60372600

CONTROL DATA CORPORATION

SCOPE 2.1
USER'S GUIDE

CDC® COMPUTER SYSTEMS:
CYBER 70 MODEL 76
7600

REVISION RECORD

REVISION	DESCRIPTION
A	Original printing.
(11-30-72)	
B	Correct technical and typographic errors, reflect changes to system through PSR Summary 65,
(9-73)	and add appendix G.
C	Add SCOPE 2.1 features through PSR Summary 175. This edition obsoletes all previous editions.
(6-74)	
D	Add SCOPE 2.1 features through PSR Summary 201 (SCOPE 2.1.3). This edition obsoletes all
(12-75)	previous editions.
E	Add description of CDC CYBER control language (CCL) and make miscellaneous corrections
(3-78)	through PSR Summary 258 (SCOPE 2.1.5).

Publication No.
60372600

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

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

© 1972, 1973, 1974, 1975, 1978
by Control Data Corporation
Printed in the United States of America

LIST OF EFFECTIVE PAGES

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

PAGE	REV
Front Cover	-
Title Page	-
ii	E
iii	E
iv	E
v	E
vi	E
vii/viii	E
ix	E
x	E
xi	E
xii	E
xiii	E
xiv	E
xv	E
xvi	E
xvii	E
1-1	E
1-2	E
1-3	E
1-4	E
1-5	E
1-6	D
1-7	E
1-8	E
1-9	E
1-10	D
1-11	E
1-12	E
1-13	E
1-14	E
1-15	D
1-16	E
1-17	E
1-18	D
1-19	D
1-20	D
1-21	D
1-22	D
1-23	D
2-1	C
2-2	E
2-3	E
2-4	E
2-5	E
2-6	E
2-7	D
2-8	C
2-9	D
2-10	D
2-11	D
2-12	D
2-13	D
2-14	D
3-1	E
3-2	E
3-3	D
3-4	E

PAGE	REV
3-5	E
3-6	D
3-7	D
3-8	E
3-9	D
3-10	D
3-11	C
3-12	C
3-13	D
3-14	E
3-15	E
3-16	D
3-17	E
3-18	E
3-19	E
3-20	E
3-21	E
3-22	E
3-23	D
3-24	D
4-1	E
4-2	E
4-3	E
4-4	E
4-5	D
4-6	D
4-7	D
4-8	D
4-9	D
4-10	D
4-11	E
4-12	E
4-13	D
4-14	D
4-15	E
4-16	D
4-17	E
4-18	D
5-1	C
5-2	E
5-3	D
5-4	D
5-5	E
5-6	D
5-7	C
5-8	E
5-9	D
5-10	E
5-11	C
5-12	E
5-13	D
5-14	E
5-15	D
5-16	E
5-17	E
5-18	E
5-19	E
5-20	D
5-21	D

PAGE	REV
5-22	C
5-23	C
5-24	D
5-25	D
5-26	D
5-27	D
5-28	C
5-29	E
5-30	D
6-1	C
6-2	C
6-3	E
6-4	E
6-5	E
6-6	E
6-7	E
6-8	E
6-9	E
6-10	E
6-11	E
6-12	E
6-13	E
6-14	E
6-15	C
6-16	D
6-17	D
6-18	C
6-19	C
6-20	E
6-21	D
6-22	E
7-1	D
7-2	D
7-3	E
7-4	E
7-5	E
7-6	D
7-7	E
7-8	D
7-9	D
7-10	D
7-11	D
7-12	E
7-13	D
7-14	E
7-15	D
7-16	D
7-17	E
7-18	E
7-19	C
8-1	E
8-2	E
8-3	E
8-4	E
8-5	E
8-6	D
8-7	E
8-8	D
8-9	E

PAGE	REV
8-10	E
8-11	D
8-12	D
8-13	D
8-14	E
8-15	D
8-16	D
8-17	D
8-18	E
8-19	E
8-20	E
8-21	E
8-22	E
9-1	E
9-2	E
9-3	E
9-4	C
9-5	D
9-6	D
9-7	D
9-8	D
9-9	D
9-10	E
9-11	D
9-12	C
9-13	C
9-14	D
9-15	E
9-16	E
9-17	D
9-18	D
9-19	D
9-20	D
9-21	D
9-22	E
10-1	E
10-2	E
10-3	E
10-4	E
10-5	E
10-6	E
10-7	E
10-8	E
10-9	E
10-10	C
10-11	E
10-12	C
10-13	E
10-14	E
10-15	D
10-16	E
11-1	C
11-2	E
11-3	D
11-4	D
11-5	D
11-6	D
11-7	E
11-8	C

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

PAGE	REV
C-3	C
C-4	C
C-5	D
C-6	C
C-7	C
D-1	E
D-2	D
D-3	C
D-4	E
D-5	E
D-6	E
D-7	E
D-8	E
D-9	E
D-10	E
D-11	E
D-12	E
D-13	E
D-14	E
E-1	E
E-2	D
E-3	E
E-4	D
E-5	E
E-6	E
E-7	E
E-8	E
E-9	E
E-10	E
E-11	E
E-12	E
E-13	E
E-14	E
E-15	E
E-16	E
F-1	E
F-2	E
F-3	C
G-1	C
G-2	E
G-3	C
G-4	D
G-5	D
G-6	E
G-7	E
G-8	D
G-9	D
G-10	D
G-11	D
G-12	D
G-13	D
G-14	D
G-15	D
G-16	D
G-17	D
G-18	D
G-19	D
G-20	D
G-21	E
G-22	D
G-23	C
G-24	C
G-25	E
G-26	D
G-27	D
G-28	D
G-29	D
G-30	D
G-31	D
G-32	C
G-33	C

PAGE	REV
G-34	D
G-35	C
G-36	D
G-37	D
G-38	D
G-39	D
G-40	D
H-1	E
H-2	E
H-3	E
H-4	E
H-5	E
H-6	E
H-7	E
H-8	E
H-9	E
H-10	E
H-11	E
Index-1	E
Index-2	E
Index-3	E
Index-4	E
Index-5	E
Index-6	D
Index-7	E
Index-8	E
Index-9	E
Index-10	E
Index-11	E
Comment Sheet	E
Back Cover	-

PAGE	REV

PAGE	REV

PAGE	REV

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 NOS/BE or SCOPE 3.4

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 led 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. Section 13 describes the CONTROL DATA® CYBER control language (CCL).

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.

NOTE

All references in this manual to the SCOPE 3.4 Operating System also apply to the NOS/BE 1 Operating System.

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 Publication	Publication Number
CYBER 70/Model 76 Computer System Reference Manual	60367200
SCOPE 2 Reference Manual	60342600
SCOPE 2 Instant Manual	60344300
SCOPE 2 Record Manager Reference Manual	60454690
SCOPE 2 Loader Reference Manual	60454780
SCOPE 2 Diagnostic Handbook	60344100
SCOPE 2 Installation Handbook	60426100
COMPASS Reference Manual	60360900
COMPASS Instant Manual	60361000
COMPASS Instruction Card	60361700
NOS/BE Enhanced Station Operator's/Reference Manual	60494200

Control Data Publication	Publication Number
UPDATE Reference Manual	60449900
UPDATE Instant Manual	60450000
FORTRAN RUN Reference Manual	60305600
FORTRAN Extended Reference Manual	60497800
FORTRAN Extended Instant Manual	60497900
ALGOL Reference Manual	60496600
SYMPL Reference Manual	60496400
APEX II Reference Manual	59158100
APEX III Reference Manual	76070000
APT Reference Manual	17313600
SIMSCRIPT Reference Manual	97400200
COBOL Reference Manual	60496800
SORT/MERGE Reference Manual	60497500
SORT/MERGE Instant Manual	60497600
Programming Reference Aids	60158600

NOTE

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

SECTION 1	GENERAL DESCRIPTION	1-1
	Introduction to SCOPE 2	1-1
	Operating Environment	1-2
	Hardware Configuration	1-2
	Multiple Mainframes	1-5
	Software Configuration	1-5
	CDC CYBER Station Operating System	1-11
	Job Flow	1-13
	Job Initiation	1-14
	Job Processing	1-16
	Job Termination	1-17
	The Job Dayfile	1-17
	Introduction to Logical Files	1-19
	Naming Files	1-20
	FORTRAN Object-Time File Names	1-20
	COBOL Object-Time File Names	1-23
SECTION 2	USING SCOPE CONTROL STATEMENTS	2-1
	The Job Name	2-1
	Optional Job Identification Statement Parameters	2-3
	CDC CYBER Station Processor Code	2-4
	Execution Time Limit	2-4
	Job Priority	2-5
	Control Statements	2-6
	Directives	2-8
	Separator Cards	2-8
	End-Of-Section Card	2-8
	End-Of-Partition Card	2-9
	End-Of-Information Card	2-9
	Examples	2-10
	Control Statement Section	2-10
	Compile Source Language Program	2-11
	Compile and Execute	2-12
	Two Compilations With Combined Execution of the Object Programs	2-13
	Complex Data Structure	2-14
SECTION 3	JOB PROCESSING	3-1
	Compiling or Assembling Programs	3-1
	Loading and Executing Programs	3-4
	Combined Load and Execute Request	3-4
	Loading of Object Modules	3-10

	Program Image Modules and How They are Loaded	3-10
	Loading and Execution as Separate Operations	3-11
	Using NOGO to Generate Program Image Modules	3-14
	Load Sequences	3-14
	Selectively Load Modules From Files	3-16
	Setting Load Sequence Characteristics	3-17
	Using Libraries	3-18
	Definition of Library	3-18
	Library Sets	3-19
	Loading Directly From Libraries	3-22
	Loading Partitions From Libraries	3-23
SECTION 4	PROGRAM AND JOB OPTIONS	4-1
	Using Memory	4-1
	User SCM	4-1
	User LCM	4-1
	Job Supervisor LCM	4-1
	I/O Buffers in LCM	4-1
	Maximum Available LCM	4-2
	Automatic Memory Management	4-3
	User-Controlled Memory Mode	4-5
	Returning to Automatic Mode	4-8
	Presetting Memory	4-10
	Inserting Comments in the Program Listing	4-11
	Pause for Operator Action	4-12
	Setting Program Switches	4-13
	Processing Interdependent Jobs	4-14
	Job Dependency Parameter	4-14
	TRANSF Control Statement	4-15
	Job Rerun Limit	4-15
	Rewinding of Load Files	4-17
SECTION 5	FILE STRUCTURES	5-1
	File Information Table	5-1
	Introduction to FILE Statement	5-1
	Multiple FILE Statements	5-2
	Specifying Record Type	5-2
	Specifying the Maximum Record Length	5-12
	Unblocked File Format	5-12
	Rules for Accessing Unblocked Files	5-13
	How to Specify Unblocked	5-14
	Blocked File Format	5-14
	The Block	5-14
	Accessing Blocked Files	5-16
	How to Specify Blocking	5-16
	Partitions	5-19
	Sections	5-21
	Access Methods	5-26
	File Processing Direction	5-27
	Program Exit Conditions	5-27
	End-of-Data Exit	5-28
	Error Exit	5-28

SECTION 6 MAGNETIC TAPE FILES 6-1

Staging Tapes 6-3
Job Statement Parameters for Tape Staging 6-3
Prestaging 6-4
Poststaging 6-4
Specifying Type of Tape Unit and Density 6-5
Identifying the Station for Staging 6-6
Character Conversion and Parity 6-7
Staging All or Part of Files 6-9
Using On-Line Tapes 6-14
Scheduling On-Line Tape Units 6-15
Requesting On-Line Tape Units 6-15
Character Conversion and Parity 6-16
Positioning On-Line Magnetic Tape Files 6-17
Using Volume Serial Numbers With On-Line Tapes 6-18
Multifile On-Line Tapes 6-18
Mount Option 6-19
Suppressing Read-Ahead/Write-Behind 6-19
Using On-Line Tape Units for Staging 6-20
Unloading/Returning On-Line Tape Units 6-21
Magnetic Tape Recovery Procedures 6-21
Standard Recovery Procedures 6-21

SECTION 7 MASS STORAGE FILES 7-1

Introduction to REQUEST Statement 7-1
Mass Storage Sets 7-2
System Set 7-3
Removable Sets 7-3
How Mass Storage Files Originate 7-3
Using the System Set 7-5
Assignment by Device Type 7-5
Assignment by SETNAME 7-5
Assignment by VSN 7-5
Using Removable Sets 7-5
Device Scheduling - 844-2 Disk Storage Subsystem 7-6
Creation of Removable Sets 7-6
Mounting Set Members 7-9
Requesting Use of Removable Sets 7-10
Dismounting Set Members 7-11
Deleting Set Members 7-11
Minimum Allocation Units (MAU) 7-12
Transfer Unit Size 7-14
Write Check Option 7-16
Returning Mass Storage Files 7-17
Job Mass Storage Limit 7-19

SECTION 8 PERMANENT FILES 8-1

Using SCOPE 2 Permanent Files 8-1
Cycles 8-1
Cycle Numbers 8-1
Logical and Permanent File Names 8-2
Creator Identification (ID) 8-2
Creating the Initial Cycle of a File 8-2
Accessing the Initial Cycle of a File 8-4

	Catalog with Passwords	8-8
	Cataloging Subsequent Cycles	8-14
	Cataloging a File on a Removable Set	8-15
	Attaching a File on a Removable Set	8-16
	Altering the Size of Permanent Files	8-16
	Purging Permanent Files	8-18
	Installation Defined Privacy Procedures	8-20
	Using Permanent Files at Other Mainframes	8-20
	Using the Station Parameter	8-20
SECTION 9	UNIT RECORD DEVICES AT STATIONS	9-1
	Card Reader Input	9-1
	Coded Punched Card Input	9-3
	SCOPE Binary Card Input	9-7
	Free-Form Binary Card Input	9-8
	End-of-Section Level Numbers	9-10
	Printer (List) Output	9-10
	Identifying Printer Output	9-11
	Printer Carriage Control	9-11
	Disposing Print Files to Stations	9-15
	Punched Card Output	9-19
	Identifying Punched Output	9-19
	Separator Cards	9-19
	Mispunched Cards	9-19
	Coded Punched Cards	9-20
	SCOPE Binary Punched Cards	9-20
	Free-Form Binary Punched Cards	9-21
	Routing Punched Files to Stations	9-21
SECTION 10	COPYING AND POSITIONING FILES	10-1
	Introduction to Copy Routines	10-1
	How to Copy Files	10-1
	Selecting the Copy	10-1
	File Descriptions for Copies	10-5
	Specifying Buffer Size	10-6
	Positioning Sequential Files	10-7
	Positioning Files Forward	10-7
	Positioning Files Backward	10-10
	Writing File Delimiters	10-12
	Comparing Files	10-12
	Comparing Sections	10-13
	Comparing Partitions	10-13
	Comparing S Records	10-15
	Error Record Count	10-15
	List Control Parameters	10-15
	Abort Parameter	10-16
SECTION 11	FILE LABELS	11-1
	Introduction	11-1
	Labels on Magnetic Tape	11-1
	Users	11-1
	Standard Labels	11-1
	Requesting Standard Labeled Tapes	11-1
	Providing Standard Label Information	11-2
	Protection of Unexpired Labeled Tapes	11-6
	Copying Labeled Tapes	11-6
	Label Density	11-8
	Label Parity and Character Conversion	11-8

SECTION 12	ANALYTICAL AIDS	12-1
	Controlling Your Job With EXIT Statements	12-1
	Setting Error Conditions	12-5
	Setting Loader Error Options	12-8
	Obtaining Program Dumps	12-9
	Requesting a Standard Dump	12-10
	Requesting SCM Dumps	12-11
	Requesting LCM Dumps	12-12
	Obtaining Load Maps	12-13
	MAP Statement	12-14
	LDSET Option	12-16
	Obtaining File Dumps	12-16
	Requesting a Dump of Entire File	12-16
	Specifying a List File	12-19
	Specifying Dump Limits	12-19
	Obtaining Dayfile Summaries	12-20
SECTION 13	CDC CYBER CONTROL LANGUAGE (CCL)	13-1
	Syntax	13-1
	Expressions	13-1
	Operators	13-1
	Operands	13-3
	CCL Statement Overview	13-4
	SET and DISPLAY Statements	13-5
	SET	13-5
	DISPLAY	13-6
	Functions	13-8
	FILE	13-8
	Conditional Statements (IFE, SKIP, ELSE, ENDIF)	13-10
	IFE	13-10
	SKIP	13-12
	ELSE	13-12
	ENDIF	13-14
	Iterative Statements (WHILE, ENDW)	13-15
	Procedures	13-16
	Procedure Header Statement (. PROC)	13-16
	Procedure Body	13-17
	Procedure Call and Return	13-18
	Keyword Substitution	13-25
	Procedure Commands	13-28

APPENDIXES

APPENDIX A	CDC CYBER STANDARD CHARACTER SET	A-1
APPENDIX B	JOB COMMUNICATION AREA	B-1
APPENDIX C	STANDARD LABELS	C-1
APPENDIX D	SUMMARY OF FILE FORMATS	D-1
APPENDIX E	USING RECORD MANAGER FOR FILE FORMAT CONVERSION	E-1
APPENDIX F	DEFAULT FILE DESCRIPTIONS	F-1
APPENDIX G	ANALYZING ERRORS IN A SAMPLE FORTRAN PROGRAM	G-1
APPENDIX H	GLOSSARY	H-1

INDEX

FIGURES

1-1	System Configuration (CDC CYBER 70/Model 76 or 7600)	1-3
1-2	CDC CYBER Station Configuration	1-12
1-3	7611-1 I/O Station Configuration	1-13
1-4	Job Flow Through System	1-14
1-5	Job Deck Translation	1-15
1-6	Switch, Swap, and Rollout	1-16
1-7	Sample Dayfile Listing	1-18
2-1	End-Of-Section Card (EOS)	2-8
2-2	End-Of-Partition Card (EOP)	2-9
2-3	End-Of-Information Card (EOI)	2-10
3-1	Structure of Loaded Program	3-11
3-2	Processing of Control Statements	3-15
5-1	Unblocked File Format	5-13
5-2	Blocking Types	5-15
5-3	Blocked File Format on Mass Storage	5-16
5-4	End-Of-Partition on Blocked Magnetic Tape	5-20
5-5	File Hierarchy	5-22
5-6	Relationship of S and Z Records (C Blocking)	5-23
6-1	Unlabeled Magnetic Tape Files	6-1
6-2	Labeled Magnetic Tape Files	6-2
7-1	Relationship of MAUs and Transfer Unit Size	7-15
9-1	Hollerith (026) Coded Card	9-4
9-2	ASCII (029) Coded Card	9-4
9-3	Coded Card Images as W Records on INPUT	9-6
9-4	SCOPE Binary Card	9-7
9-5	Flag Cards to Delimit Free-Form Binary Deck	9-9
9-6	Free-Form Binary Card Translation	9-9
9-7	Printer Banner Page (Two Styles)	9-12
9-8	Lace Card	9-19
10-1	Selecting the Proper Copy Statement	10-2
12-1	Flow Chart of EXIT Processing	12-2
12-2	Standard Dump	12-10
12-3	SCOPE 2 Load Map	12-15
12-4	Sample of DMPFILE Output	12-18
13-1	Calling a Procedure from a Job	13-18
13-2	Calling a Procedure from Another Procedure	13-19

EXAMPLES

1-1	Correlating File Names With FORTRAN I/O Statements	1-21
1-2	Equating File Declarations on FORTRAN PROGRAM Statements	1-22
1-3	COBOL File Name Assignments	1-23
2-1	Sample Job	2-2
2-2	Job Containing Control Statements Only	2-10
2-3	Job With Source Language Program	2-11
2-4	Job With Source Language Program and Data	2-12
2-5	Job With Two Compiler Language Programs	2-13
2-6	Job With Complex Data Structure	2-14
3-1	Request for FORTRAN Extended Compilation	3-1
3-2	Request for COBOL Compilation With Options Specified	3-2

3-3	LGO File Name Statement	3-4
3-4	Renaming the Load-and-Go File	3-5
3-5	No Substitution of File Names on LGO	3-5
3-6	Substitution of File Names on LGO	3-6
3-7	Precedence of Equating File Names	3-6
3-8	Loading From INPUT	3-7
3-9	Illegal Verb/Name Call Statement	3-9
3-10	Load From LGO and INPUT; Then Execute	3-13
3-11	Selective Load From a File by Program Name	3-17
3-12	Using LDSET Statements in Load Sequence	3-18
3-13	Defining Global Library	3-20
3-14	Combining New and Old Library Sets	3-21
3-15	Defining a Local Library	3-22
3-16	Direct Load From Library Using LIBLOAD	3-23
3-17	Load Partition From Library Using LOAD	3-24
3-18	Load Partition From Library Using SLOAD	3-24
4-1	Job Using Automatic Memory Management	4-4
4-2	Using the CM Parameter to Control SCM	4-6
4-3	Using the RFL Statement to Control SCM	4-7
4-4	Mixed Mode Control of SCM	4-9
4-5	Mixed Mode Control of Both SCM and LCM	4-10
4-6	Comments in Dayfile Listing	4-11
4-7	Comment Two Lines Long	4-12
4-8	Directing the Operator Through a PAUSE Statement	4-12
4-9	Using the SWITCH Statement	4-13
4-10	COBOL Test of Sense Switches	4-14
4-11	Job Dependency String	4-16
4-12	Rewind or No Rewind of Load Files	4-18
5-1	Placement of FILE Statement	5-2
5-2	Overriding Default of W Record Type for FORTRAN Program	5-10
5-3	COBOL Assignment of Record Types Through File Description	5-11
5-4	Using a FILE Statement to Specify Blocking	5-18
5-5	FORTRAN Treatment of EOS on Input File	5-24
5-6	COBOL Treatment of EOS on Input File	5-25
5-7	Specifying Error Option as Accept With No Display	5-29
6-1	Prestaging an Unlabeled Tape	6-4
6-2	Poststaging an Unlabeled Tape	6-5
6-3	Identifying the Station for Staging	6-7
6-4	7-Track Code Conversion for Poststaged Tape	6-8
6-5	9-Track Code Conversion for Poststaged Tape	6-9
6-6	Prestaging Using Volume Serial Numbers	6-11
6-7	Partial Staging by Blocks	6-12
6-8	Partial Staging by Tapemark	6-13
6-9	Staging Entire File	6-14
6-10	Scheduling and Requesting On-Line Tape Units	6-16
6-11	9-Track Code Conversion for On-Line Input Tape	6-17
6-12	On-Line Staged Tape	6-20
6-13	No Recovery and Accept Data Options	6-22
7-1	REQUEST Statement Placement	7-2
7-2	Scheduling of Removable Device	7-6
7-3	Identify Master Device	7-7
7-4	Changing Default Setname for Job	7-7
7-5	Adding a Member to a Set	7-9
7-6	Mounting the Master Device for a Removable Set	7-10
7-7	Deleting Set Members	7-12
7-8	Using the REQUEST Statement Allocation Parameter	7-13

