8/9 in column one
     (DATA)
6/7/8/9 in column one
```

2. Example 2 illustrates how to print from an external-coded, even-parity tape recorded one print line per block (Industry standard unblocked). Each print line is 136 characters; the first character of which is a printer control character. Tape density is 556 bits per inch. The file format is also known as E mode for SCOPE 1.1.



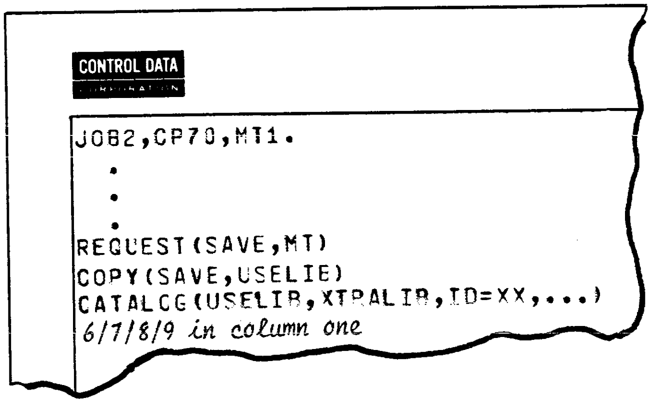
```
JOB,CP70.
STAGE(CF)
FILE(CF,RT=F,FL=136,CM=YES)
COPY(CF,OUTPUT)
6/7/8/9 in column one
```

3. In this example, library USELIB is saved on staged magnetic tape and later loaded on-line, perhaps at another site. USELIB is an unblocked W format sequential file. The first job blocks an unblocked file. The process is reversed in the second job; the file must be unblocked before the loader can load from it.



CONTROL DATA
CORPORATION

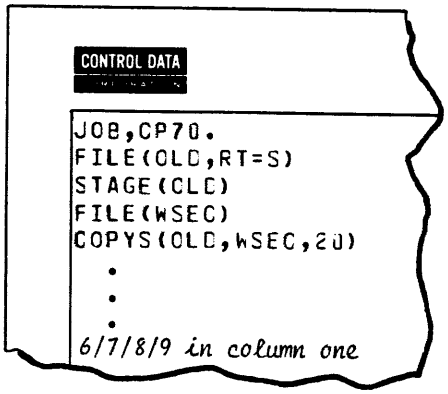
```
JOB1,CP70.
  •
  •
  •
ATTACH(USELIB,TESTLIBRARY,PW=READ,ID=XX)
STAGE(SAVE,POST)
COPY(USELIB,SAVE)
6/7/8/9 in column one
```



CONTROL DATA
CORPORATION

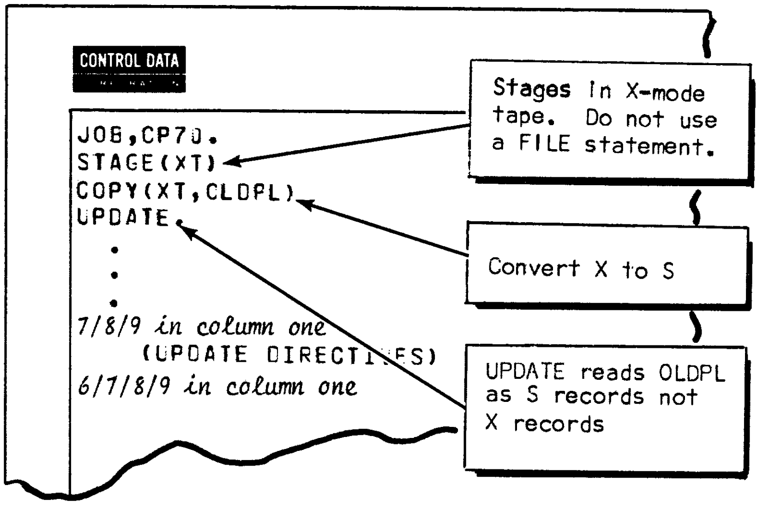
```
JOB2,CP70,MT1.
  •
  •
  •
REQUEST(SAVE,MT)
COPY(SAVE,USELIB)
CATALOG(USELIB,XTRALIB,ID=XX,...)
6/7/8/9 in column one
```

4. This example illustrates an S to W conversion using COPYS. Each S record is a library routine. To be able to be used at all by LIBEDT, each routine must be converted to a section in W format. COMPARE cannot be used following this copy to show equivalence.



```
JOB,CP70.
FILE(OLD,RT=S)
STAGE(OLD)
FILE(WSEC)
COPYS(OLD,WSEC,20)
   •
   •
   •
6/7/8/9 in column one
```

5. In this example, the first X record on tape XTAPE contains coded punched card images. Each image consists of full words of data terminated by a zero-byte in the low-order position of the last word. The data cannot be considered as Z records because section delimiters are in X-record format. Blocking type is C. The job copies XTAPE to DATA, which is in S-record format. DATA is then closed and rewound by the SKIPB statement and then redefined as Z records with C blocking. The Z records are then copied to PUNCHB in unblocked W format. Each of the card images becomes a W record.