7-9	Using the REQUEST Statement Transfer Unit Parameter	7-16
7-10	Using the REQUEST Write Check Parameter	7-17
7-11	Returning Mass Storage Files	7-18
8-1	Permanent File With No Password Requirements	8-6
8-2	Using the Retention Parameter	8-7
8-3	Defining the Read Password	8-9
8-4	Modifying a Permanent File	8-10
8-5	Using the Turnkey Password	8-11
8-6	Cataloging a New Cycle	8-14
8-7	Cataloging File on Removable Set	8-15
8-8	Attaching a Permanent File From a Removable Set	8-16
8-9	Using the ALTER Control Statement	8-17
8-10	Using the EXTEND Control Statement	8-17
8-11	Purging a Cycle of a Permanent File	8-18
8-12	Purging a Cycle and Adding a New Cycle	8-19
8-13	Moving a Cycle From One Permanent File to Another	8-19
8-14	Attaching a CDC CYBER Station Permanent File	8-21
8-15	Cataloging a Permanent File Attached From a Linked 7600	8-21
8-16	Maintaining UPDATE OLDPL at a Linked NOS/BE Mainframe	8-22
9-1	Reading Cards From INPUT	9-2
9-2	Rewinding INPUT	9-3
9-3	ASCII (029) Coded Punch Input	9-5
9-4	FORTRAN Binary Input (CDC CYBER Station)	9-8
9-5	Copy INPUT to OUTPUT Shifting Each Record	9-14
9-6	Placement of DISPOSE Statement	9-16
9-7	Disposing Print File Created by FORTRAN Program	9-17
9-8	Generate Printer Character Sets	9-17
9-9	Punching Binary Output From Compiler	9-21
9-10	Punching Free-Form Binary	9-22
10-1	File-to-File Copy	10-2
10-2	Copying Records	10-3
10-3	Copying Sections	10-4
10-4	Copying Partitions	10-5
10-5	Copying a Tape as Record Type U, Block Type K	10-6
10-6	Setting MRL	10-6
10-7	Skipping Records Forward	10-7
10-8	Skipping Sections Forward	10-8
10-9	Skipping Partitions Forward	10-9
10-10	Skipping to End-Of-Information	10-9
10-11	Skipping Sections Backward on INPUT	10-11
10-12	Copy and Compare Files	10-14
10-13	Logical Compare of F Records and W Records	10-14
10-14	Literal Copy and Compare of Tapes Described as U Records	10-16
11-1	Using LABEL Statement for Label Generation With COBOL Program	11-5
11-2	Using LABEL Statement for Label Checking With COBOL Program	11-7
12-1	Selective Exit Processing	12-4
12-2	Combination of Exit Paths	12-5
12-3	Using the MODE Statement	12-7
12-4	Requesting No Program Execution for Any Loader Error	12-9
12-5	Request for Standard Dump	12-11
12-6	SCM Dump Taken Within an Exit Path	12-12
12-7	User Control of Load Map	12-17
12-8	Requesting a File Dump	12-19
12-9	Intermediate Accounting Information	12-20
13-1	SET Statement	13-6

13-2	DISPLAY Statement Register 1	13-6
13-3	DISPLAY Statement Register 2	13-7
13-4	DISPLAY Statement	13-7
13-5	DISPLAY Statement True or False	13-7
13-6	FILE Function Using Symbolic Name MS	13-9
13-7	File Function Using TP and LB	13-9
13-8	IFE Statement	13-10
13-9	IFE Expression is False	13-11
13-10	Conditional IFE Statement Within Another IFE Statement	13-11
13-11	SKIP Statement	13-12
13-12	Use of ELSE When Expression in IFE Statement is False	13-13
13-13	Use of ELSE When Expression in IFE Statement is True	13-13
13-14	ELSE Does Not End a SKIP or Another ELSE	13-14
13-15	Skipping Control to the ENDIF Statement	13-14
13-16	WHILE Expression is True	13-15
13-17	Expression in WHILE Statement is False	13-16
13-18	Calling Procedure A from a Library	13-20
13-19	Implicit REVERT Sequence by CCL	13-21
13-20	Values Passed from Procedure File to GLOBAL to Control Statement Section	13-22
13-21	User's REVERT Sequence	13-23
13-22	Keyword Substitution	13-25
13-23	Equivalence Mode	13-26
13-24	Inhibit Substitution	13-26
13-25	Substitution Without Delimiters	13-27
13-26	Value of R1 Substituted for Formal Keyword C4	13-28
13-27	COPYSBF Statement Copies Names from Temporary File to OUTPUT File	13-29
13-28	The Default Temporary File is Assigned a Name	13-30
13-29	FORTRAN Program is in the Procedure File Named VAST	13-30
13-30	Call-by-Name Statement to Call Procedure from a Library	13-31
13-31	All Procedure Commands	13-22
13-32	Use of → in Rewinding a Data File	13-33

TABLES

1-1	System File Names	1-20
3-1	Requests for Compilation or Assembly	3-1
3-2	Options Available During Compilation/Assembly	3-3
3-3	System Verb Table	3-8
4-1	Preset Options	4-11
5-1	FORTRAN Record Type Constraints	5-8
5-2	COBOL Specified Record Types	5-9
5-3	COBOL Specification of Blocking	5-17
5-4	Maximum Block Sizes Allowed for Staged and On-Line Tapes	5-18
5-5	COBOL Determined File Organization	5-26
6-1	Magnetic Tape Density Parameters	6-6
7-1	Allocation of Mass Storage	7-13
9-1	Carriage Control Characters	9-13
12-1	EXIT Statement Processing	12-3
12-2	MODE Statement Parameters	12-6
12-3	MAP Options	12-16
13-1	Commonly Used Symbolic Names	13-4

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 CDC® CYBER 70/Model 76, CDC CYBER 170/Model 176, and the 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 can print 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
- The capability to link several independent computer systems in a multiple mainframe environment
- Permanent file management to protect information from access by unauthorized persons. Permanent files may be maintained at linked main frames as well as at the SCOPE 2 system.
- A chronological history called a dayfile for each job run showing the results of each control statement processed and any problems encountered. This history is printed automatically when the job 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
- Checkpoint restart capability

OPERATING ENVIRONMENT

SCOPE 2 resides in a CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, 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 CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, and 7600 Computing Systems are large-scale, solid-state, general-purpose, digital computing systems. The advanced design techniques incorporated in this system provide for extremely fast and efficient solutions for large-scale, general-purpose processing.

A basic system (refer to Figure 1-1, which depicts one for a CDC CYBER 70/Model 76 or 7600) 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 equipments 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 central memory, large central memory, and an input/output multiplexer. The sections are all contained in the main frame cabinet and operate in a tightly synchronous mode under control of the master clock. Communication outside the main frame cabinet is asynchronous; that is, independent of the master clock.

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.

CENTRAL MEMORY

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

- The instruction stack contains 12 60-bit words for issuing instructions. The stack holds the latest 10 instruction words and the two-instruction word look-ahead. Program loops can be held in the stack thereby avoiding memory references.
- The small central memory (SCM) contains 32,768 or 65,536 60-bit words of semiconductor or core memory, or 131,072 60-bit words of semiconductor memory. Each word holds 10 6-bit characters.

- The large central memory (LCM) contains 262,144 or 524,288 60-bit words. However, the large central memory extension (LCME), or computer systems that contain LCME, can consist of up to 2,097,152 60-bit words. Instructions cannot be executed directly from LCM/LCME.

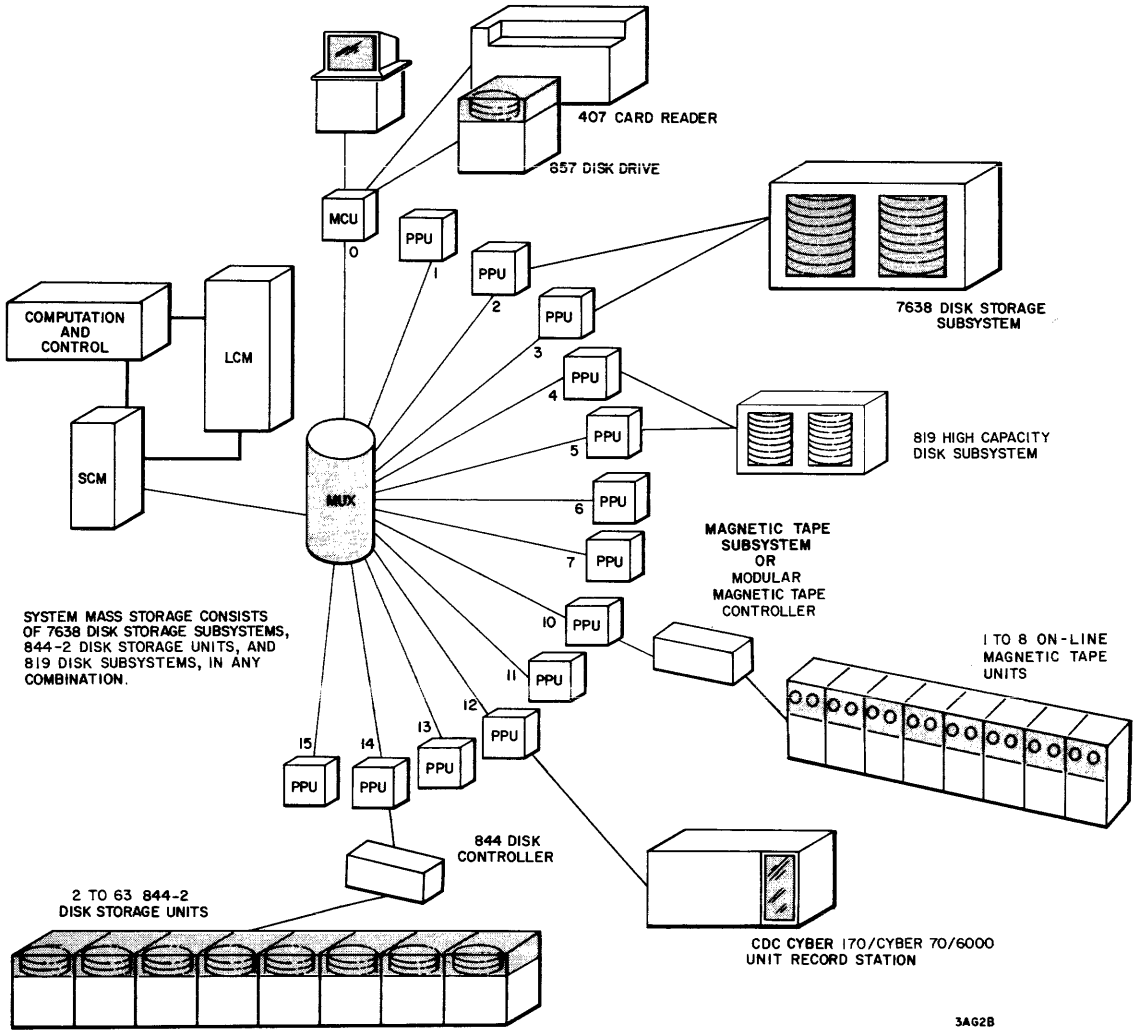


Figure 1-1. System Configuration (CDC CYBER 70/Model 76 or 7600)

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 several thousand 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 CPU programs and associated data. 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. For a CDC CYBER 170/Model 176, the MCU refers to the peripheral processor subsystem and its associated peripheral equipment on which the system maintenance monitor is running.

MASS STORAGE

Mass storage consists of 7638 Disk Storage Subsystems, 819 High Capacity Disk Subsystems, and/or 844-2 Disk Storage Units.

Devices are grouped into sets of which there are two types, system and removable. Each device in a set is a set member.

Mass storage devices of either type can be in the system set. The devices forming the system set are identified at deadstart or recovery and are not logically removable.

Devices that can be logically added to or removed from the configuration belong to removable sets. A user can control the availability of removable set members through control statements. Any of the devices listed above can be used for removable set members.

The user can also control the assignment of files to set members, removable or system, through control statements. Files are assigned to the system set by default.

STATIONS

Stations submit job files and data files to the SCOPE 2 Operating System and receive user-created output data files. Each station operates under control of its own operating system and is responsible for supporting unit-record, magnetic tape, or communication equipment. A station may be physically or logically specified by a three-character identifier used in conjunction with the ST parameter on control statements. A logical identifier refers to a capability of the station rather than to the station itself. A system analyst should be consulted for information on station identifiers used at your site. Refer to Multiple Mainframes, in this section, for more information.

ON-LINE MAGNETIC TAPE UNITS

A configuration optionally includes one to eight 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 or from the on-line unit without the intermediate transfer to mass storage that takes place for staged magnetic tapes from stations.

On-line tape units can be 657-X or 659-X magnetic tape units driven by the Modular Magnetic Tape Controller (MMTC) or can be 667-X or 669-X units driven by the Magnetic Tape System (MTS).

MULTIPLE MAINFRAMES

The CDC CYBER 70/Model 76 and CDC CYBER 170/Model 176 can be linked to other CDC CYBER Series Computer Systems in a multi-mainframe environment. A mainframe can be a CDC CYBER 170/Model 172, 173, 174, or 175, CDC CYBER 70/Model 72, 73, or 74, or 6000 Series Computer System station or can be another CDC CYBER 70/Model 76 or a CDC CYBER 170/Model 176. Each mainframe can process its own jobs; that is, act as a host mainframe, or jobs may be sent from one mainframe to another. Through the use of logical and physical identifiers, the user selects the mainframe on which the job is to be run. In addition, the user may choose the mainframe on which a permanent file is to be maintained or from which the job is to obtain a copy of a permanent file. Refer to section 8 for permanent file information.

Each mainframe is identified by a 3-character physical identifier, abx, where a and b are any alphanumeric characters and x is an alphanumeric character unique to all mainframes connected in the network. Installation options also permit each mainframe to be optionally referred to by one or more logical identifiers. Usually, a logical identifier refers to a capability of the mainframe.

SOFTWARE CONFIGURATION

Software can be considered as consisting of an operating system (SCOPE 2) and a product set that perform as a team 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 commerical data management.

The SCOPE 2 product set includes:

COMPASS Assembler

FORTRAN (RUN) Compiler

FORTRAN Extended (FTN) Compiler

COBOL Compiler

ALGOL Compiler

APEX Program

SIMSCRIPT Language

APT Program

Sort/Merge (SORTMRG) Program

UPDATE Library Maintenance Program

7611-1 Input/Output Station Operating System

CDC CYBER Control Language

SCOPE 2 consists of a group of program modules. Some of these modules are particularly significant to the user. These are the loader, the segment loader, the record manager, the permanent file manager, and the checkpoint/restart routine.

LOADER

The primary function of a loader is to take the program unit (object module) produced by the assembler or compiler and place it into available memory linking it with related modules, if necessary, so that the program can be executed. An object module consists of specially defined loader input that designates blocks, their contents, and address relocation information. Because the location at which an object module can be loaded is variable, the object module is often called relocatable.

When all of the units comprising a program have been loaded and are ready for execution, they represent an absolute form of the program in memory. If a copy, termed a program image, of this loaded program is written on a file, this image can be reloaded upon demand and executed.

The SCOPE 2 loader loads program 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 memory allocation can be overridden if the user desires.

A number of loader options permits the user to request load maps, presetting of memory, 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 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 memory storage can also 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 is an integral part of the SCOPE 2 Operating System. It acts as an interface between the user I/O functions and the SCOPE 2 physical I/O functions. The record manager supplies the following facilities to the user.

- Recognizes a variety of data formats, some of which are industry standard
- Blocks and deblocks data
- Passes data between the user buffer and the system buffer in LCM
- Controls the transfer of large blocks of data from the system devices (mass storage or on-line tapes) to LCM, and vice versa
- Manipulates tape labels (labels can be in ANSI or user formats)
- Detects errors in format

The variety of formats allows the user to reformat specific problems to optimize problem solution.

The record manager is accessed through a simple repertoire of directives that free the user from the problems of performing physical I/O.

The speed of the logical data transfers depends only on internal LCM/SCM data rates, eliminating any direct interface between the user and devices.

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

CHECKPOINT/RESTART

Valuable machine time could be lost if a job were to terminate abnormally due to a machine malfunction, operator error, or program error. The checkpoint/restart facility captures the environment of the job on a permanent file so that if a malfunction occurs, the job can be restarted from the most recent capture point (checkpoint) rather than from the beginning. This record of the environment includes all files associated with the job.

PERMANENT FILE MANAGER

SCOPE 2 permanent file management offers the user a privacy scheme that provides security and integrity of user information. Any mass-storage file local to a job can be made permanent. When it is made permanent it can be assigned passwords limiting access to those users that supply the passwords when they request the use of the file.

SCOPE 2 includes utility routines for maintaining permanent files. These routines provide for dumping, loading, archiving, and auditing all permanent files residing on a specified set.

Permanent files may reside on removable or system set members.

FORTRAN COMPILERS

FORTRAN is the primary higher-level language used within the scientific computer industry. It enables engineers and scientific programmers to solve complex problems without requiring a substantial amount of training. Control Data Corporation provides two FORTRAN compilers with the CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, and 7600 Computer Systems.

FORTRAN	Called by the RUN control statement
FORTRAN Extended	Called by the FTN control statement

The FORTRAN (RUN) compiler has been enhanced to comply with ANSI standards and RUN version 1 compatibility and to allow the user to access and manage data in LCM.

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.

The compiler also provides 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.

The FORTRAN Extended (FTN) compiler provides ANSI FORTRAN features plus a considerable number of language extensions. In particular, it allows five modes of compilation. These modes are:

Syntax Scan Mode

Single Pass Compile Mode

Fast Compile Mode

Normal Compile Mode

Highly Optimized Mode

A programmer uses the syntax scan mode early in development of a program when he is checking the source code for syntax and spelling errors. The mode provides extremely fast compilation and returns diagnostic messages to the programmer without generating any executable code. This mode does not provide for program execution.

The single pass compile mode is used for very fast compilation and an executable program.

The fast compile mode provides fast compilation as well as producing an executable program. This mode is used for program debugging and in other cases in which the program need not be optimized.

In normal compile mode, compilation takes slightly longer than in fast compile mode. However, the resulting program runs faster because it contains code that has been optimized.

Finally, the executable program can be further optimized by selecting the highly optimized mode. Optimization results in slower compilation speeds but the resulting program runs much faster. This mode should be used for debugged production programs which are to be run on a regular basis.

FORTRAN LIBRARIES

Each of the FORTRAN compilers has associated with it a library of I/O routines and mathematical subroutines. The library for FTN is called FORTRAN, the library for RUN is called RUNLIB.

When the SCOPE 2 Operating System is installed, these libraries are generated through assembly options from a program library known as the FORTRAN Common Library. The FORTRAN Common Library contains the source programs (in UPDATE format) for the NOS/BE 1 Operating System, SCOPE 3.4 Operating System, the SCOPE 2 Operating System, and the NOS 1 Operating System.

COBOL COMPILER

The COBOL compiler combines with SCOPE 2 and the record manager to simplify the programming of business data processing problems. The compiler produces easily modifiable source programs, thus decreasing the cost of development and conversion.

The COBOL compiler very carefully adheres to ANSI standards. It provides full ANSI 68 compatibility through level 3 sort files. Data files can be sorted in conjunction with the Sort/Merge Program. A report writer facilitates flexible formats for printed reports. In addition, COBOL contains a subprogram capability based on CODASYL recommendations for standard procedures.

The compiler has been internally optimized to take advantage of the features of the CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, or 7600 Computer Systems. It interfaces directly with the record manager, removing from the user any concern for the physical recording formats of each device.

ALGOL COMPILER

ALGOL is a language used to express problem-solving formulae for machine solution. It is applicable for solving problems involving commercial, engineering, research, process control, and nonnumerical applications.

Features of the compiler include:

- Close conformance to ALGOL-60 Revised Report
- Extensive compile-time and object-time diagnostics
- Fast compilation
- Comprehensive input/output procedures
- Generation of segmented or nonsegmented programs

SORT/MERGE PROGRAM

The Sort/Merge applications program accepts input from magnetic tape or disk and constructs sorted output to user specifications on tape or disk.

Sort/Merge provides both flexibility and speed. The user can specify either ANSI standard collating sequence or any desired arbitrary sequence. A tournament sort technique ensures that optimum speed is attained.

APEX

APEX is a mathematical programming system that uses a matrix generator to arrive at optimal solutions to large linear programming problems. It produces either standard or customized reports of the results. Solution strategies involve primal and dual algorithms, mixed integer programming, generalized upper bounding, transportation, and special ordered sets. APEX includes an all-in-memory algorithm to solve a certain class of problems.

SIMSCRIPT LANGUAGE PROCESSOR

Using the SIMSCRIPT language, it is possible to simulate a real situation that changes over a time interval.

SIMSCRIPT automatically generates a timing routine which keeps track of simulated time and calls user-written routines at their scheduled times. A report generator compiler provides for many different types of reports. Random table look-up procedures and probability functions aid the user in simulating a situation in a realistic and accurate manner. With SIMSCRIPT, different configurations may be compared as to economy, efficiency, or feasibility, and reports made to management as to the best configuration for the application involved.

Although the SIMSCRIPT language was developed primarily for simulation programming, it is also a powerful language for nonsimulation problems; it offers many data processing features, as well as all the elements of a scientific programming language. Much of the SIMSCRIPT instruction repertoire is similar to that of FORTRAN. SIMSCRIPT has access to the FORTRAN Extended library routines and can reference user-coded subprograms compiled by FORTRAN Extended.

APT IV SYSTEM

APT is the acronym for Automatically Programmed Tools. It denotes a language and also a computer program (the APT system) to process statements written in that language. The result of such processing is a control tape for automatically directing a numerically controlled machine tool.

Input to the APT IV system is an APT part program. Output is a cutter-location file, a file containing successive location coordinates and other control information ready for postprocessing. Postprocessing converts the general solution produced by the main processor into the exact form needed for some particular machine. Currently, APT does not include postprocessors.

The cutter-location file is compatible with the APT III standard established by the APT Long-Range Program. Printout consists of the part program, diagnostics, and the edited cutter-location file.

COMPASS ASSEMBLER

The COMPASS Assembler language allows the user to express symbolically all hardware functions of the CPU and PPUs.

Augmenting the instruction repertoire of COMPASS are over 100 pseudo instructions that provide the user with a variety of options for generating macro instructions, controlling list output, organizing programs, and so on.

COMPASS enables the user to tailor programs to the architecture of the central processor or peripheral processors. This detailed and precise level of programming is of special use to those writing hybrid or real-time applications programs requiring code that is optimized to the hardware.

COMPASS language macros are available for communicating with elements of the operating system, among them the loader, the record manager, and the permanent file manager.

UPDATE PROGRAM

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

SERVICE/ NUCLEUS LIBRARY

The library consists of programs, routines, and subroutines used by the operating system or users.

The library resides on mass storage. User programs demand loading from the library during several phases of job processing. For example, the compilation phase requires the compiler to be loaded from the library.

LIBEDT AND COPYLB

Two utilities, LIBEDT and COPYLB, are available for library creation and maintenance. Libraries can be either system libraries or user libraries.

CDC CYBER STATION OPERATING SYSTEM

A site that has a multiple of CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, or 7600 Computer Systems, or one or more SCOPE 2 Operating Systems and a CDC CYBER 170/Model 172, 173, 174, or 175, CDC CYBER 70/Model 72, 73, or 74, or a 6000 Series Computer System may choose to link up to three systems. The 6000 or CDC CYBER Series machine serves as a batch entry station and provides the peripheral processing for the 7600, CDC CYBER 170/Model 176, or CDC CYBER 70/Model 76. The station executes under control of the SCOPE 3.4 Operating System (Figure 1-2). Job decks entered at the station card reader are routed either to the 7600 or the 6000/CDC CYBER Series computer according to 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. Three 6000/CDC CYBER Series stations can be connected to a first level PPU.

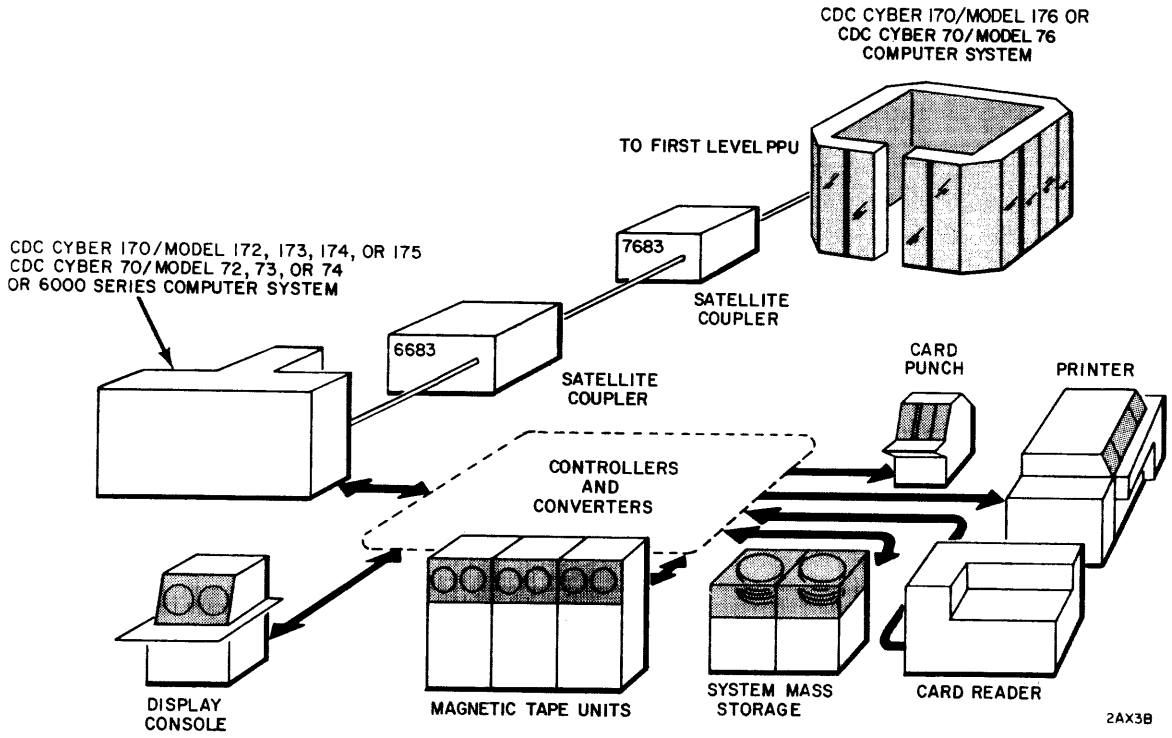


Figure 1-2. CDC CYBER Station Configuration

7611-1 INPUT/OUTPUT STATION OPERATING SYSTEM

The 7611-1 I/O Station Operating System services the 7600, CDC CYBER 170/Model 176, or CDC 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 1-3). 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

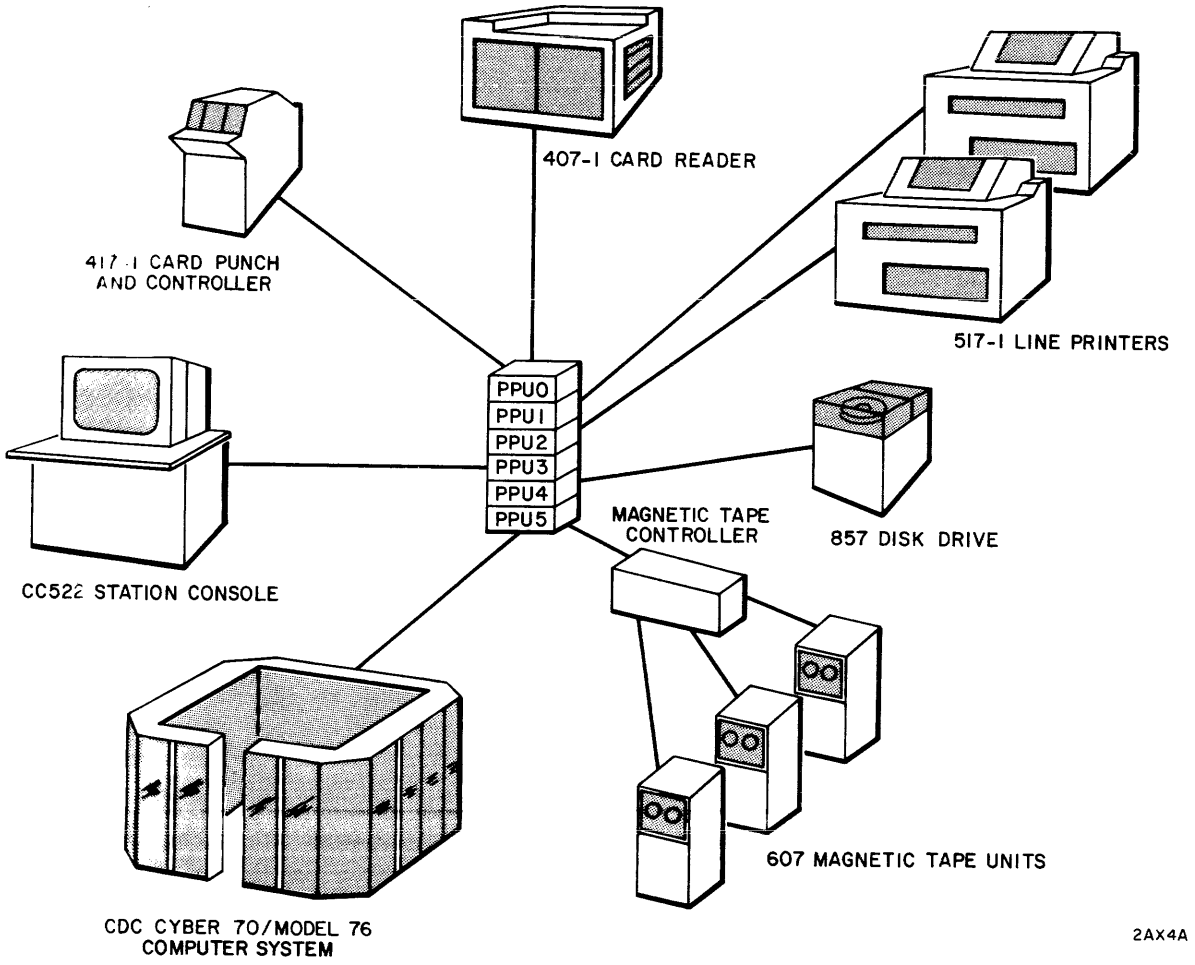


Figure 1-3. 7611-1 I/O Station Configuration

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 CDC CYBER 70/Model 76, CDC CYBER 170/Model 176, or 7600 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 1-4 shows a generalized diagram of job flow.

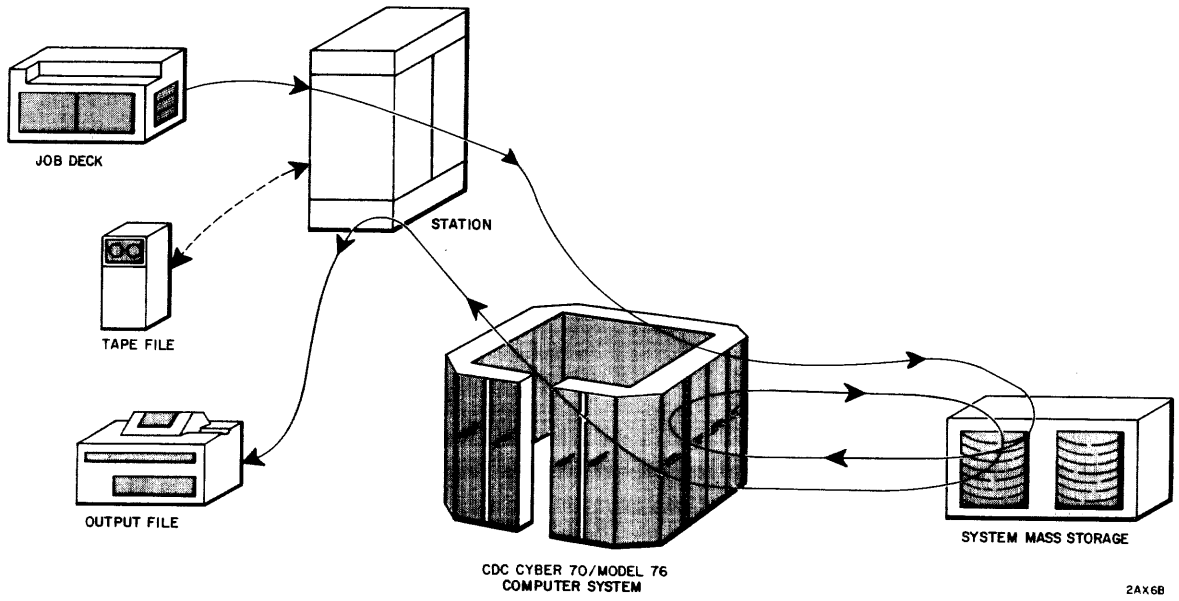


Figure 1-4. Job Flow Through System

JOB INITIATION

The first card in a job deck is a job identification statement. The operating system examines the parameters on the job identification statement 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 the dependencies are satisfied and until system resources (for example, table space in LCM) required for initiation are available. The algorithm used for scheduling on-line drives eliminates the possibility of system deadlock when the job is in some stage of processing even though the total number of units required is not available.

SCOPE 2 divides the job deck into two files (Figure 1-5); the control statement section becomes the job control file and 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.

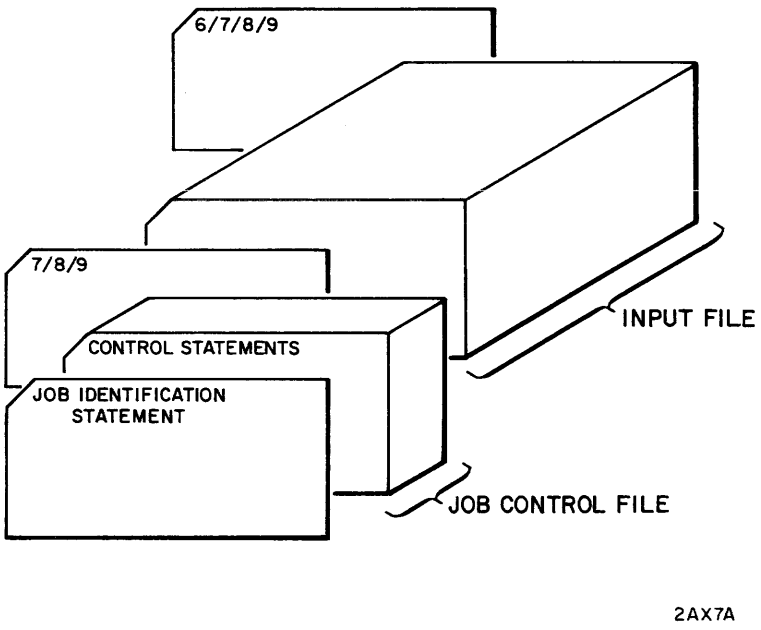


Figure 1-5. Job Deck Translation

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. If a job is waiting for resources, operator action, data transfer, or some other action, the CPU may be assigned to some other job (job switching), the job may be moved from SCM to LCM (job swapping), or its residence could change from LCM to mass storage (job rollout). The operating system changes residence of a job as needed by the job and as determined by the overall considerations of scheduling the CPU (Figure 1-6). 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 LCM and SCM. 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.

A history of the job is maintained in a dayfile for the job. At job end, the system can place one copy of the dayfile at the beginning of the file named OUTPUT and/or append one or more copies of the dayfile to the end of OUTPUT. The total number of dayfile copies printed depends on the system default value. The NDFILE control statement can change the default value. (Refer to the SCOPE 2 Reference Manual for more information on the NDFILE control statement.) 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, unless redirected by a user control statement or macro.

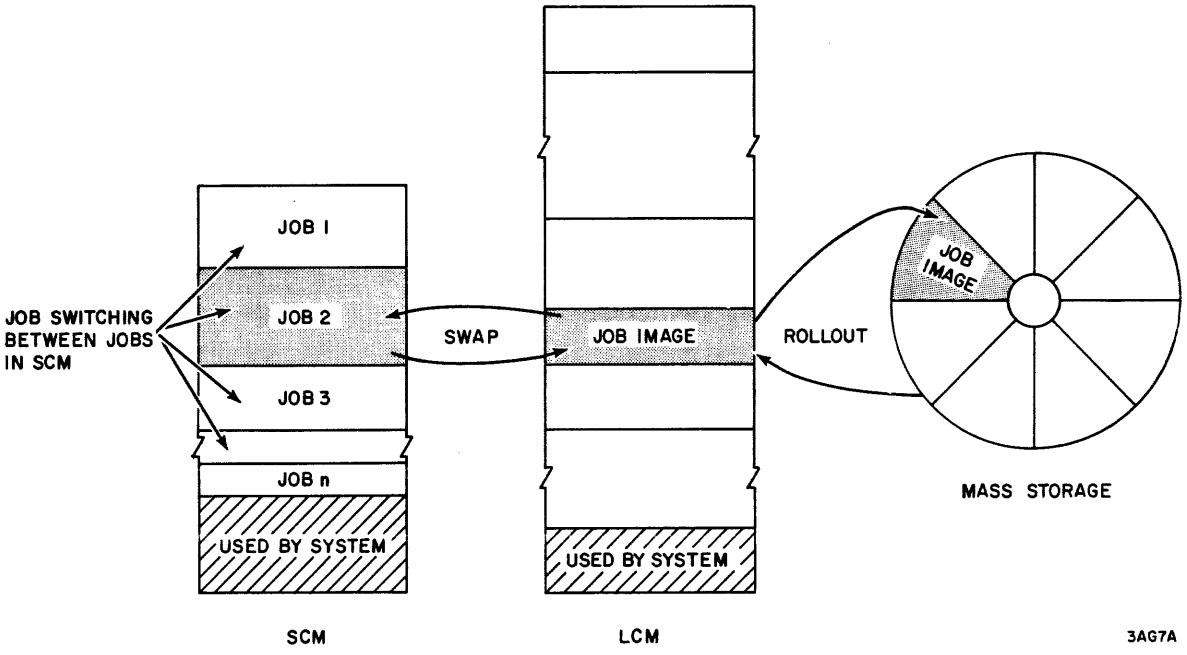


Figure 1-6. Switch, Swap, and Rollout

JOB TERMINATION

At completion of a job, the operating system returns all resources still assigned to the job to the system for rescheduling. These resources include on-line 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 and/or beginning 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 clock times. The list is terminated by the message JOB RERUN.

A dayfile usually consists of the following illustrated as items 1 through 6 in Figure 1-7.

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: contains information determined by an installation parameter (here, system resources information is given).

3. Column heads: the leftmost column identifies the clock time for each job step, the middle column 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. All times are in decimal. Entries commonly noted are the following.

Sys	System (I/O requests, etc). Sys is mainframe identifier; here, MFZ.
USR	User program, including compilers and assembler time
LOD	Loader
JOB	Control statement processing
ggg	Station processing (ggg is station identifier; for example, MFF may indicate the CDC CYBER station)

4. Station subheading: gives the clock 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, as this one did, 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.

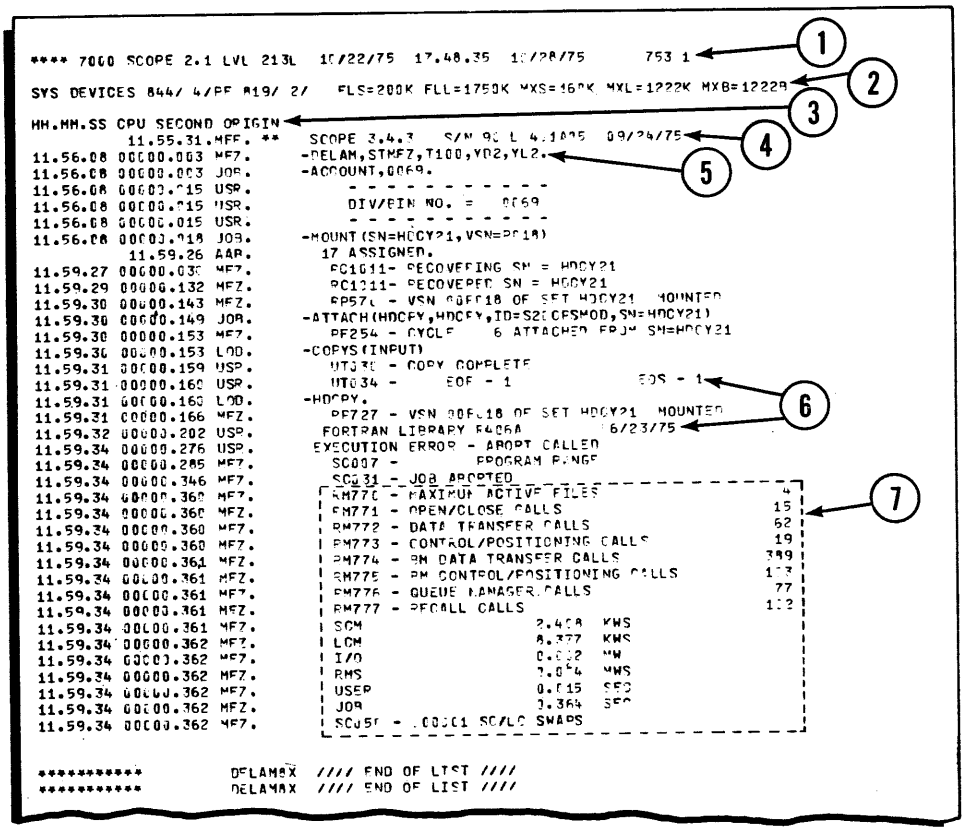


Figure 1-7. Sample Dayfile Listing

ACCOUNTING INFORMATION

When a job reaches completion, SCOPE writes a summary of basic accounting data onto the dayfile for the job (item 7 in Figure 1-7.) 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).