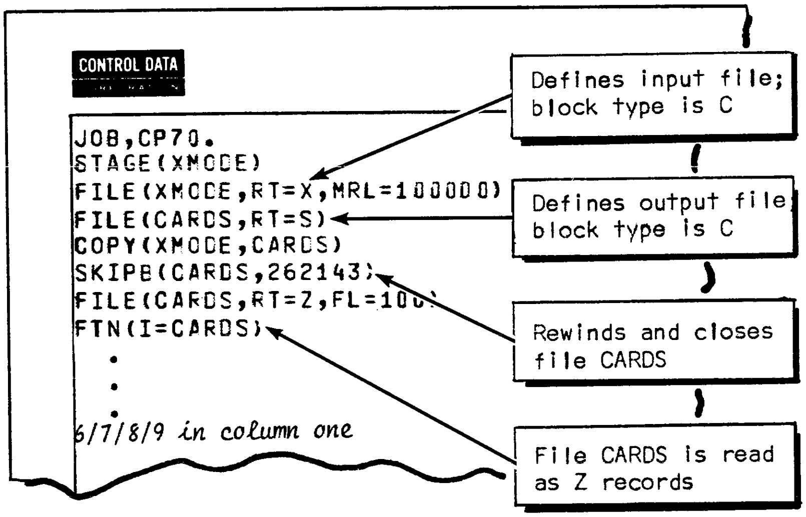
```
JOB,CP70.
STAGE(XT)
COPY(XT,CLDPL)
UPDATE.
   •
   •
   •
7/8/9 in column one
     (UPDATE DIRECTIVES)
6/7/8/9 in column one
```

Stages In X-mode tape. Do not use a FILE statement.

Convert X to S

UPDATE reads OLDPL as S records not X records

6. Here, we have a job that converts a tape in X-record format to a file in S-record format. The X-record tape contains 1000 FORTRAN source card images terminated by zero-byte terminators. Thus, the file can be redefined as Z-type records with FL of 100 for use as input to FORTRAN. The SKIPB(CARDS,262143) statement rewinds CARDS and closes it so that it can be redefined.



```
JOB,CP70.
STAGE(XMODE)
FILE(XMODE,RT=X,MRL=100000)
FILE(CARDS,RT=S)
COPY(XMODE,CARDS)
SKIPB(CARDS,262143)
FILE(CARDS,RT=Z,FL=100)
FTN(I=CARDS)
  •
  •
  •
6/7/8/9 in column one
```

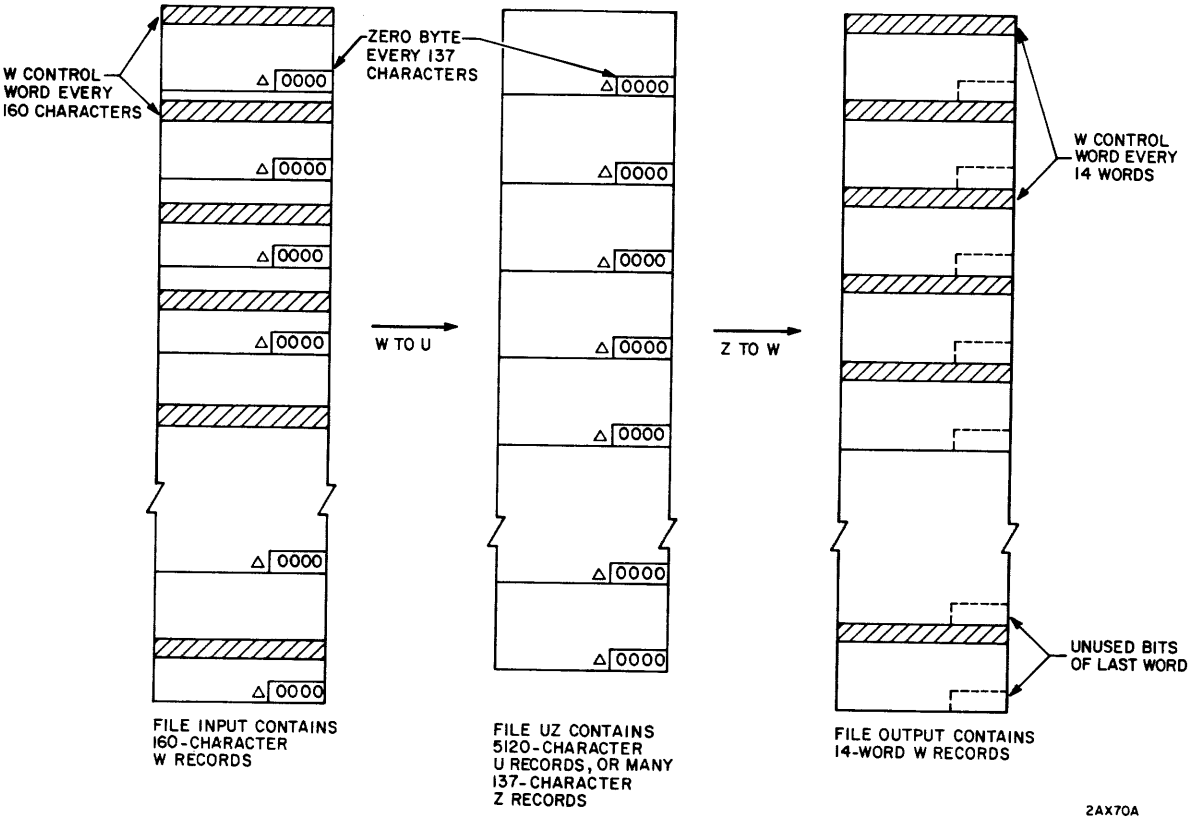
Defines input file; block type is C