** • 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.

** • File open and close requests

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

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

** • Record manager/buffer manager data transfer requests for next buffer

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

** • Mass storage requests to queue manager by record manager

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

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

• 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.

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

** • Mass storage used expressed in megaword seconds

** • On-line tape unit usage expressed as tape seconds, which measure the CPU time for which the job has possession of an on-line tape unit

** • 844 Disk Storage Unit use averaged over total job time

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

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

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

INTRODUCTION TO LOGICAL FILES

A SCOPE 2 file is a quantity of information kept by the system for a user and known to the user by a 1 through 7 character symbolic name called a logical file name (lfn).

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 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, or can be maintained on removable disk packs.

In considering a request for a file, the system looks for the logical file name in lists of files local to the requesting job.

Users accustomed to SCOPE 3.4 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 alphabetic. Any reference to the file (for example, to read from it, write on it, position it, or change its characteristics) must use the logical 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 (other than INPUT) is automatically processed at job termination.

TABLE 1-1. SYSTEM FILE NAMES

lfn	Action
INPUT	Consists of job deck minus control statement section
OUTPUT	Line printer listing
PUNCH	Punching on 80-column Hollerith cards
PUNCHB	Punching in SCOPE binary on 80-column cards
FILMPR	Microfilm printing
FILMPL	Microfilm plotting
HARDPR	Hardcopy printing
HARDPL	Hardcopy plotting
	Reserved for future use

INPUT, OUTPUT, PUNCH, and PUNCHB are described in Section 9. Processors for the microfilm 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 for input and output files. 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.

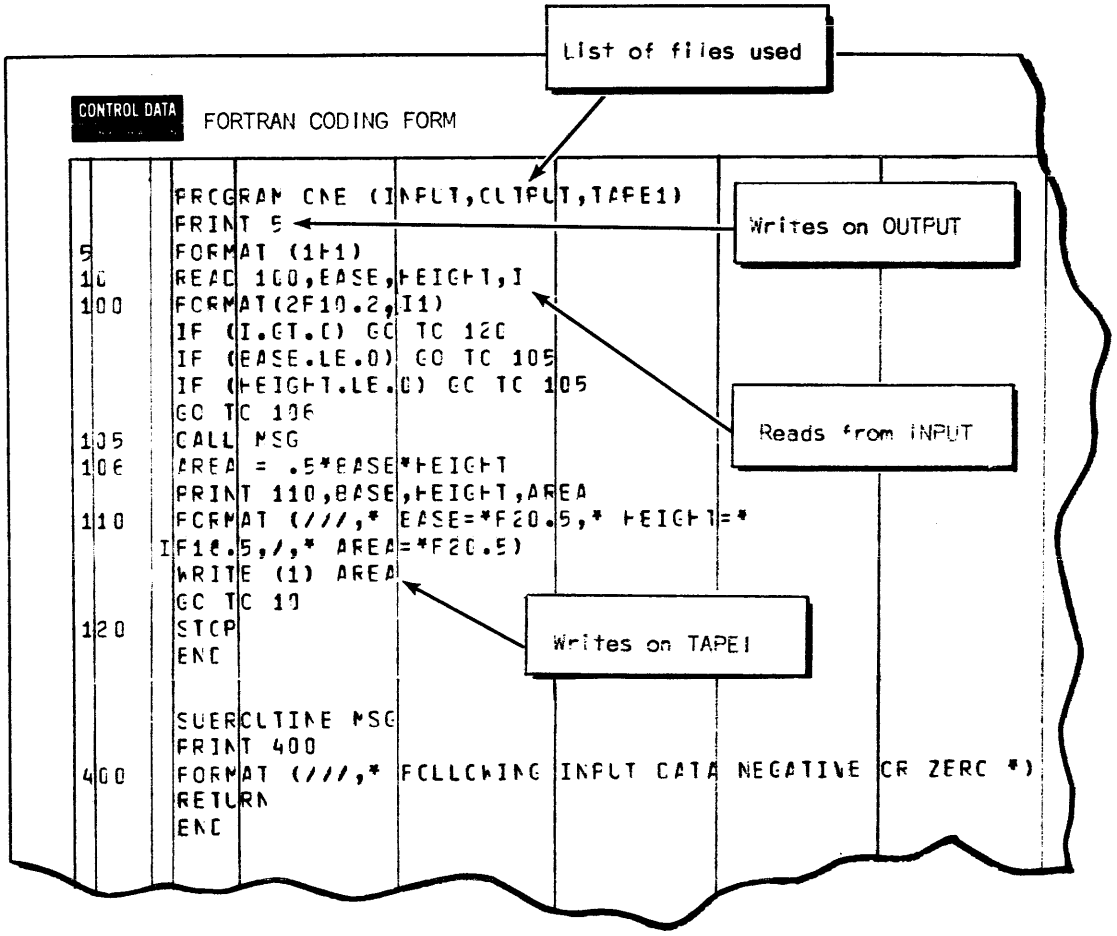
FORTRAN OBJECT-TIME FILE NAMES

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. Most READ and WRITE statements, positioning statements, and unit checking statements use unit numbers, where the number can be 1 through 99.

As a FORTRAN programmer, 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.

An analysis of this sample program, which appears many times in this publication, is contained in Appendix G.

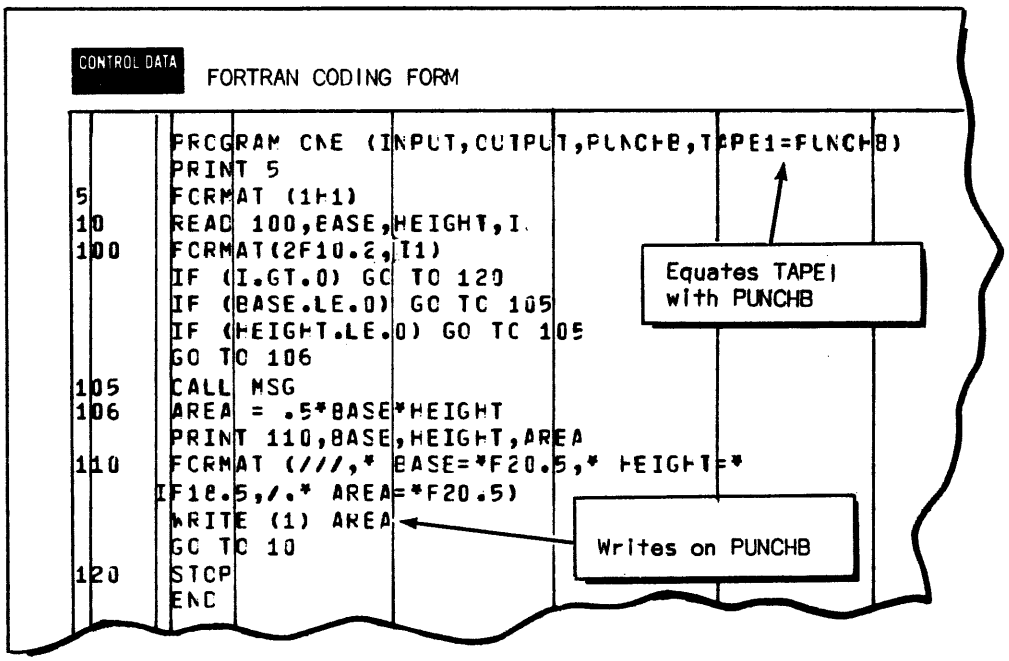


Example 1-1. Correlating File Names with FORTRAN I/O Statements

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.

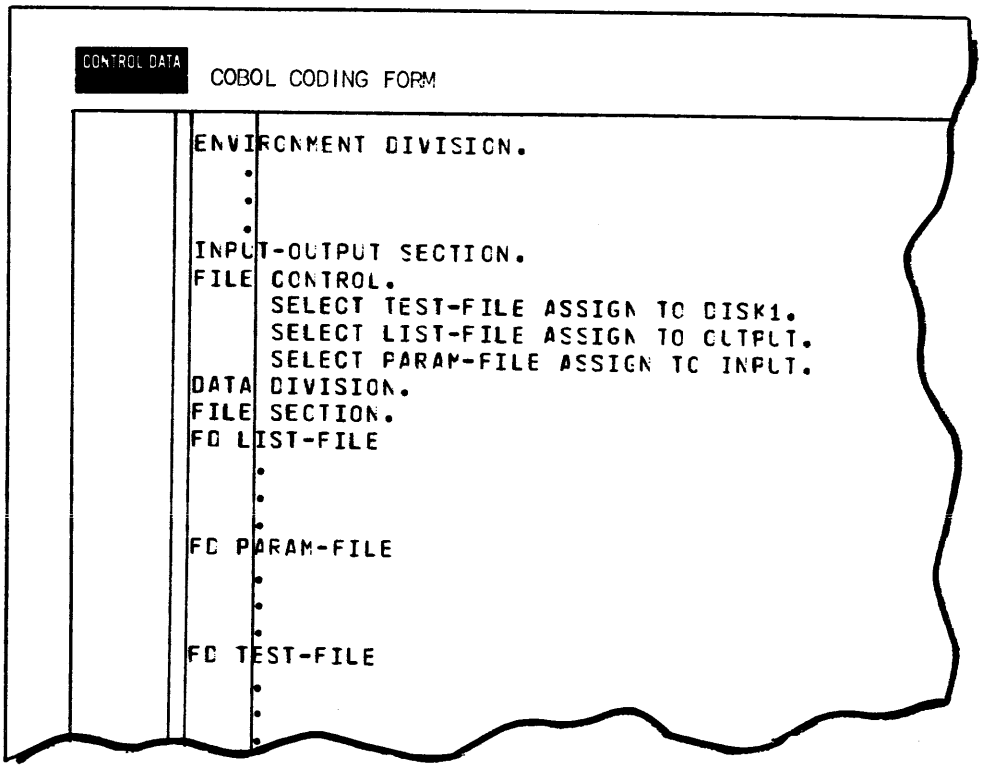


Example 1-2. Equating File Declarations on FORTRAN PROGRAM Statement

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.



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.

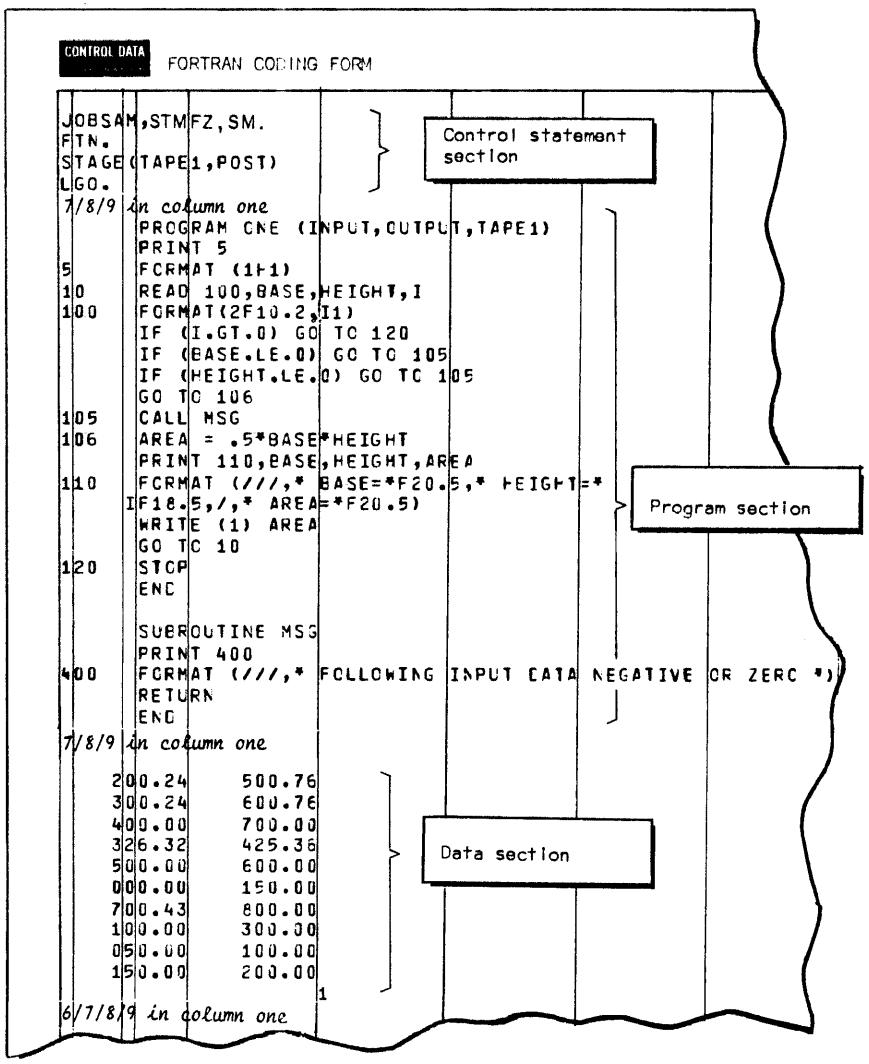
```
JOBNAME.        JOBNAME)
```

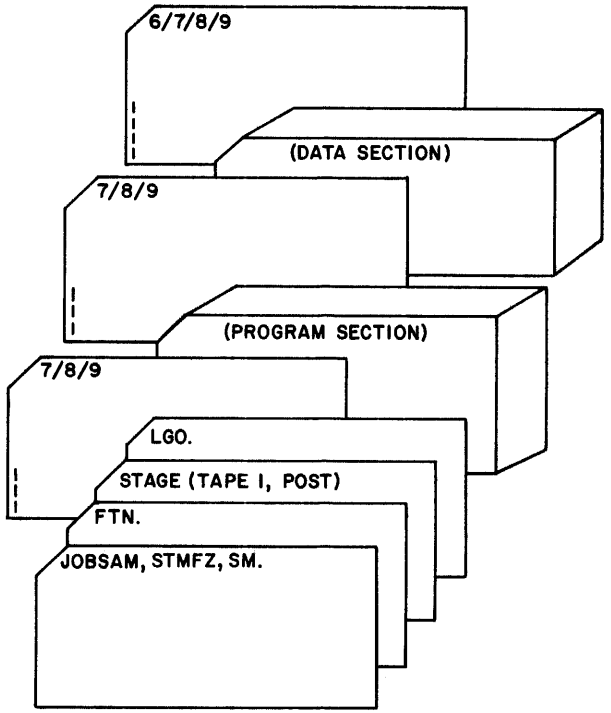
Each job must be identified by a unique name both at the station and at the central computer. Suppose you select the name PROCEED for your job and a job with this name has already been submitted. The problem is twofold. First, how does the station differentiate your job from other jobs of the same name,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





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.

STggg	Processor code for the CDC CYBER Station. This parameter is described in the following text.
Tn	CPU time limit in octal. This parameter is described under Execution Time Limit in this section.
Pn	Processing priority in octal. This parameter is described under Job Priority in this section.
CMn	Fixed number of words of small central memory allocated for the job. This parameter is not usually specified since it overrides dynamic memory management. The parameter is described with Using Memory, section 4.
ECn	Fixed number of words (expressed in octal thousands) of large central memory allocated for the job. This parameter is not usually specified since it overrides dynamic memory management. The parameter is described with Using Memory, section 4.
Dym	Job dependency string parameter. This parameter is described with the TRANSF control statement, with which it is used, in section 4.
Rn	Job rerun limit. This parameter is described in section 4, under Job Rerun Limit.
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, under Using On-Line Tapes, in section 6.
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, under Using On-Line Tapes, in section 6.

YDn	Octal number of 844-2 Disk Drives that can be used concurrently by the job (refer to section 7).
YLd	Octal number of CDC 881 Disk Packs that can be used by the job (refer to section 7).
SM	Indicator that job may require staged 7-track tapes (MT is used for on-line staging). (Refer to the SCOPE 2 Reference Manual.)
SN	Indicator that job may require staged 9-track tapes (NT is used for on-line staging). (Refer to the SCOPE 2 Reference Manual.)
SP	Indicator that the job may require an operation with permanent files on a linked main frame. (Refer to the SCOPE 2 Reference Manual.)
JCclas	clas is a 4-character alphanumeric string which is examined by the system in making assignment to a job class, provided clas is defined by the installation. Job classes are installation defined; the user should consult a site analyst to obtain a description of the job classes used by that installation. (Refer to the SCOPE 2 Reference Manual.)

CDC CYBER STATION PROCESSOR CODE

The STggg parameter is relevant for jobs entered through a CDC CYBER 170, CDC CYBER 70, or 6000 Series 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 station, the SCOPE 3.4 Operating System must determine from the job identification statement which central processor is to process your job. When the ST parameter is omitted, the job is processed at the station where it was submitted. When the parameter is included, the job is routed to the processor identified by the physical or logical identifier ggg. It is possible to route a job to the 7600 processor, or to another station if your site has multiple stations, by including the proper processor code.

7600 PROCESSING

Use STggg to unconditionally specify processing at the 7600 (CDC CYBER 170/Model 176, or CDC CYBER 70/Model 76), where ggg is the processor code for the 7600 Computer System at your site. For the purposes of this publication, STMFZ is used to identify the 7600 system. If the 7600 is not currently communicating with the CDC CYBER station, the job waits indefinitely for communication to be established.

```
JOB,STMFZ.
```

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 10_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.

$$sec_8 \approx 100 \times min$$

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,STMFZ,T1100.
```

If a job contains an EXIT statement, and the job step 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 T77777. This special value acts as an infinite time limit. It also represents the maximum possible value for the T parameter.

NOTE

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 lowest priority that can be assigned and still have the job processed is 1. If the priority is 0, the job will not be processed until the operator assigns a valid priority.

† Some installations make more extensive use of the priority parameter. At some sites, specifying any priority other than the default can be detrimental to the job turnaround. Check with a systems analyst at your site for specific information on P parameter usage.

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

```
SWIFTY,STMFZ,P2000.
```

CONTROL STATEMENTS

A control statement consists of one or more coded (punched) cards or card images.

All control statements applying to a job must be in the control statement section, which is always the first section in the deck; that is, control statements are the cards between the job card and the first 7/8/9 card. 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
- CDC CYBER control language (CCL) statements (refer to section 13)

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 central memory and loading of data into large central memory
- Call for loading and execution of file utility programs such as COPY and REWIND
- Call for execution of the object program resulting from compilation or assembly
- Specify exit paths and job termination conditions
- Control checkpoint/restart activities
- Control the order of execution of other control statements

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, or one or more blanks followed by a nonblank separator, are also interpreted as one 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. Keyword parameters are order-independent.

On the other hand, some control statements require certain parameters to be in a specific order. These parameters are positionally dependent. A statement that has both positionally dependent and keyword parameters always requires that the positionally dependent parameters be listed first.

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, a (, / or = must end the card containing the statement to be continued. It must not contain a terminator. The final card of a statement must contain a terminator. Comments cannot be continued because they follow a terminator.

more parameters)comments

name(parameters,

Each control statement is termed a job step. After successful completion of 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.

Input sections optionally follow the SCOPE control statement section and may consist of source language decks, binary object decks, data, or 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.

Each input section is terminated by a 7/8/9 card. The job deck can also be divided into units of a higher order than sections, called partitions. A partition consists of one or more input sections and is terminated by a 7/8/9 card having a Hollerith 17 punched in columns 2 and 3.

Example 2-1 represents the typical case in which the first job step that requires an input section in the job deck is the FORTRAN Extended compiler (called by the name FTN). Thus, the FORTRAN language program is the first input 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 second and final section of the input.

DIRECTIVES

It is not unusual for a program called by a control statement to derive its control information from a secondary kind of control statement known as a directive. Directives for a program can be cards in the job deck in an input section, 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.

SEPARATOR CARDS

Cards with certain unique patterns define the internal structure of a deck and identify the end of the deck.

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 ignored by SCOPE 2† but have significance to SCOPE 3.4. End-of-section cards are often referred to as end-of-record cards as a holdover from SCOPE 3.3 terminology.

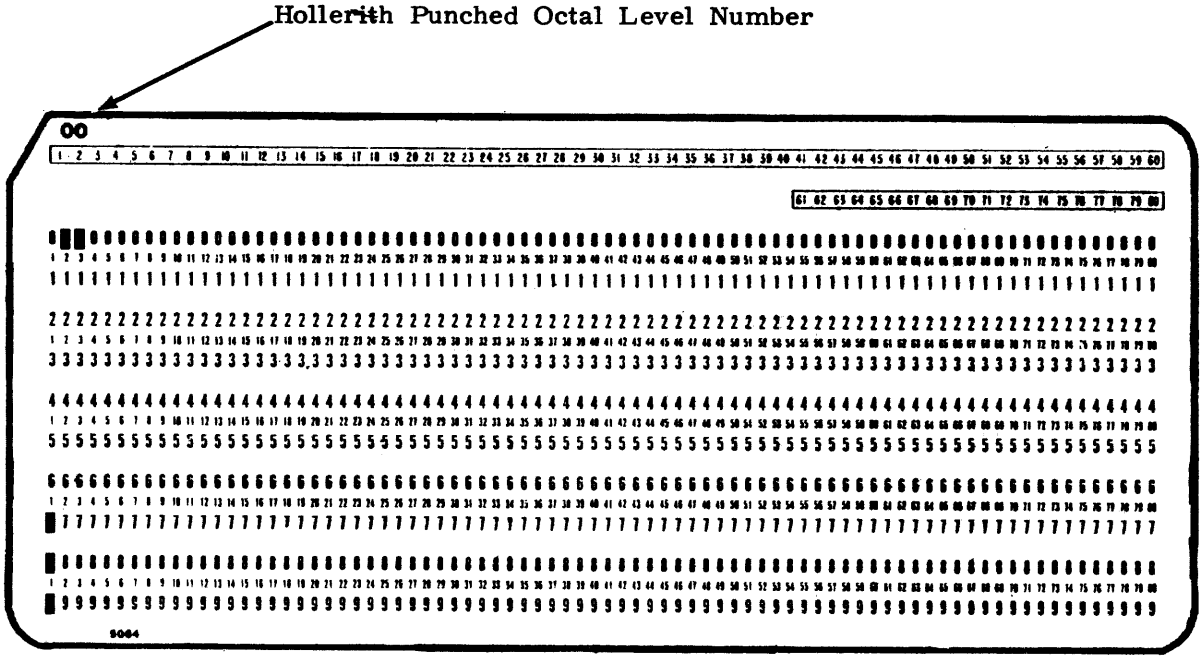


Figure 2-1. End-Of-Section Card (EOS)

† For remote entry terminals, levels 0, 1, and 2 have special significance, as described in section 9 under Station-appended Level Numbers.

END-OF-PARTITION CARD

When loading, terminate a binary deck (program 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. As will be shown later, FORTRAN and COBOL object-time routines also recognize the EOP separator.

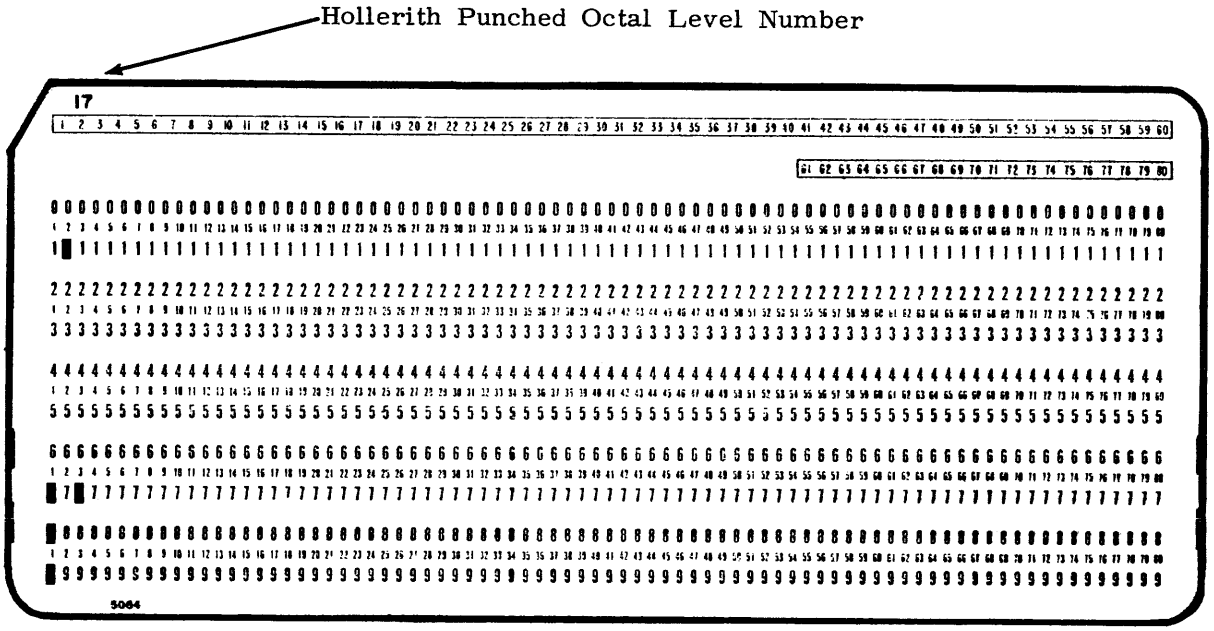


Figure 2-2. End-Of-Partition Card (EOP)

END-OF-INFORMATION CARD

The last card in a job deck is 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 usually unnecessary since the EOI serves to terminate the last section and the job deck. In previous SCOPE 3.4 systems, the end-of-information card is referred to as an end-of-file card.

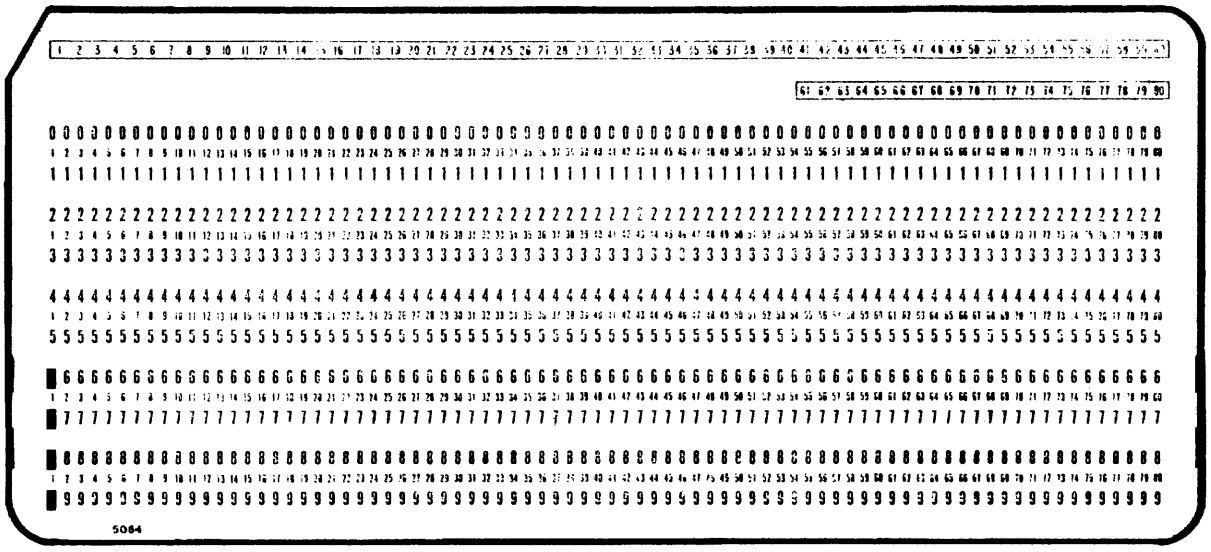


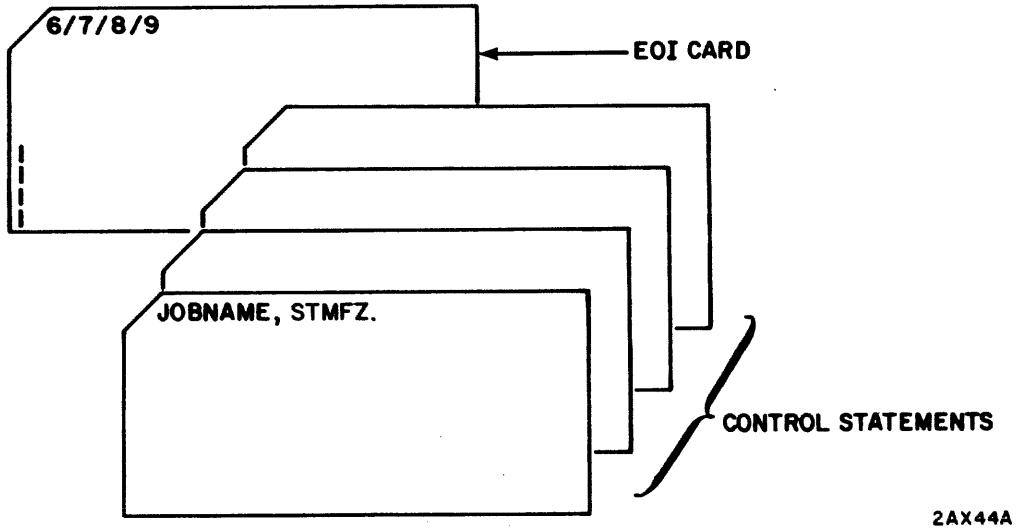
Figure 2-3. End-Of-Information Card (EOI)

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.



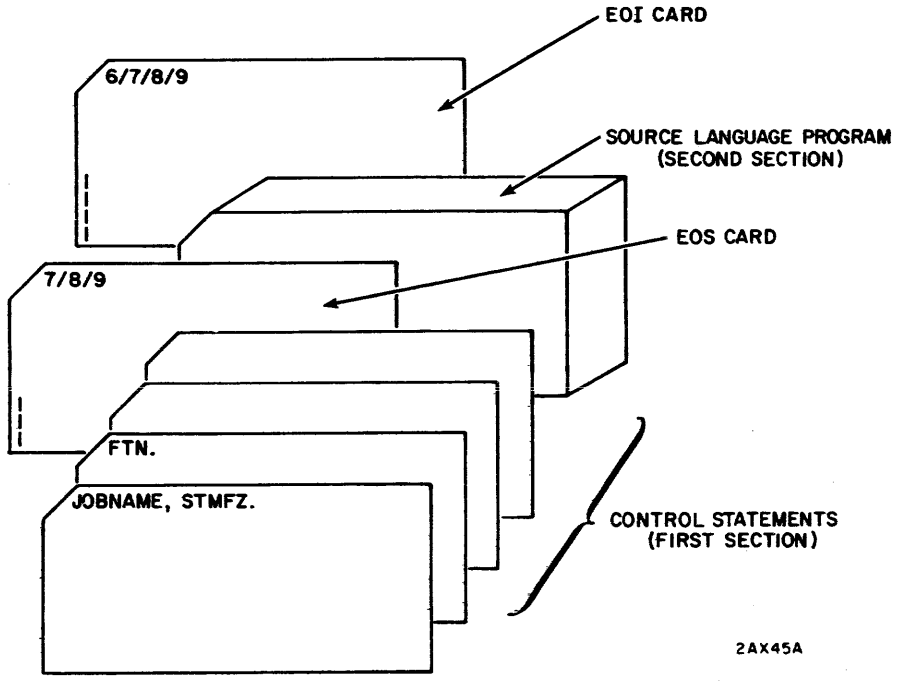
Example 2-2. Job Containing Control Statements Only

COMPILE SOURCE LANGUAGE PROGRAM

A job consists of more than one section if one or more job steps (programs called by the control statements) require card input from the job deck.

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

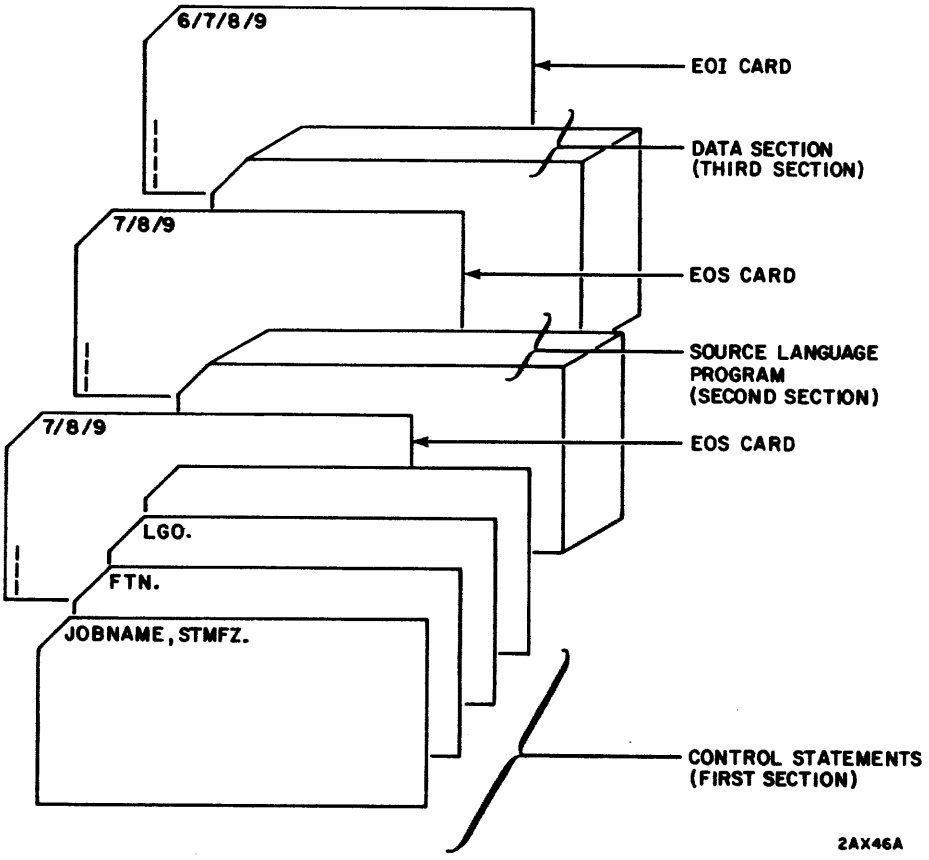
Example 2-3 illustrates a job that calls for compilation of a FORTRAN source language program.



Example 2-3. Job With Source Language Program

COMPILE AND EXECUTE

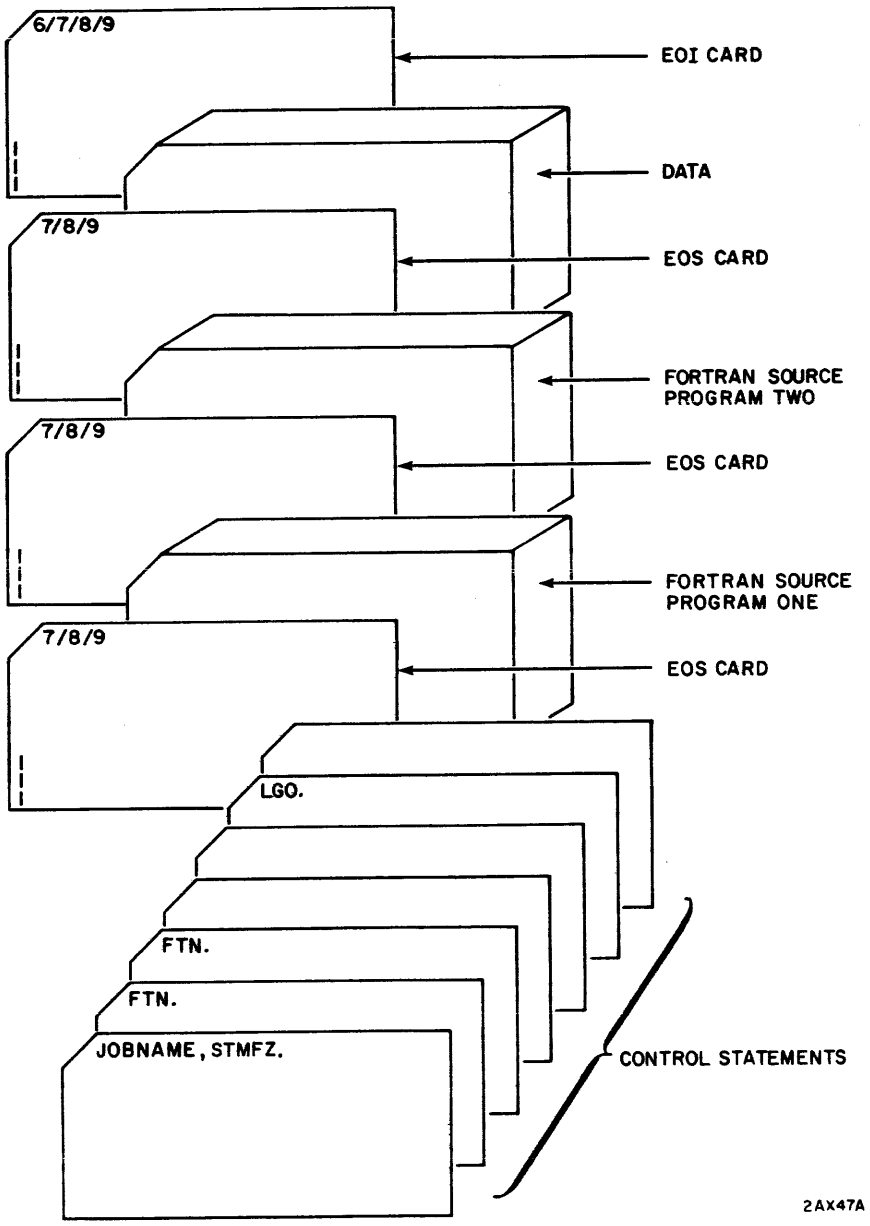
Typically, execution of the program compiled by the compiler is called for following the compilation. In this example, execution is requested through the LGO statement. LGO is a file on which all the compilers and assemblers place object programs unless some other file is specified. This statement is 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

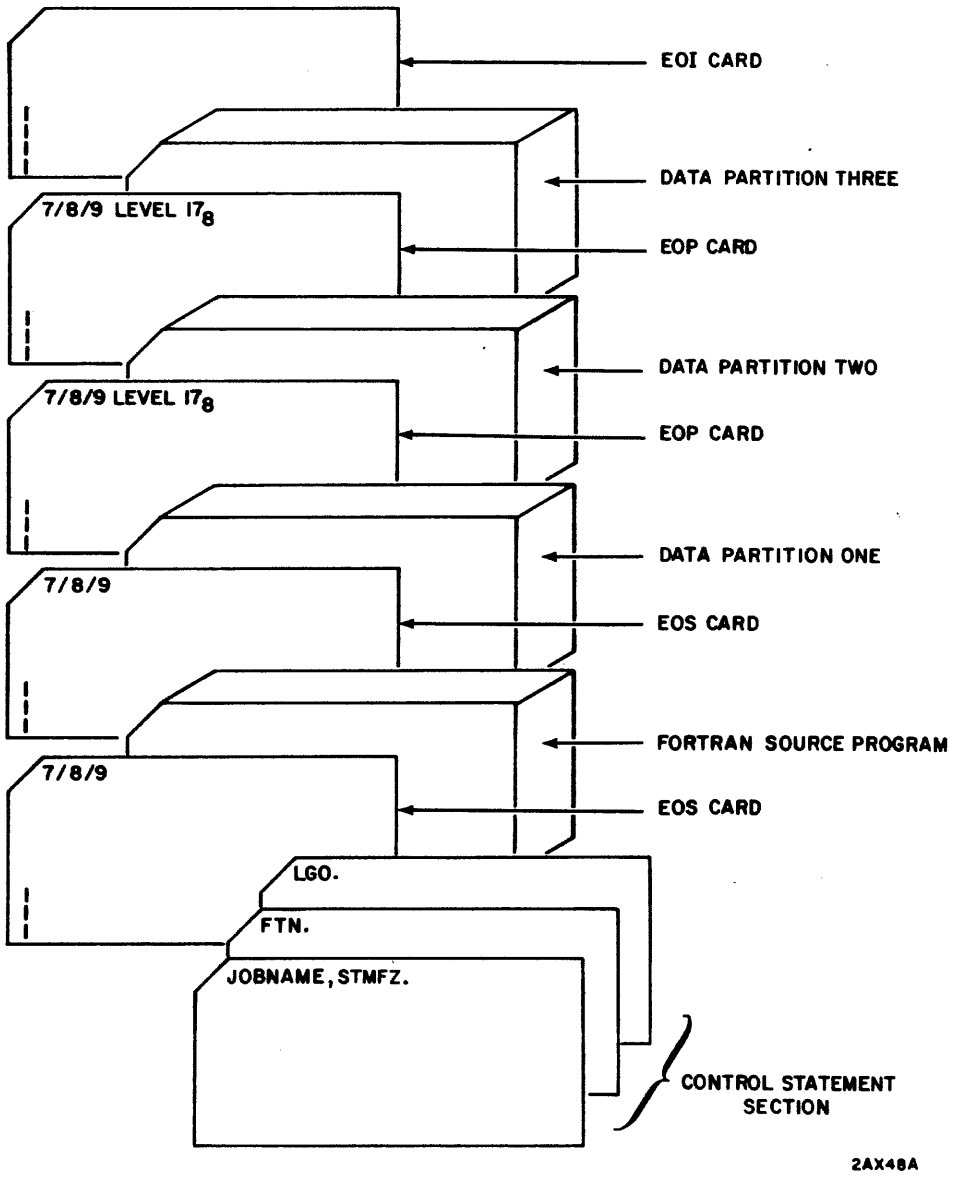
Very elaborate jobs are possible, such as those that include compilation and execution of more than one program. Example 2-5 illustrates a job containing two FORTRAN programs. Both calls to the compiler write the object programs on a file named LGO. The object programs are loaded and executed as a single 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 PROCESSING 3

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

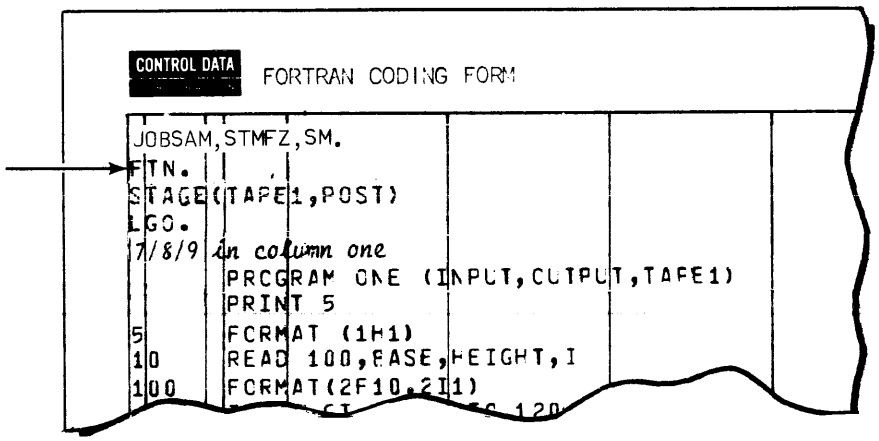
The most common first step in job processing is translation of the source language program into an object program, that is, into machine language. This occurs through compilation if the source language program is written in a compiler language such as FORTRAN or COBOL, or through assembly if it is written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

The request for compilation or assembly is a request for SCOPE to load the compiler into small central 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.