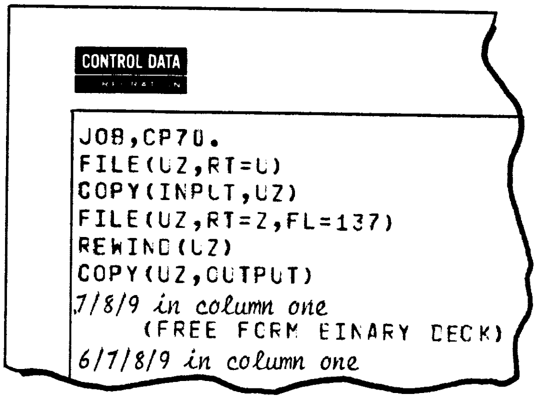
Defines output file block type is C

Rewinds and closes file CARDS

File CARDS is read as Z records

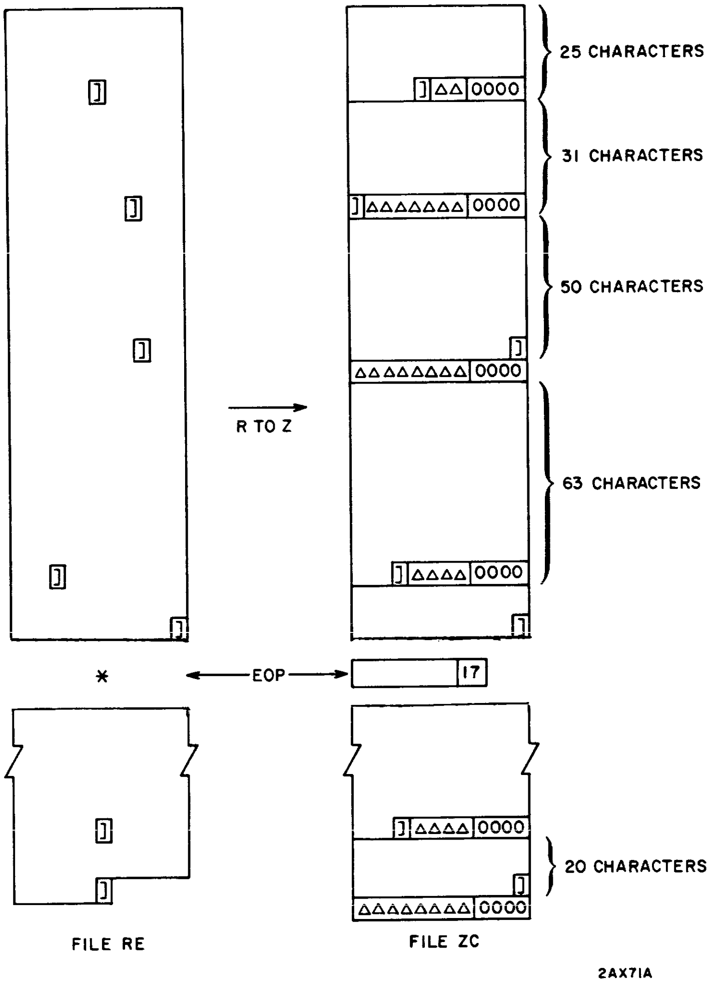
7. In this example, the second section on INPUT is a free-form binary deck that originally consisted of 137-character print lines in the form of Z records. On input, the deck has W control words inserted every 160 characters. The job shows how the W control words are removed through a W to U copy. Then, the U-type file is redefined as Z-type and copied to the OUTPUT file.

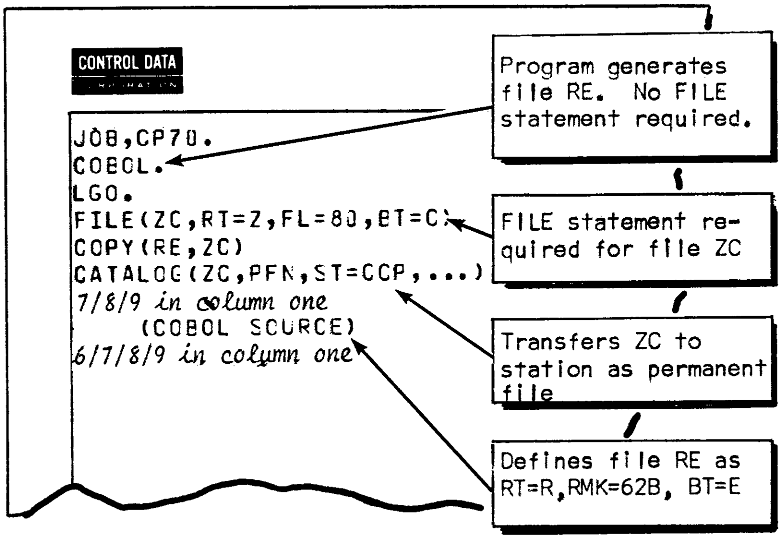

W CONTROL WORD EVERY 160 CHARACTERS
ZERO BYTE EVERY 137 CHARACTERS
Δ 0000
W TO U
Z TO W
W CONTROL WORD EVERY 14 WORDS
UNUSED BITS OF LAST WORD
FILE INPUT CONTAINS 160-CHARACTER W RECORDS
FILE UZ CONTAINS 5120-CHARACTER U RECORDS, OR MANY 137-CHARACTER Z RECORDS
FILE OUTPUT CONTAINS 14-WORD W RECORDS
2AX70A



```
JOB,CP70.