ALGOL-60	ALGOL.
SIMSCRIPT I.5	SIMI5.

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



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.

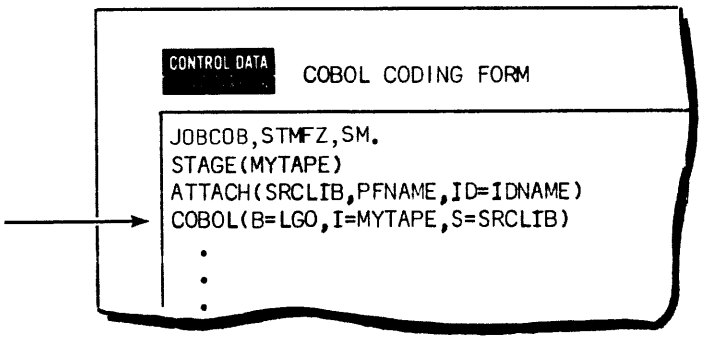
As you become more familiar with the compiler or assembler language, you will want to take advantage of the several programming options available on the compiler or assembler request statement. These options have a direct effect on the compilation or assembly.

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 written on file LGO, 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.



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	ALGOL
Reads source language program from INPUT file.	omitted or I or I=INPUT	omitted or I=INPUT	omitted or I=INPUT	omitted or I=INPUT	omitted or I or I=INPUT
Takes source language from file named lfn. †	I=lfn or INPUT=lfn	I=lfn	I=lfn	I=lfn	I=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	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 or P	B=lfn	B=lfn
Punches binary cards of object programs.	B=PUNCHB	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 series of control statements called a loader control statement sequence.

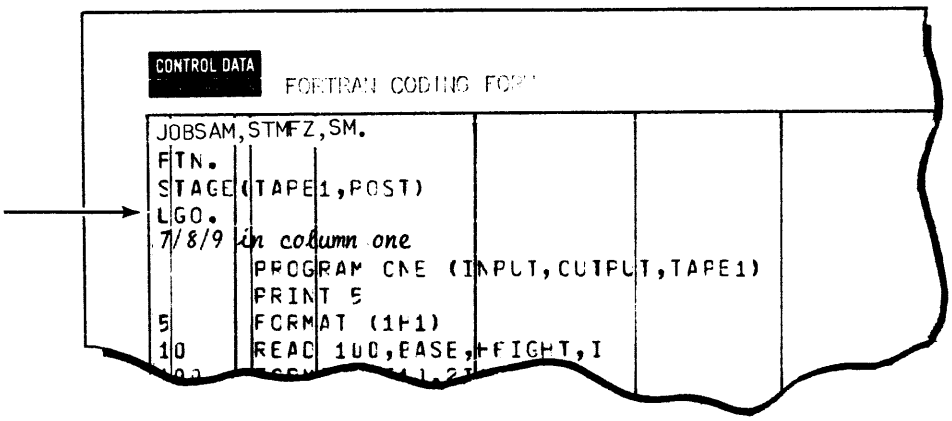
COMBINED LOAD AND EXECUTE REQUEST

THE LGO STATEMENT

After the program has been compiled or assembled, the most direct way to load the object program into small central 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 operating system "load and go". In Example 3-3 the arrow points to the LGO statement.

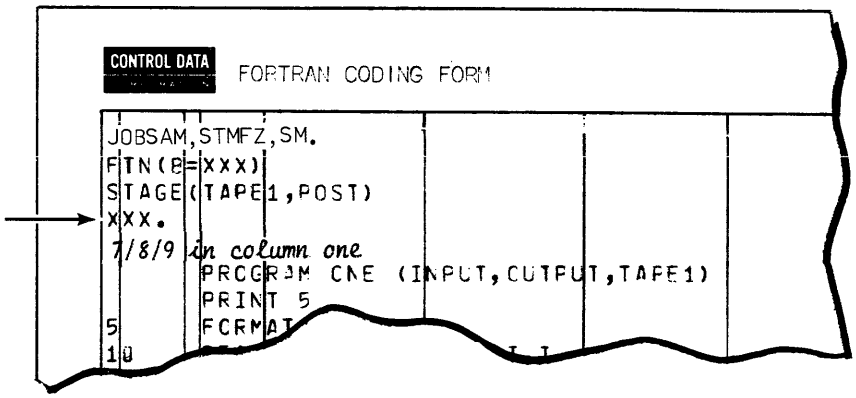


Example 3-3. LGO File Name Statement

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 all the language processors is loaded and executed as a single program.

The compiler options permit you to specify another file as 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.



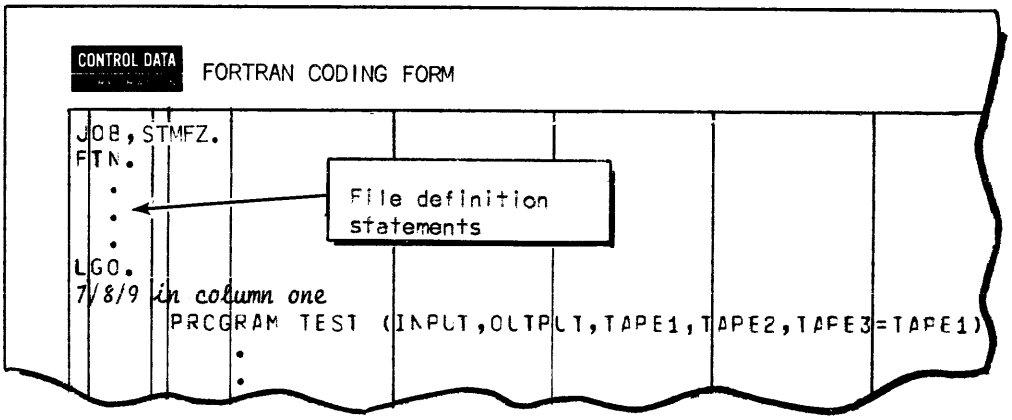
Example 3-4. Renaming the Load-And-Go File

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 files are those in the PROGRAM statement.

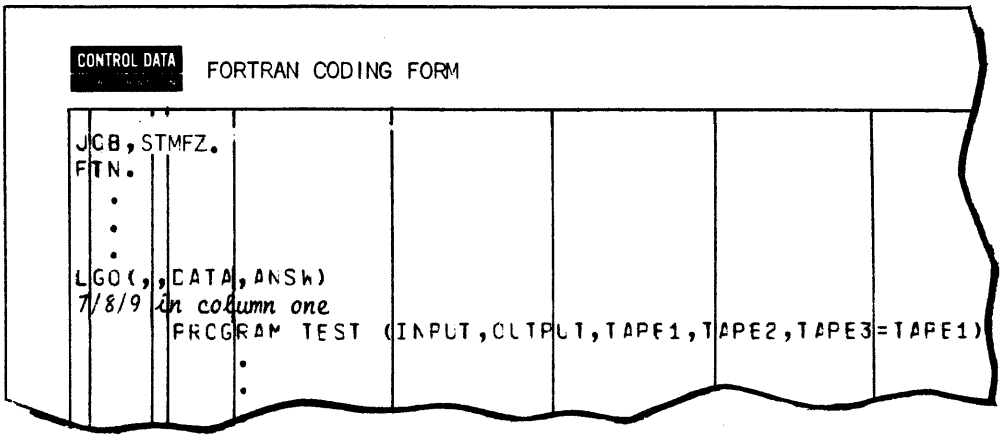
In Example 3-5, the program uses files INPUT, OUTPUT, TAPE1, and TAPE2.



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.

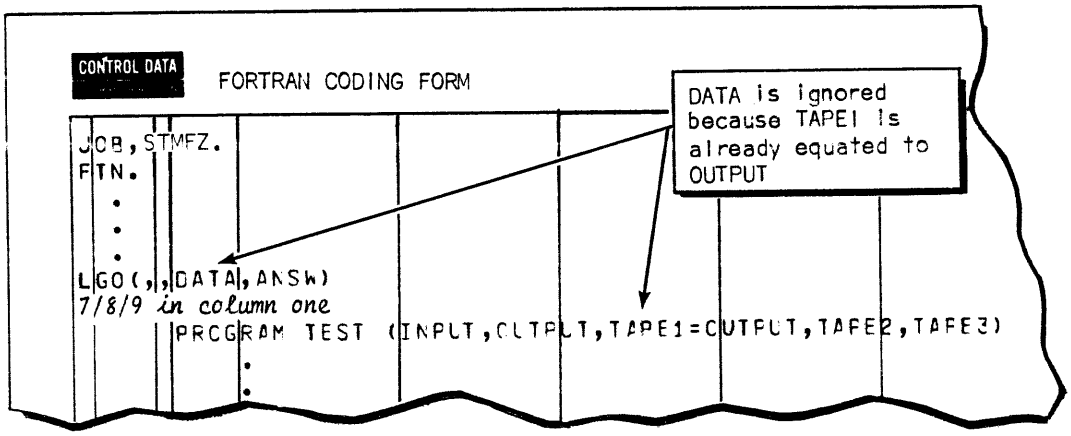
In Example 3-6 the program still uses files INPUT and OUTPUT. However, files TAPE1 and TAPE2 have been replaced by the files named DATA and ANSW, respectively. TAPE3 which was equated to TAPE1 is also replaced by DATA. Any reference to TAPE1 or TAPE3 in the program actually references DATA. Any reference to TAPE 2 actually references ANSW.



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.



Example 3-7. Precedence of Equating File Names

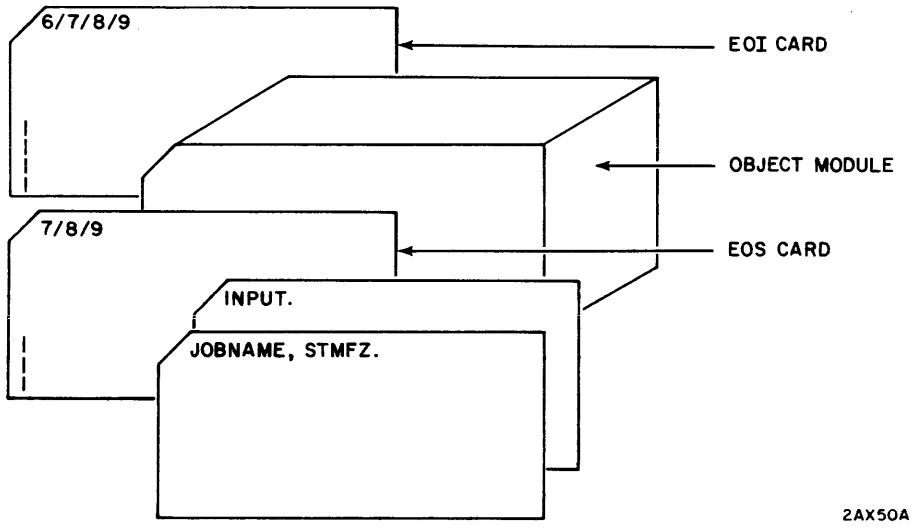
LOADING FROM INPUT

If you have on hand the punched binary 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. Terminate the binary 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 binary 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 into your SCM and LCM fields and then execute it. System verb 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 verbs given in Table 3-3.

A file from which loading is to take place cannot have the same name as one of these verbs.

TABLE 3-3. SYSTEM VERB TABLE

SCOPE 2 Verbs			Loader Verbs
ACCOUNT	DSMOUNT	PAUSE	EXECUTE
ADDSET	DUMPF	PURGE	LDSET
ALTER	DUMPOST†	RECOVER	LIBLOAD
ATTACH	EXIT	REDUCE	LOAD
AUDIT	EXTEND	REQUEST	NOGO
CATALOG	FILE	RESTART	SEGLOAD
CKP	GETPF	RFL	SLOAD
COMMENT	LABEL	RTRVSIF	
DELSET	LIBRARY	SAVEPF	
DISPOSE	LIMIT	SETNAME	
DMP	LOADPF	STAGE	
DMPECS	MAP	SUMMARY	
DMPFTB	MODE	SWITCH	
DMPJSL	MOUNT	SYSLIBE	
DMPJT	NDFILE	TRANSF	
DMPL†	PASSWRD	VSN	
DSJW†			

† Reserved for system use.

CCL verbs described in section 13 are as follows:

BEGIN, DISPLAY, ELSE ENDIF, ENDW, IFE. REVERT, SET, SKIP, and WHILE.

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 verbs. In interpreting a control statement, the system first determines whether the control statement is one of the verbs. If it is, it performs the requested action. If the statement is not a verb statement, the system checks to see whether the statement is a name call statement. If the statement is not recognized as either a verb statement or a name call statement, a control statement error occurs.

NOTE

Many system routines and programs use internal file names in the form ZZZZZxx. Users should avoid assigning names beginning with five Z's to their files.

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.

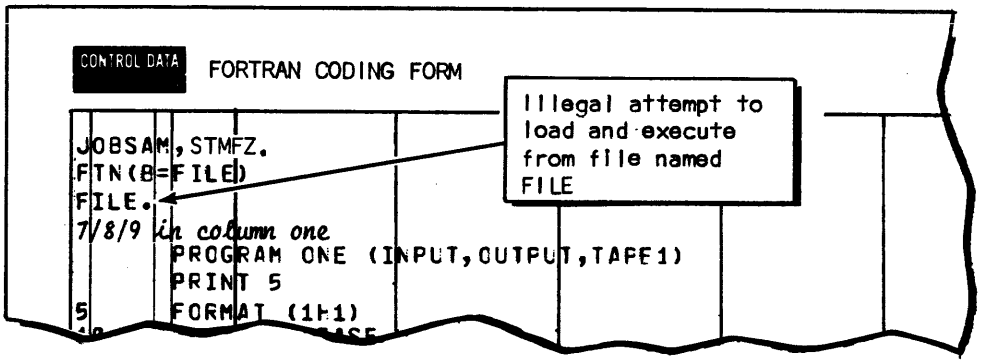
$$lfn(p_1, p_2, p_3, \ldots, p_n)$$

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 verb 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 system verb statement.

Example 3-9 illustrates a futile attempt to load from a file that has the same name as a verb statement. In this case, the system interprets the FILE statement as a verb, not as a name call. 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.



Example 3-9. Illegal Verb/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.

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

In interpreting the statement, SCOPE 2 first determines that it is not a verb statement by checking eptname against the list of verbs, 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.

For FORTRAN and COBOL programs, the program name is a primary entry point. For COMPASS programs, the name is not an entry point. The ENTRY pseudo instruction defines the name as an entry point.

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 card or two consecutive end-of-section cards are 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.

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.

PROGRAM IMAGE MODULES AND HOW THEY ARE LOADED

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

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

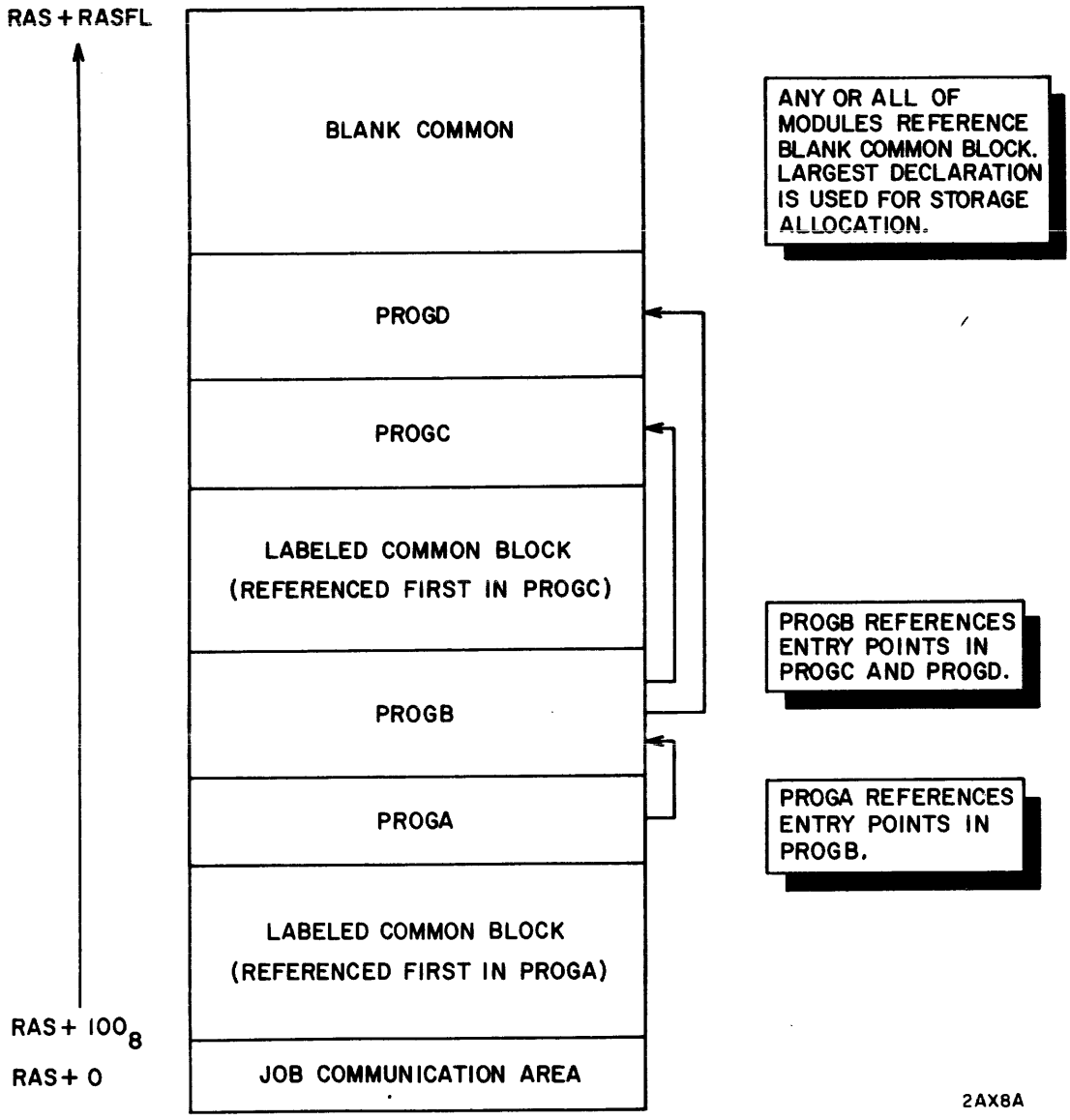


Figure 3-1. Structure of Loaded Program

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.

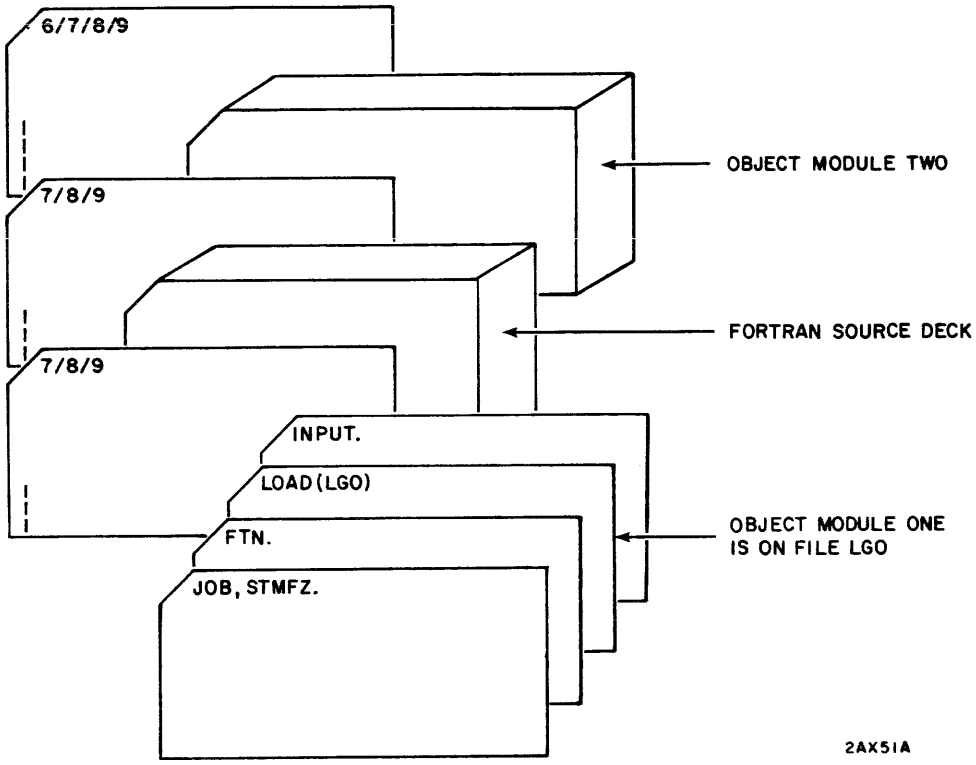
The following are equivalent: either sequence causes files ALPHA, BETA, and GAMMA to be loaded and executed as a single program. Remember that the files must be local to the job.