FILE(UZ,RT=U)
COPY(INPUT,UZ)
FILE(UZ,RT=Z,FL=137)
REWIND(UZ)
COPY(UZ,OUTPUT)
7/8/9 in column one
    (FREE FORM BINARY DECK)
6/7/8/9 in column one
```

8. The user has a COBOL-generated file containing R records with E blocking. The record mark character is] . He wishes to convert the file to C-blocked Z records for use as a SCOPE standard permanent file at the 6000 Station (blocking not shown). Each tapemark on input becomes a level 17_8 48-bit appendage (EOP).




CONTROL DATA

```
JOB,CP70.
COBOL.
LGO.
FILE(ZC,RT=Z,FL=80,BT=C)
COPY(RE,ZC)
CATALOG(ZC,PFN,ST=CCP,...)
7/8/9 in column one
     (COBOL SOURCE)
6/7/8/9 in column one
```

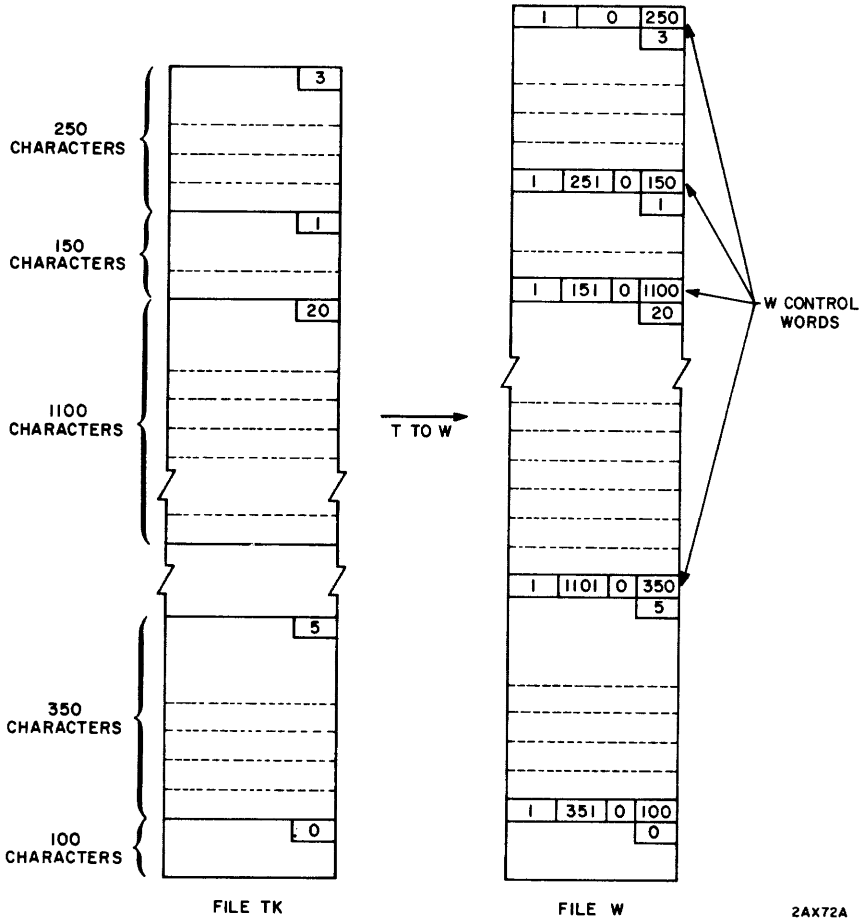
Program generates file RE. No FILE statement required.

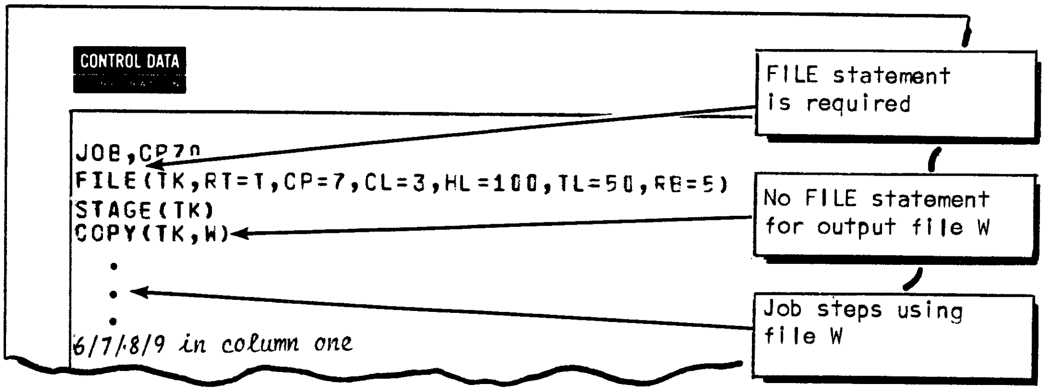
FILE statement required for file ZC

Transfers ZC to station as permanent file

Defines file RE as RT=R,RMK=62B, BT=E

9. This example illustrates use of a copy routine to convert T records to W records. The T records contain a trailer count in the eighth through tenth character positions of each record. This count field is not removed by the conversion. The input file is K blocked; the output file is unblocked. Blocking is not illustrated.




CONTROL DATA

```
JOB,CP70
FILE(TK,RT=T,CP=7,CL=3,HL=100,TL=50,RB=5)
STAGE(TK)
COPY(TK,W)
   •
   •
   •
6/7/8/9 in column one
```

FILE statement is required

No FILE statement for output file W

Job steps using file W

DEFAULT FILE DESCRIPTIONS F

The following table summarizes characteristics of files acceptable by SCOPE 2 and its product set. Except for LIBEDT libraries, the default file organization (FO) is sequential (SQ). Any tape file is assumed to be unlabeled and in binary mode. The default file name (lfn) can be overridden.

System Routine/Program	lfn	RT	BT	Redefinable with FILE Statement
Loader				
Object module	---	W	unbl	No
Core image module	---	W	unbl	No
Utilities				
COPY, COPYS, COPYP, etc.				
Input file	INPUT	W	unbl	Yes; see Appendix E
Output file	OUTPUT	W	unbl	Yes; see Appendix E
COPYSP				
Input file	INPUT	W	unbl	Yes
Output file	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
COPYXS				
Input file	TAPE1	U	K	No
Output file	TAPE2	S	C	No
COMPARE				
File one	OLDLIB	W	unbl	Yes
File two	NEWLIB	W	unbl	Yes
List file	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
DMPFILE				
Input file	INFILE	W	unbl	Yes
List file	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
SKIPF, SKIPB, and BKSP	FILE	W	unbl	Yes; record type can be F, R, or Z with C blocking, or any other record type if BT=K, RB=1
FORTRAN compilers				
Source input file	INPUT	W	unbl	Yes; record type can be Z or F is FL≤90; otherwise, warning message is issued
List output file	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
Object binary file	LGO	W	unbl	Cannot be printed under SCOPE 2 if redefined

System Routine/Program	lfn	RT	BT	Redefinable with FILE Statement
COBOL compiler				
Source input	INPUT	W	unbl	Yes; any block type is acceptable; record type can be F or Z or can be U if RB=1
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
Object binary	LGO	W	unbl	Cannot be loaded under SCOPE 2 if redefined
COMPASS assembler				
Source input	INPUT	W	unbl	Yes; can be blocked. Record type can be Z (coded with FL≤100) or can be S (binary)
Compile file	COMPILE	W	unbl	Yes; can be I-blocked with W record type. Record type can be Z (coded with FL≤100) or S (binary)
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
Object binary	LGO	W	unbl	Cannot be loaded under SCOPE 2 if redefined
UPDATE program				
Old library	OLDPL	W	unbl	Yes; blocked or RT=S
New library	NEWPL	W	unbl	Yes; blocked or RT=S
Compile file	COMPILE	W	unbl	Yes; RT=S if compressed
Source input	INPUT	W	unbl	Yes; can be blocked; record type can be RT=Z, FL≤100.
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
LIBEDT				
Directives input	INPUT	W	unbl	Must be W but can be blocked
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
Libraries	---	W	unbl[†]	Not recommended
Sequential files	---	W	unbl	Yes; can be blocked or RT=S
TRAP				
Directives input	INPUT	W	unbl	Yes; can be RT=Z or F, FL≤100
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
SEGLOAD				
Directives input	INPUT	W	unbl	Yes; can be RT=Z or F, FL≤100
Loader input	---	W	unbl	Cannot be loaded by SEGLOAD if redefined
Segment output	ABS	W	unbl	Cannot be loaded by SEGRES if redefined

[†] Word addressable file organization

GLOSSARY

ABORT

To terminate a program or job when a condition (hardware or software) exists from which the program or computer cannot recover.

ABSOLUTE ADDRESS

1. An address that is permanently assigned by the machine designer to a storage location.

2. A pattern of characters that identifies a unique storage location without further modification. Synonymous with machine address.

ABSOLUTE INFORMATION

Optionally included as a block in an object module, this information must be stored at a specific origin in the field length. Generally, it is used to store information in the job communication area from RA(S) + 77_8. It is not acceptable for segment or overlay loading.

ADDRESS

1. An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network.

2. Any part of an instruction that specifies the location of an operand for the instruction.

ALLOCATE

To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

ALLOCATION UNIT

The smallest amount of storage space (for example, 5 sectors of mass storage) that can be assigned upon request.

ASSEMBLE

To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

ASSIGN

To reserve a part of a computing system for a specific purpose (usually refers to an active part such as an I/O device or a computation module).

ATTACH

Allows a job to gain access to a permanent file. The type of access can be controlled by requiring passwords, if desired, before the file can be read and/or written.

AUDIT

A listing of non-private type permanent file information (obtained from permanent file catalog entries) which can be used for accounting or historical purposes.

BASE ADDRESS

A given address from which an absolute address is derived by combination with a relative address.

BLANK COMMON BLOCK

A common block into which data cannot be stored at load time. The first declaration need not be the largest. In basic loading and segmented loading, but not in overlay loading, the blank common is allocated after all object modules have been processed. For overlay loading, allocation of blank common is more complex.

BLOCK

A group of contiguous characters recorded on and read from magnetic tape as a unit. Blocks are separated by record gaps. A block and a physical record are synonymous.

BUFFER

A storage device used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another. It is normally a block of memory used by the system to transmit data from one place to another. Buffers are usually associated with the I/O system.

CARD IMAGE

A one-to-one representation of the contents of a punched card, for example, a matrix in which a 1 represents a punch and a 0 represents the absence of a punch.

CATALOG (Noun)

A list or table of items with descriptive data, usually arranged so that a specific kind of information can be readily located. (Example: the Permanent File Catalog.)

CHANNEL

A path along which signals can be sent. (Example: data channel, output channel.)