```
LOAD(ALPHA,BETA,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(LGO)
NOGO.
```

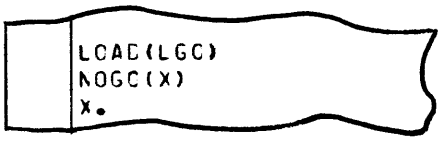
The NOGO tells the loader that no more loads are to take place and that execution is not to occur. If a map is requested, the loader generates the map but does not execute the loaded program.

USING NOGO TO GENERATE PROGRAM 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 program 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.



Generate program image module
Load program 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 verb 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, four SCOPE control statements, COMMENT, 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.

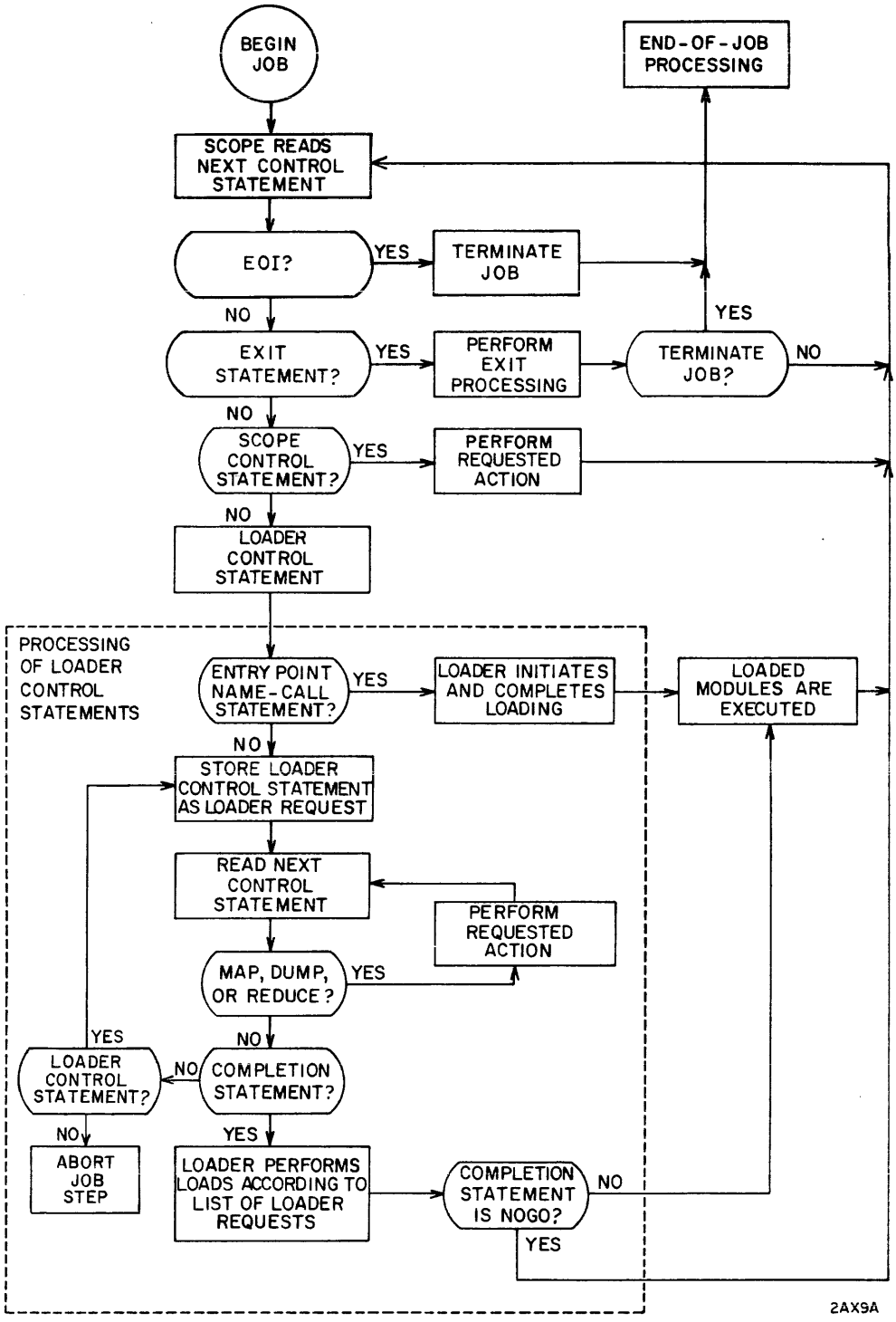
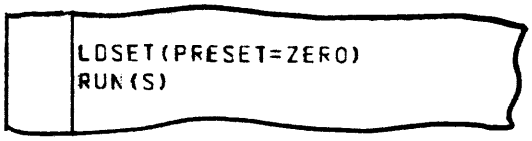


Figure 3-2. Processing of Control Statements

Thus, the following sequence is illegal:



Begin load sequence
Entry-point name illegal
in loader sequence

However, the following is legal:

Begin load sequence
File name completes
loader sequence

SELECTIVELY LOAD MODULES FROM FILES

Suppose a file has a number of object or program 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.

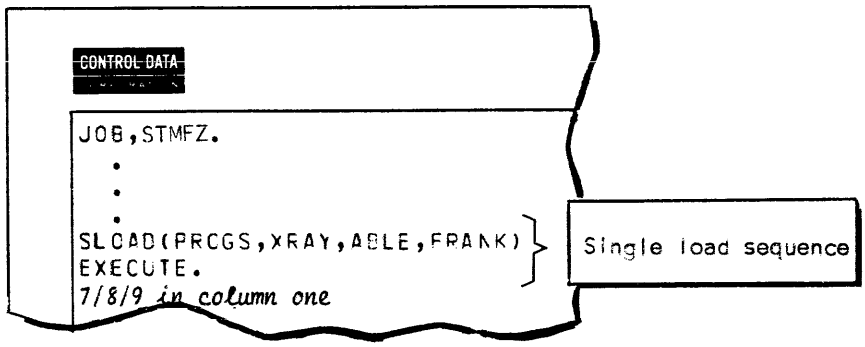
SLOAD(lfn, $modname_1$, $modname_2$, $modname_3$, . . . , $modname_n$)

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.



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$_n$)

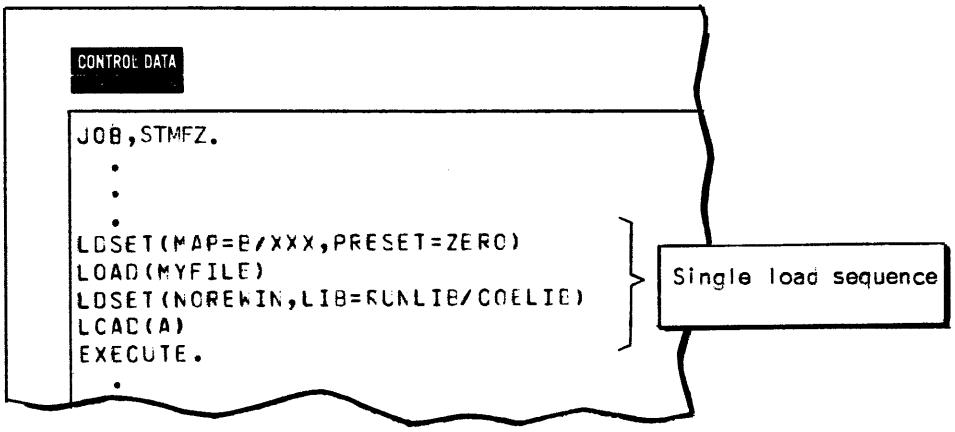
Each option consists of a keyword which may or may not be equated to a parameter. You can use LDSET statements anywhere in the load sequence.

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 Options, section 12. 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 in this section. The default is the system library declared by the compiler during compilation.
MAP=p/lfn	Specifies degree of load map produced and file on which map is to be written. This is described under Obtaining Load Maps, section 12. The default map is determined by an installation parameter 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 Memory, section 4.
REWIND and NOREWIN	Specifies file positioning prior to load. Default is REWIND. This option is described under Rewinding of Load Files, section 4.

Example 3-12 illustrates a load sequence that contains two LDSET statements. The first statement requests that memory be preset to zeros and that a partial map be written on file XXX. The second statement requests that subsequent files be not rewound and that libraries RUNLIB and COBLIB be used for satisfying externals.



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 program 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 program 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 a library named RUNLIB. 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 program 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 also 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. The NUCLEUS library is not considered as part of a library set.

THE SEARCH FOR ENTRY POINT NAMES

As previously noted, 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 system verb 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, the FTN compiler references FORTRAN, and the COBOL compiler references COBLIB.

The system searches all libraries in the set for all unsatisfied externals. If there are any externals left unsatisfied, the user can be sure they are not in any library of the set.

NUCLEUS is not searched. It contains program 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.

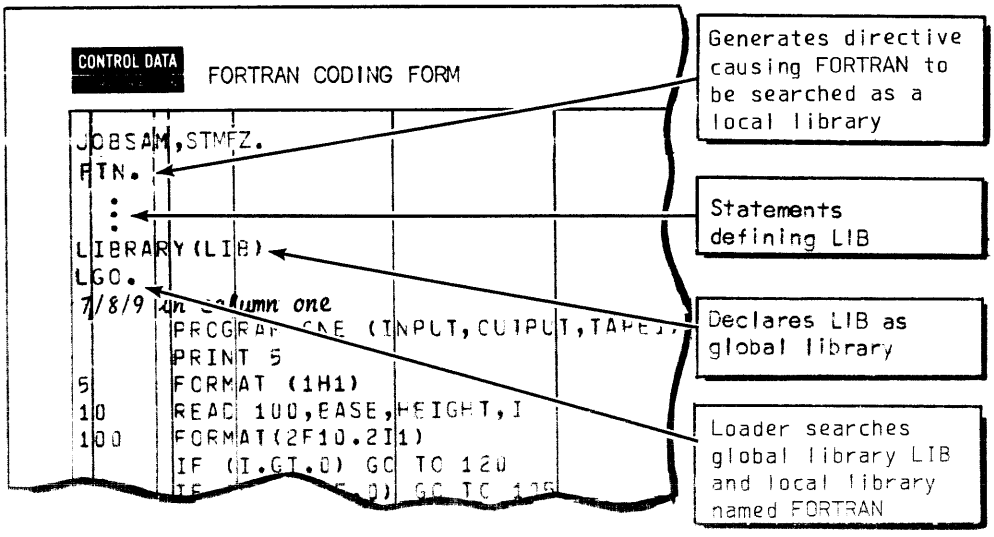
DEFINING THE GLOBAL LIBRARY SET

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 user 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.

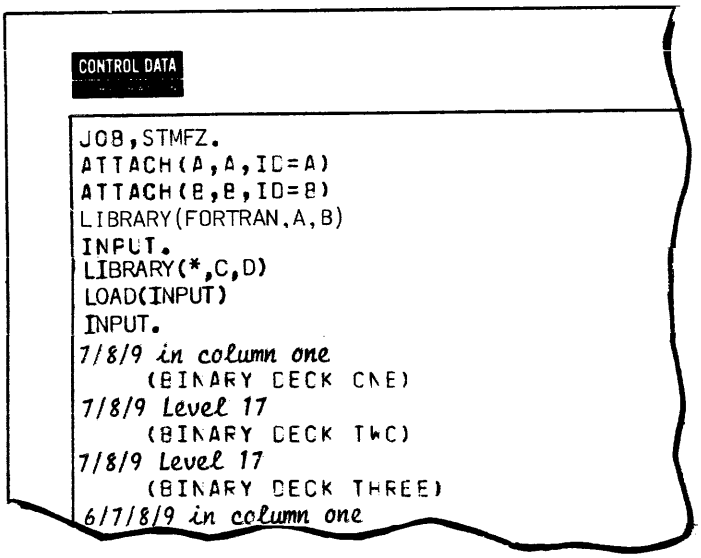
Example 3-13 illustrates a job that creates user library, LIB. The LIBRARY statement declares this library to be a member of the global library set to be used for satisfying externals when LGO is loaded. It takes precedence over FORTRAN, the system library automatically entered in the local library set.



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 system 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 and user libraries A, B, C, and D.



Example 3-14. Combining New and Old Library Sets

DEFINING THE LOCAL LIBRARY SET

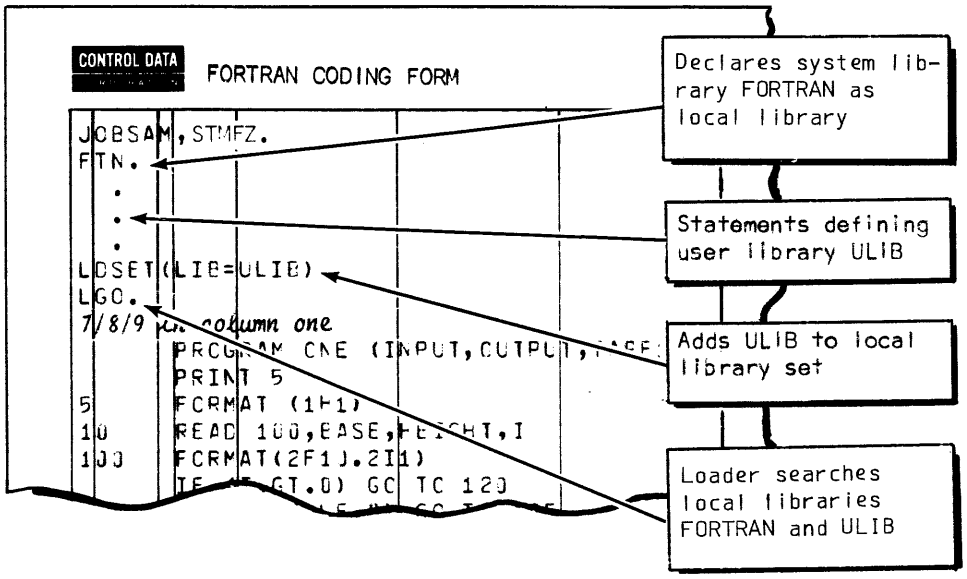
Use the LIB option on the LDSET loader control statement to declare a library local to the load sequence. Place the LDSET statement inside the load sequence for which the library is to be used. The LDSET statement is a loader control statement and either initiates or continues a load sequence.

LDSET(LIB=libname$_1$/libname$_2$/libname$_3$/libname$_n$)

Library files can be either system or user libraries. Libraries are searched in the order listed.

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 all parameters. This also clears any previous 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. LIB can occur in any parameter position in the LDSET statement.

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



Example 3-15. Defining a Local Library

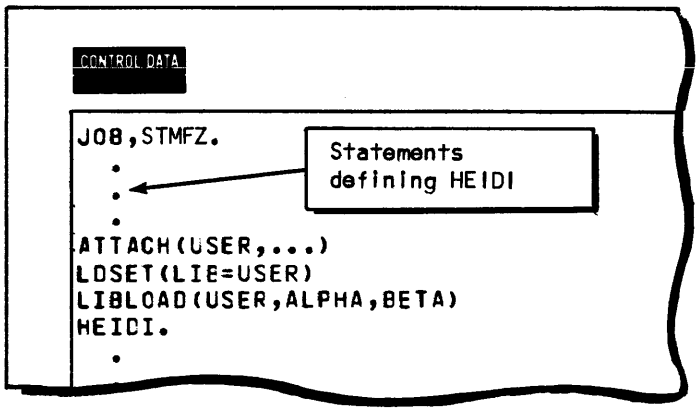
LOADING DIRECTLY FROM LIBRARIES

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.

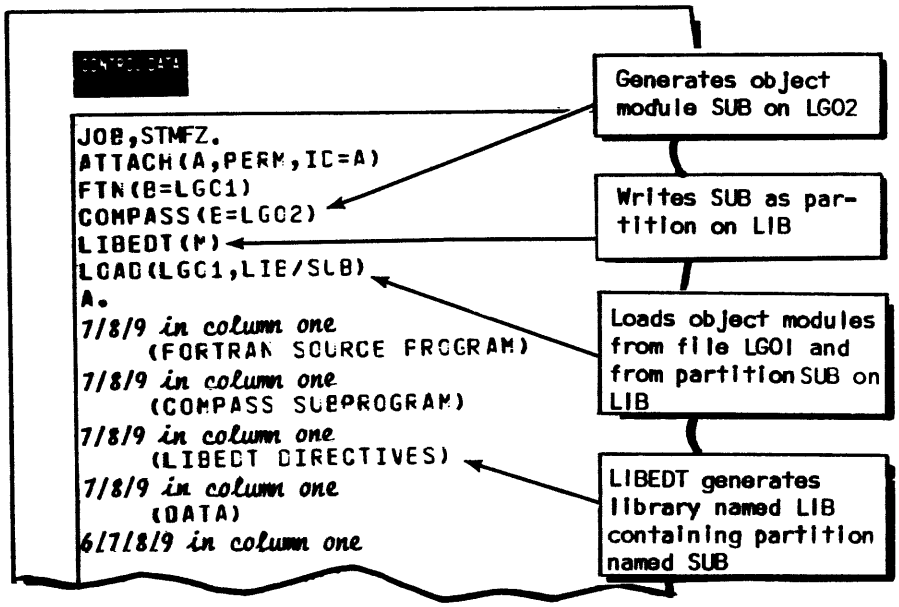


Example 3-16. Direct Load From Library Using LIBLOAD

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 program 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.

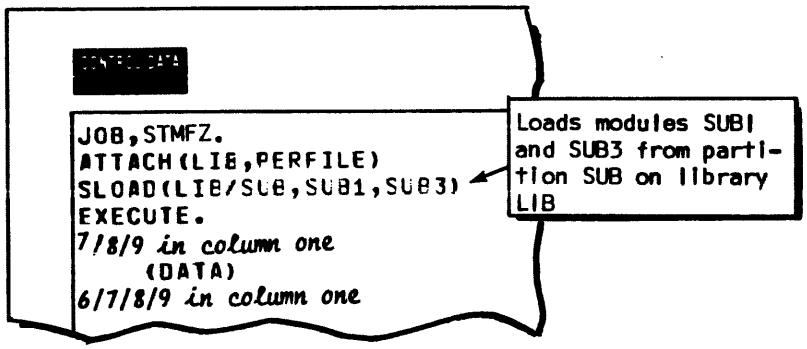
LOAD allows loading of modules from both libraries and files, as shown in Example 3-17.



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.



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 omit this section and continue with section 5.

USING MEMORY

USER SCM

Each user job is composed of at least an SCM field length image, the length of which depends on the memory management mode selected by the user and may vary from job step to job step as the job progresses through execution.

USER LCM

A user job may also have an LCM field length image which may contain operands for the user program. The length of the user LCM image is also controlled by the way the user structures his job and by the user's selection of memory management mode.

JOB SUPERVISOR LCM

In conjunction with any user job there is always a fixed-length job supervisor (JS) resident contiguous with the user's SCM image. This job supervisor also maintains a work area in LCM, outside the user LCM, if any, which contains various tables required for processing the job. The size of JS LCM is variable (minimum length is 1024 words), depending upon job I/O activity.

I/O BUFFERS IN LCM

SCOPE 2 also allocates and manages all I/O buffers associated with files used by a job. These buffers are contained in LCM outside the user LCM field length. Each buffer is 512 words. The number of buffers allocated varies according to the number of files used by the job and according to current file I/O activity during job processing. In particular, the following system files are assigned to each job regardless of the job's other file activities.

The job dayfile

The job control file containing the job control statements

The SCOPE 2 file containing the system library (NUCLEUS). This file requires 0 through 12_8 buffers

The absolute minimum number of buffers for a job is four.

MAXIMUM AVAILABLE LCM

The maximum LCM field that a user could request is either $400,000_8$ or $1,400,000_8$, depending on the half or full system configuration.† If a user attempts to request more LCM than is available in the system, he receives a message informing him that the LCM limit per job has been exceeded.

The maximum LCM field length assumes a minimum of four system buffers. In actual practice, this minimum is often difficult to achieve. However, by using the following techniques, the user can expect to have access to the maximum LCM for his job. A user may not need to employ all of these techniques to run a job using LCM. In fact, if all these procedures are required, it may be a good idea to reconsider the program's use of LCM.

1. Use a RETURN ($lfn_1,\ldots,lfn_n$) statement to return all files no longer needed. If you have created one absolute program image of your program, the following files can be returned.

ZZZZZNC	System NUCLEUS library
LGO	Relocatable binary file

   Although the library file names listed above are quite common, system library names may be different at your site. Check with a systems analyst if you are uncertain about system library names.

   You can also return any local or global library files you may be using.

2. Restrict the number of buffers allocated for any I/O operations on a given file.

   a. For mass storage scratch files, you can ensure that only two buffers are allocated by using the T parameter on the REQUEST statement for each mass storage file or on the STAGE statement for each staged file.

      REQUEST(lfn, T) or STAGE(lfn, $\begin{matrix}\text{PRE}\\\text{POST}\end{matrix}$, T)

      Refer to Transfer Unit Size, section 9.

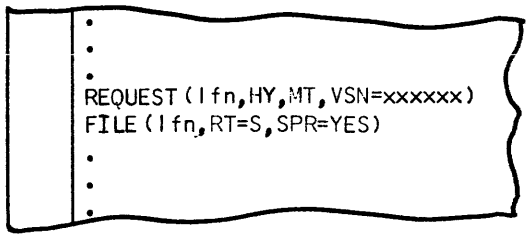
   b. In addition, the number of buffers per file will be further reduced by one if you change the file organization from sequential (default) to word-addressable by specifying

      FILE(lfn, FO=WA)

      Refer to Access Methods, section 5.

† A site may elect to reduce or increase the maximum during system deadstart. More field length is available with large LCME memory.

c. Buffer space for on-line tape files is a function of physical tape block size. Normally, tapes with short blocks will use up to five LCM buffers (5000 octal words). Tapes with long physical records require additional buffers. The maximum buffer space needed for any on-line tape can be reduced by half by suppressing read-ahead via the FILE statement SPR parameter; for example:



Note, however, that reducing the number of buffers for an I/O-bound program may significantly increase running time. It may be necessary to balance LCM requirements against I/O requirements. The SPR parameter is described further in section 6.

AUTOMATIC MEMORY MANAGEMENT

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

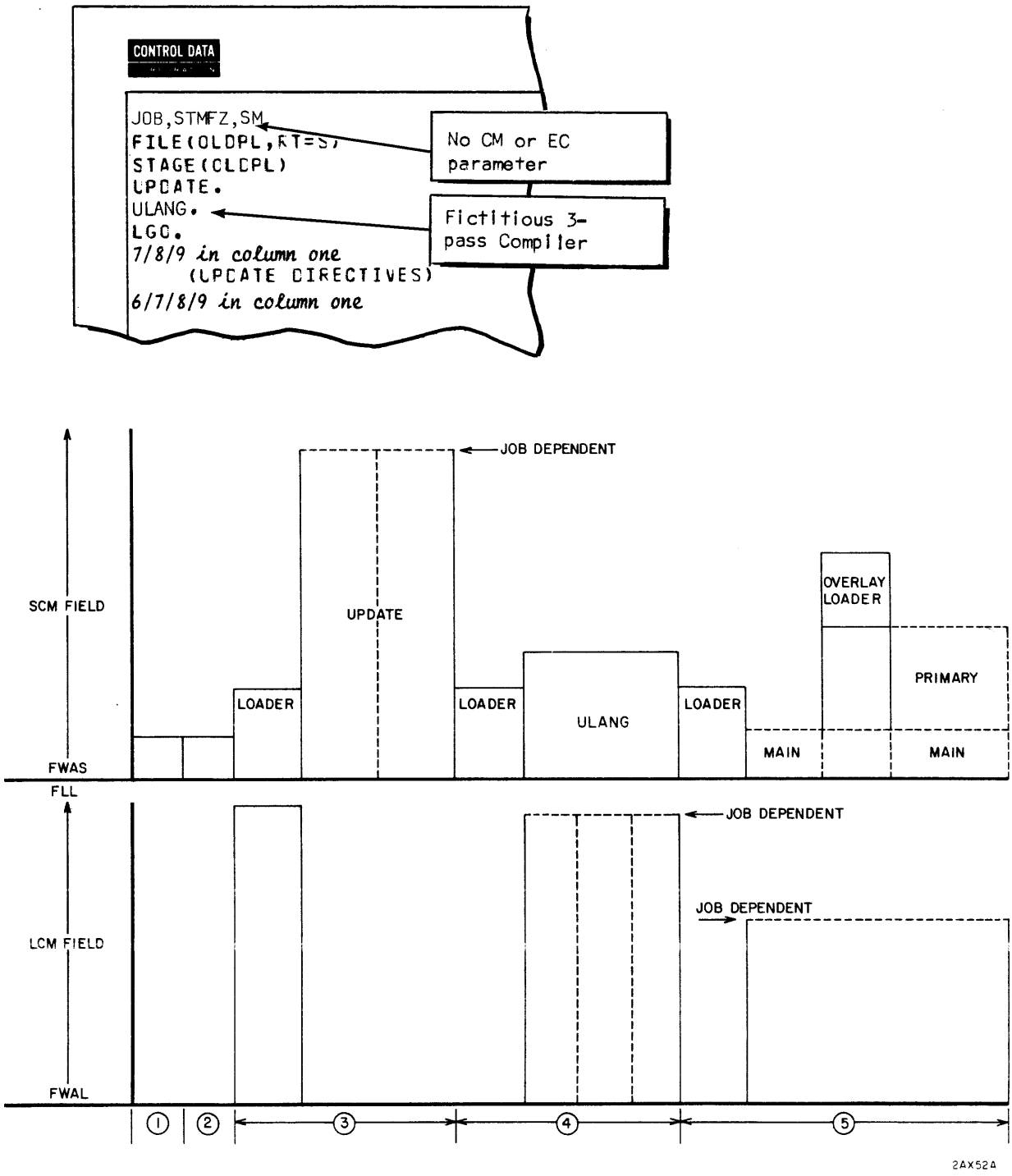
The most common, most efficient, and easiest way of managing SCM and LCM field lengths is by using the automatic mode. This mode (also called dynamic field assignment and system controlled mode) is the system default. It is initially in effect for small central memory if the CM parameter is omitted from the job identification statement and is in effect for large central memory if the EC parameter is omitted. Automatic mode is overridden for the applicable memory 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. 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. ULANG is a fictitious compiler that 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 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 memory assigned to the job is reevaluated.

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.



Example 4-1. Job Using Automatic 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 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 memory location that exceeds the highest address loaded with the 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 legally.

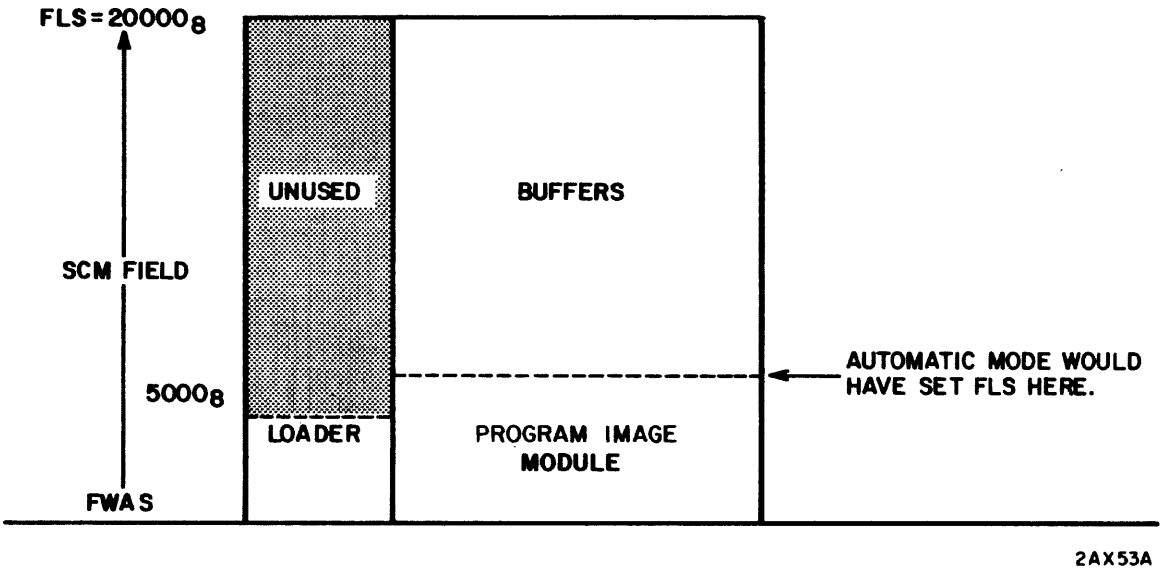
CONTROLLING SCM

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 to be 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 program image module to speed up loading. This technique is described in the COMPASS Reference Manual.

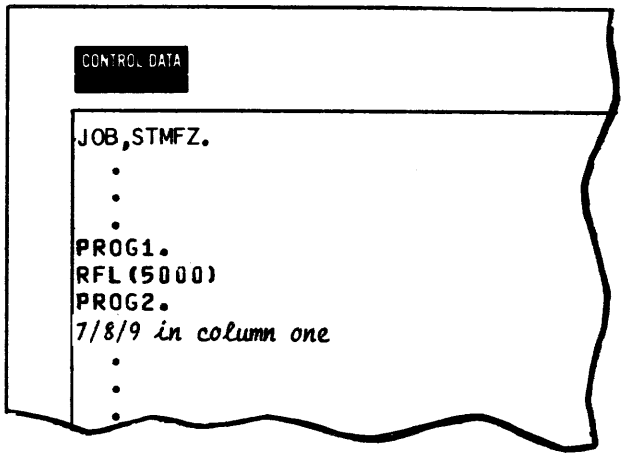
```
JOB,STMFZ,CM20000.
ATTACH(PROGA,PERMFILE,ID=XX)
PROGA.
6/7/8/9 in column one
```



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.



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 statement.

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,STMFZ,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)
```