CHARACTER

A logical unit composed of bits. Internally, SCOPE 2 uses 6-bit display code characters. 8-bit characters on 9-track magnetic tapes are converted to and from 6-bit characters.

CODE

1. A system of characters and rules for representing information in a form understandable by a computer.

2. Translation of a problem into a computer language.

COMMON BLOCK

A block that can be declared by more than one object module. More than one module can specify data for a common block, but if a conflict occurs, information is loaded over previously loaded information. A module may declare no common blocks or as many as 509 common blocks. The two types of common blocks are labeled common and blank common.

CORE IMAGE

Also referred to as the loaded program or an absolute program. This is the final image produced by the load operation. For control-statement-initiated load, the core image is the entire job field length from RA(S) + 100_8 through RA(S) + field length -1. In addition, it may include the ECS/LCM field image.

For a user-call-initiated load operation the core image occupies only that portion of the field length specified by the user as being available.

CORE IMAGE MODULE

The core image module is loader tables consisting of the core image. It can be saved on a file for subsequent reloading and execution.

CYCLE

One of up to five separate and distinct files under a permanent file name. Each cycle is identified by the permanent file name and a cycle number.

DATA

1. Information manipulated by or produced by a computer program.

2. Empirical numerical values and numerical constants used in arithmetic calculations. Under SCOPE, data is considered to be that which is transformed by a process to produce the evidence of work. Parameters, device input, and working storage are considered data. In the CPU, the exchange jump package is considered data.

DAYFILE

A short history of a job, that includes a list of control statements, comments, and messages.

DEADSTART

That process by which an inactive machine is brought up to an operational condition ready to process jobs.

DEBUG

To detect, locate, and remove mistakes from a routine or malfunctions from a computer. Synonymous with troubleshoot.

DIAGNOSTIC

1. Pertaining to the detection and isolation of a malfunction or a mistake.

2. A message printed when an assembler or compiler detects a program error.

DISPOSITION CODE

A code used in I/O processing to indicate the disposition to be made of a file when its corresponding job is terminated or the file is closed (for example, print, punch Hollerith, punch binary).

END-OF-INFORMATION DELIMITER

1. A card with a 6/7/8/9 punch in column 1.

2. End-of-information of job file.

END-OF-PARTITION DELIMITER

1. A card with a 7/8/9 punch in column 1 and a level 17_8 Hollerith punch in columns 2 and 3.

2. A W format flag record (zero length, flag bit set).

3. A tapemark or equivalent RCW for a blocked file with record type other than W, X, S, or Z with C blocking.

4. A level 17_8 48-bit appendage on a C blocked file with record type S or Z.

END-OF-SECTION DELIMITER

1. A card with a 7/8/9 punch in column 1 and (optionally) level 0 to 16_8 in columns 2 and 3.

2. A W-format flag record on a file (zero length, flag and delete bit set).

3. A level 0 to 16_8 48-bit appendage in a Z record C blocked file.

ENTRY POINT

A location within a block that can be referenced from object modules that do not declare the block. Each entry point has a unique name associated with it. The loader is given a list of entry points in a loader table. A block can contain any number of entry points.

The loader accepts an entry point name that is 1 to 7 characters; colons are illegal.

Some language processors may produce entry point names under more restricted formats due to their own requirements.

EXCHANGE JUMP

Execution of a CPU program is initiated by an exchange jump. The particular program is defined by the contents of the exchange package area before the exchange jump took place. In order for the program to execute, the proper contents of its operational registers must be loaded into the CPU. These contents are what is contained in the exchange package area associated with the program in question.

EXTERNAL REFERENCE

A reference in one object module to an entry point in a block not declared by that module. Throughout the loading process, externals are matched to entry points (this is also referred to as satisfying externals); that is, addresses referencing externals are supplied with the correct address. In some cases, for SCOPE 3.4, this process is inhibited (for example, OMIT request); the external reference then remains unsatisfied.

FILE

A logically connected set of information. It is the largest collection of information that may be addressed by the name. Each FILE has a logical file name and reference to it must be by name. A file has a beginning called the beginning-of-information, before which no data exists, and an end-of-information, after which no data exists. Tape labels are not considered part of file data.

FILEMARK

Refer to tapemark.

GLOBAL LIBRARY SET

A library set to be used for all subsequent loads in your job until you give the loader further notice.

INPUT FILE

After the job has entered the system and has become a candidate for processing, the second through last sections are separated from the first section and become the INPUT file. This file contains the programs/data referenced by various job steps. The user can manipulate the INPUT file just like any other file (excluding write operations).

JOB

1. An arbitrarily defined parcel of work submitted to a computing system.

2. A collection of tasks submitted to the system and treated by system as an entity. A job is presented to the system as a formatted file. With respect to a job, the system is parametrically controlled by the data content of a job file.

JOB CONTROL FILE

After the job has entered the system and has become a candidate for execution, the job control statement section is made into a separate file called the job control file (also known as control statement file). The user cannot manipulate his job control file.

JOB CONTROL STATEMENT

Any of the statements used to direct the operating system in its functioning, as compared to data, programs, or other information needed to process a job but not intended directly for the operating system itself. A control statement may be expressed in card, card image, or user terminal keyboard entry medium.

JOB DAYFILE

During the execution of the job, a special log or dayfile is maintained. At job termination, the job dayfile is appended to the OUTPUT file of the job. The job dayfile serves as a time ordered record of the activities of the job--all control statements executed by the job, significant information such as file assignment or file disposition, all operator interactions with a job, and errors are recorded in this file.

JOB DECK

The physical representation of a job, before execution, either as a deck of cards or as a file of W-format records. The first section of the job file begins with a job statement and contains the job control statement which will be used to control the job. Following sections contain the programs and data which the job will require for the various job control statements. The job deck is terminated by an end-of-information delimiter.

LABEL (Standard)

An 80-character block at the beginning or end of a magnetic tape volume or file, which serves to identify and/or delimit that volume or file. The record manager supports the following ANSI standard labels:

1. VOL1
2. HDR1
3. EOV1
4. EOF1

LABELED COMMON

A common block into which data can be stored at load time. Depending on the type of source statement (FORTRAN, COMPASS, etc.), a labeled common block may specify CM/SCM or ECS/LCM for storage. Upon encountering the first declaration of a labeled common block, the loader allocates the amount of memory required of the type specified. A later declaration of the same block should not be larger than the initial declaration. If it is, a non-fatal error occurs and the original declaration holds.

LIBRARY FILE

A mass storage file composed of a directory and a set of sequentially organized partitions. Both system and user library files have the same structure and are created in the same way. This file organization manages the directory for the user and allows the user to position to the start of a partition and to subsequently retrieve the records contained in the partition.

LIBRARY SETS

A list of libraries to be searched for entry-point names and for satisfying externals. It can consist of both system and user libraries. NUCLEUS is excluded.

LOAD COMPLETION

Actions taken by the loader after all requests have been performed. The last action normally taken is to start execution of the loaded program. Only a certain type of request can be the last request processed before the load is completed. All other requests, when processed, cause the loader to seek the next request to be processed.

LOADER TABLES

The form in which object code and loader object directives are presented to the loader. Loader tables are generated by compilers and assemblers according to loader requirements. The tables contain information required for loading such as type of code, names, types and lengths of storage blocks, data to be stored, etc.

A sequence of tables is sometimes referred to as an object module.

LOADING

The placement of instructions and data into core so that it is for execution. Loader input is obtained from one or more local files and/or libraries. Upon completion of loading, execution of the program in the job's field length is optionally initiated.

Loading also involves performance of load-related services such as generation of a load map, presetting of unused core to a user-specified value and generation of overlays or segments. A load that does not generate overlays or segments is referred to as a basic load.

LOCAL FILE

A file associated with a particular job on a temporary basis (not a permanent file).

LOCAL LIBRARY SET

A library set to be used for a single load sequence in addition to the global library set.

NUCLEUS LIBRARY

The library that contains most of the operating and product set members as core image modules. It contains no object modules and cannot be searched for externals.

OBJECT MODULE

Often referred to as a relocatable subprogram, this is the basic program unit produced by a compiler or assembler. COMPASS normally produces an object module from source statements delineated by IDENT and END. In FORTRAN, the corresponding beginning statements are PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION. The corresponding end statement is END.

An object module consists of several loader tables that define blocks, their contents, and address relocation information.

ON-LINE TAPE

A magnetic tape unit from which a file is accessed directly without first being copied to mass storage. The concepts of staging tapes and on-line tapes are mutually exclusive, as are the configurations they represent. That is, under SCOPE 2, the stations do not read/write tape directly and the on-line tapes are not staged.

OPERATING SYSTEM

1. The executive, monitor, utility, and any other routines necessary for the performance of a computer system.

2. A resident executive program (an executive routine in internal storage which has a language of its own and automates certain aspects of machine operation).

OUTPUT FILE

This file contains the list output from compilers and assemblers unless the user designates some other file. At job end, the dayfile is added to the OUTPUT file and the file is sent to a station for printing.

OVERLAYING

A technique for bringing routines into high-speed storage from some other form of storage during processing, so that several routines will occupy the same storage locations at different times. Overlaying is used when the total storage requirements for instructions exceed the available main storage.

PARTITION

This is a group of sections which is terminated by a special record or condition. The terminator is different for different types of records.

W type records	Undeleted W flag
S type records and Z record with C blocking	Level 17_8
Other types	Tapemark or recovery control word (indicating tapemark)

PARTIAL STAGING

A technique permitted only for unlabeled tapes. It allows the user job to stage some blocks while bypassing others.

PERMANENT FILE

A file known to the operating system as being permanent (the file will survive deadstarts). Permanent files may be:

1. Created by a job (by cataloging a local file)
2. Attached by a job for its own use
3. Detached by the job (returned to the operating system) when finished

Permanent files may have certain restrictions for their access such as:

1. Access only with a special keyword identifier
2. Read only access

PHYSICAL RECORD

Refer to block.

PROGRAM

1. A sequence of coded instructions that solves a problem.

2. To plan the procedures for solving a problem. This may involve, analyzing the problem, preparing a flow diagram, providing details, developing and testing subroutines, allocating storage, specifying I/O formats, and incorporating a computer run into a complete data processing system.

PROGRAM BLOCK

The block within an object module that usually contains executable code. It is automatically declared for each object module (though it may be zero-length). It is local to the module; that is, it can be accessed from other modules only through use of external symbols. Data placed in a program block always comes from its own object module.

PROGRAM NAME

Also referred to as ident name or deck name, it is the name contained in the loader PRFX table at the beginning of each module. A program name is 1 to 7 characters; colons are illegal.

PURGE

To delete a permanent file from the system. This enables releasing its mass storage space, erasing its catalog entries, etc.

RECORD

A group of contiguous words or characters related to each other by virtue of convention. A record may be fixed or variable length. Record and logical record are synonymous.

REEL

Refer to volume.

REFERENCE ADDRESS

The starting absolute address of the field length assigned to the user's job.

RELOCATE

In programming, to move a routine from one portion of internal storage to another and to adjust the necessary address references so that the routine, in its new location, can be executed.

Instruction addresses are modified relative to a fixed point or origin. If the instruction is modified using an address below the reference point, relocation is negative. If addresses are above the reference point, relocation is positive. Generally, a program is loaded using positive relocation.

REQUEST

To specify the need for a system resource such as an on-line magnetic tape unit.

RESOURCE

An element that can be temporarily assigned upon request.

REWIND

To return a tape or disk to its beginning.

SECTION

A group of records that is terminated by a special record or condition. Generally it is greater than a record but less than a partition. Terminators are:

W type records	Deleted W flag
Z record with C blocking	Level 0 to 16_8
S record	None (except for COPYS which considers each record a section)
Other types	None

SEQUENTIAL FILE

A collection of records that are placed in physical rather than logical order. Given the location of one record, the location of the next can be determined by the physical position of the previous record. A tape file, punch card file, printer file, etc., are all classified as sequential.

SOFTWARE

The collection of programs and routines associated with a computer system; for example, compilers, library routines.

SPOOLING

A system-controlled process by which I/O to and from a unit record device is handled on a lower priority basis for overall system efficiency.

STAGED TAPE

A file from which a volume is copied from magnetic tape unit at a station to the system mass storage or vice versa. The user accesses the file from the system mass storage copy when needed.

SYSTEM LIBRARY

That file or group of files containing the core image system overlays and the system relocatable code available to all users on a read-only basis. The system library may include code generated and inserted by the user.

SYSTEM TABLES

Tables used by the operating system and which lie outside of the user's field length.

TABLE

A collection of data, each item being uniquely identified either by some label or by its relative position.

TAPEMARK

A special hardware bit configuration recorded on magnetic tape. It indicates the boundary between files and labels. It is sometimes called a file mark.

TIME SLICE

The maximum amount of time during which the CPU can be executing a job without a re-evaluation at to which job should have the CPU next.

UNIT RECORD DEVICE

A device such as a card reader, line printer or card punch.

UNLOAD

To remove a tape from ready status by rewinding beyond the load point. The tape is then no longer under control of the computer.

UNSATISFIED EXTERNAL

An external reference for which the loader has not yet loaded a module containing the matching entry point.

VOLUME

A physical unit of storage media. The term volume is synonymous with reel of magnetic tape.

WORD

A group of bits (or 6-bit characters) between boundaries imposed by the computer. Word size must be considered in the implementation of logical divisions such as characters. In this publication, a word is assumed to be 60 bits.

WORD ADDRESSABLE FILE

A mass storage file that may be considered by the user as continuously non-blocked data. Data is written to/retrieved from the file by specification of a relative word address on the file. The data on the file may be unstructured (U record format) or structured (any other record format except S).

INDEX

- C parameter on LABEL 11-3, 6
- Cards, punched 9-1, 18

COMMENT SHEET

MANUAL TITLE CONTROL DATA® CYBER 70/MODEL 76

COMPUTER SYSTEM - SCOPE 2 USER'S GUIDE

PUBLICATION NO. 60372600 REVISION A

FROM: NAME: ____________________

BUSINESS ADDRESS: ____________________

COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

PRINTED IN U.S.A

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

CUT ALONG LINE

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Technical Publications Department
4201 North Lexington Ave.
Arden Hills, Minnesota 55112

FOLD

FOLD

INSTALLATION DEFINED PARAMETERS

Use the following table to record values of parameters defined at your site. The table lists only those installation-defined defaults that affect control statement processing. Other default values are either not alterable, affect the internal performance of the system, or are not directly related to control statements.

Parameter	Default	Range
Job time limit (Tn)	________$_8$ seconds	0 to 77777_8 (infinite)
Job priority (Pn)	________$_8$	0 to ________$_8$
SCM field assignment (CMn)	Automatic	______$_8$ to ______$_8$ words
LCM field assignment (ECn)	Automatic	0 to ________$_8$ thousand words
7-track magnetic tape units (MTn)	0 units	0 to ________$_8$ units
9-track magnetic tape units (NTn)	0 units	0 to ________$_8$ units
9-track conversion code (US/EB)	________	ASCII or EBCDIC
Tape data density (HI,HY,LO,PE)	________	200, 556, or 800 bpi
Tape label density	________	200, 556, or 800 bpi or same as data
Retry count for parity errors	________	
Staging direction	________	Pre or post
Mass storage allocation units (An)	________ MAU	1 to 16 MAU (A0 to A4)
Mass storage limit for job	________ characters	________________ characters
Loader map	________	None, S, B, E, or X
Loader abort conditions	________	All/fatal/none
Loader preset value	________	None, zeros, ones, indef, inf, ngindef, nginf, alt. zeros, alt. ones.
Load file positioning	________	Rewind or no rewind
Coded punched card format	________	Hollerith (026) or ASCII (029)
Station identifier (ggg)	________	
Terminal identifier (ttt)	________	

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.