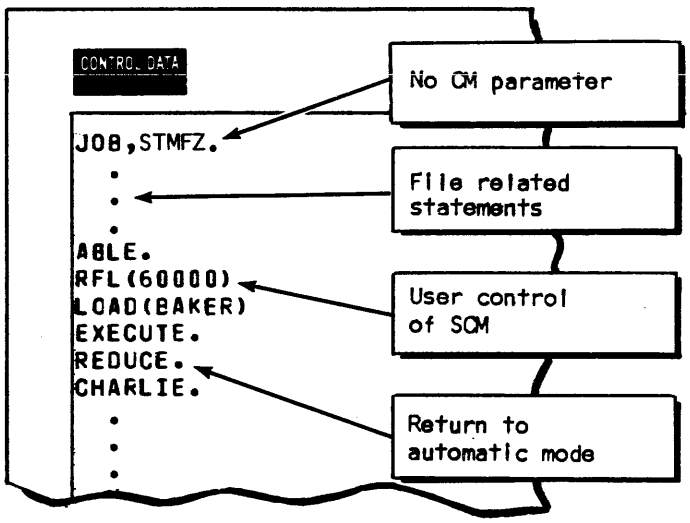
The preceding statement sets LCM to a fixed value of 16000_8.

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.

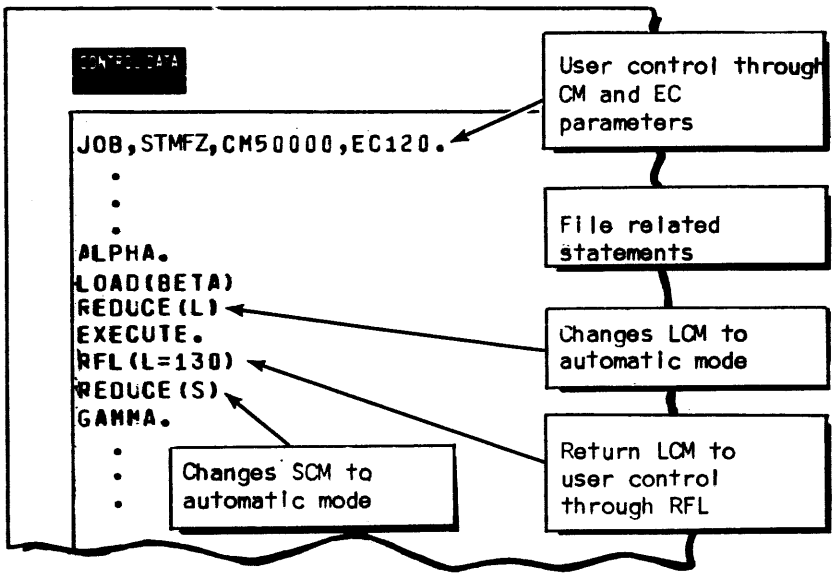


Example 4-4. Mixed Mode Control of SCM

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.



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

PRESETTING 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 or of not setting the fields. For this discussion, let us assume that the installation default specifies no presetting of memory.

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 memory 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

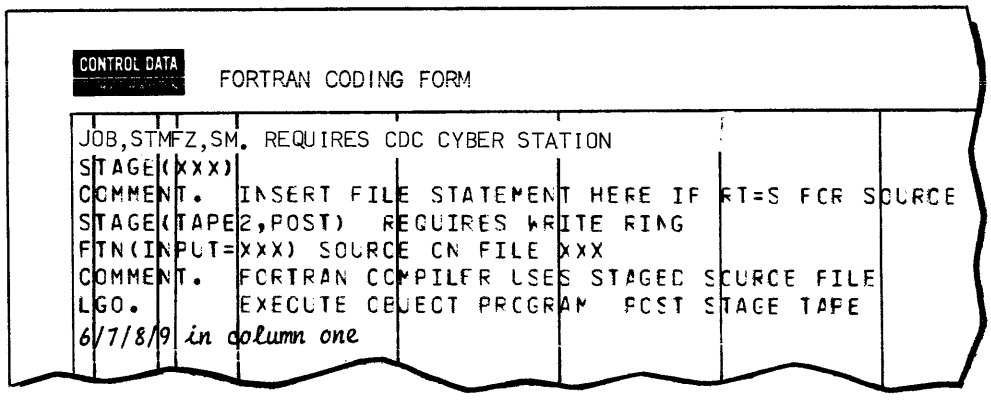
For SCM, addr is a maximum of 18 bits. For LCM, it is a maximum of 21 bits.

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

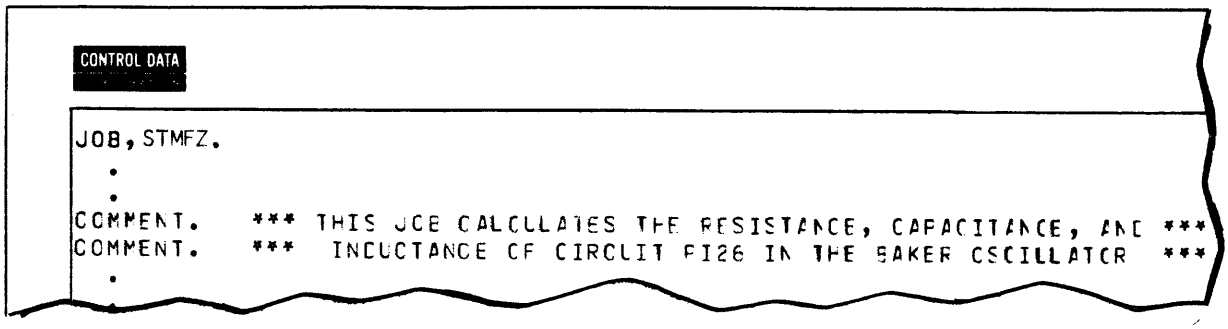
Comments help provide a history of a job. Insert special comments or remarks after the terminator on any control statement. Such remarks are useful in interpreting program listings, or in providing general information. Example 4-6 illustrates some control statements that include comments.



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 (end-of-line). Blanks, periods, and other punctuation are allowed.

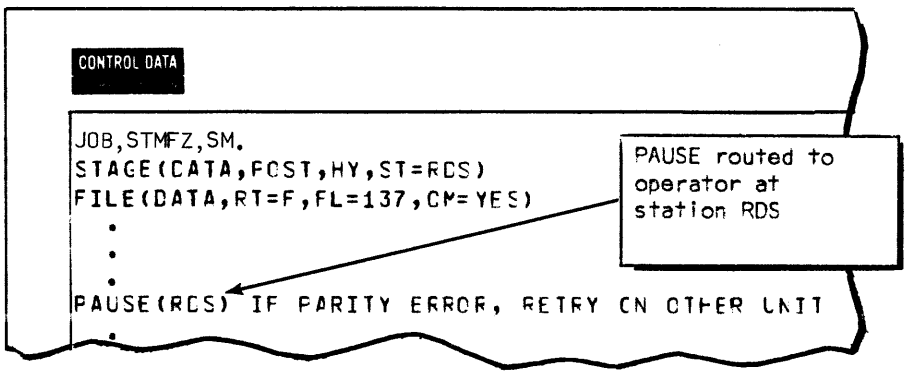
Remarks following COMMENT. are printed in the dayfile and the first 50 characters of the statement including COMMENT. are displayed to the operator on the console screen at the originating station. If you have a message to the operator, however, use PAUSE rather than COMMENT. because comments from the 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 statement (Example 4-7).



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.



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

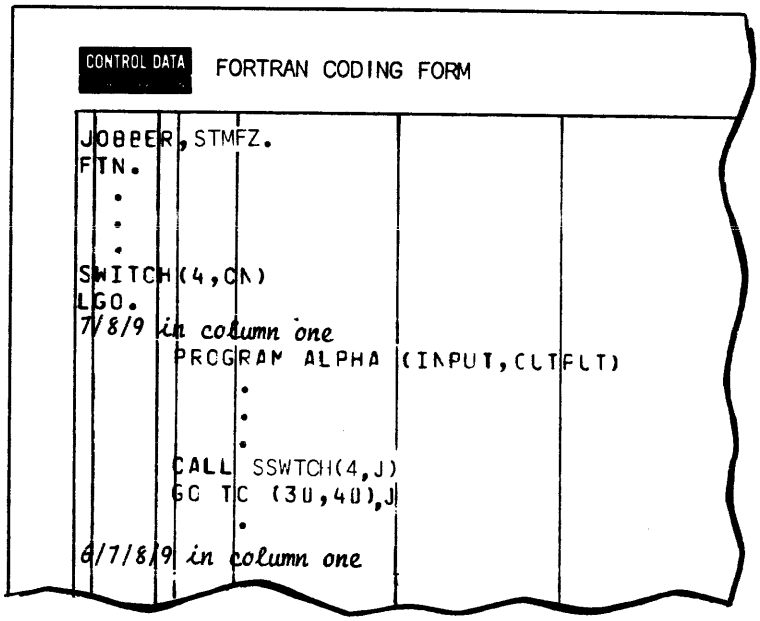
SETTING PROGRAM SWITCHES

SCOPE maintains for the user six control 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, ON / OFF)

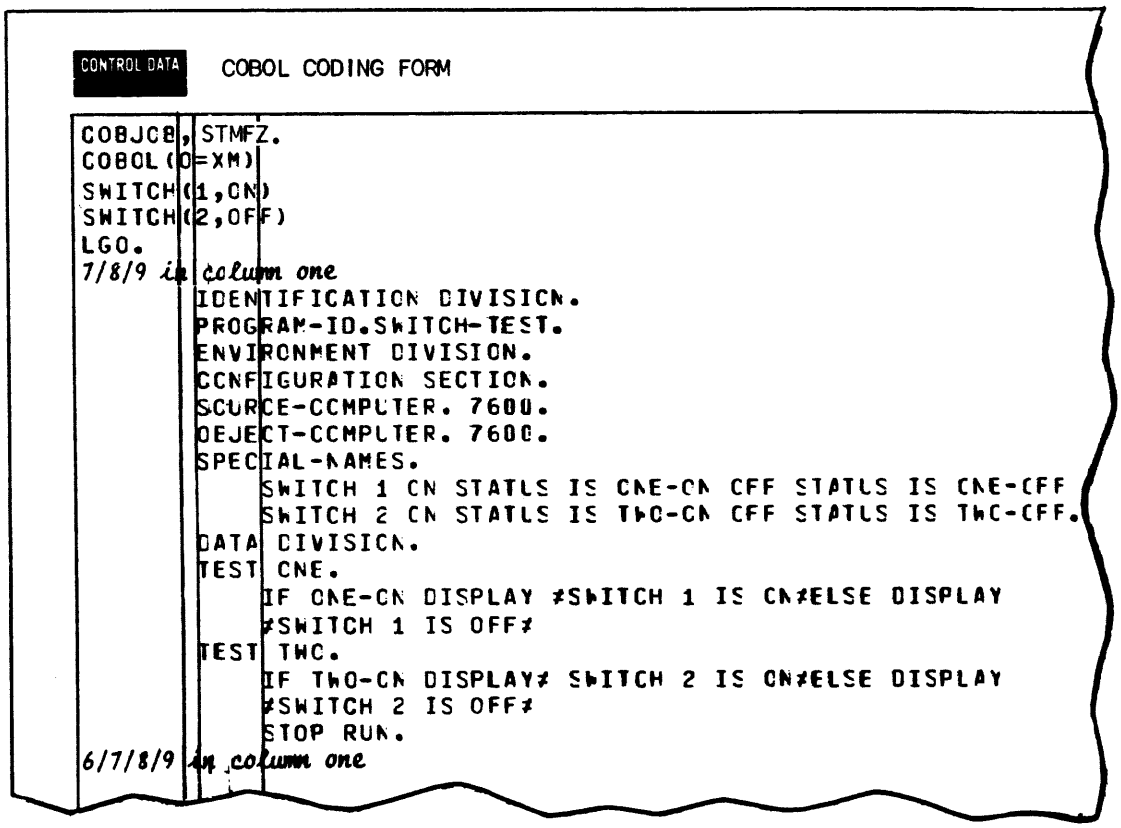
Switches are numbered 1 through 6. All of the switches are initially off. To turn a switch on or off, specify the switch number followed by ON or OFF, respectively. If the setting (ON or OFF) is omitted, the current switch position is reversed (toggled). That is, if it is off it is turned on and vice versa.

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



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

```
COBJOB,STMFZ.
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 together, and can delay 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 is the same for all jobs in a string. That is, it:

- 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. Note that if the job name is fewer than five characters, zeros are used for the missing characters.
- 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.

$$\text{TRANSF}(\text{job}_1, \text{job}_2, \ldots, \text{job}_n)$$

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.

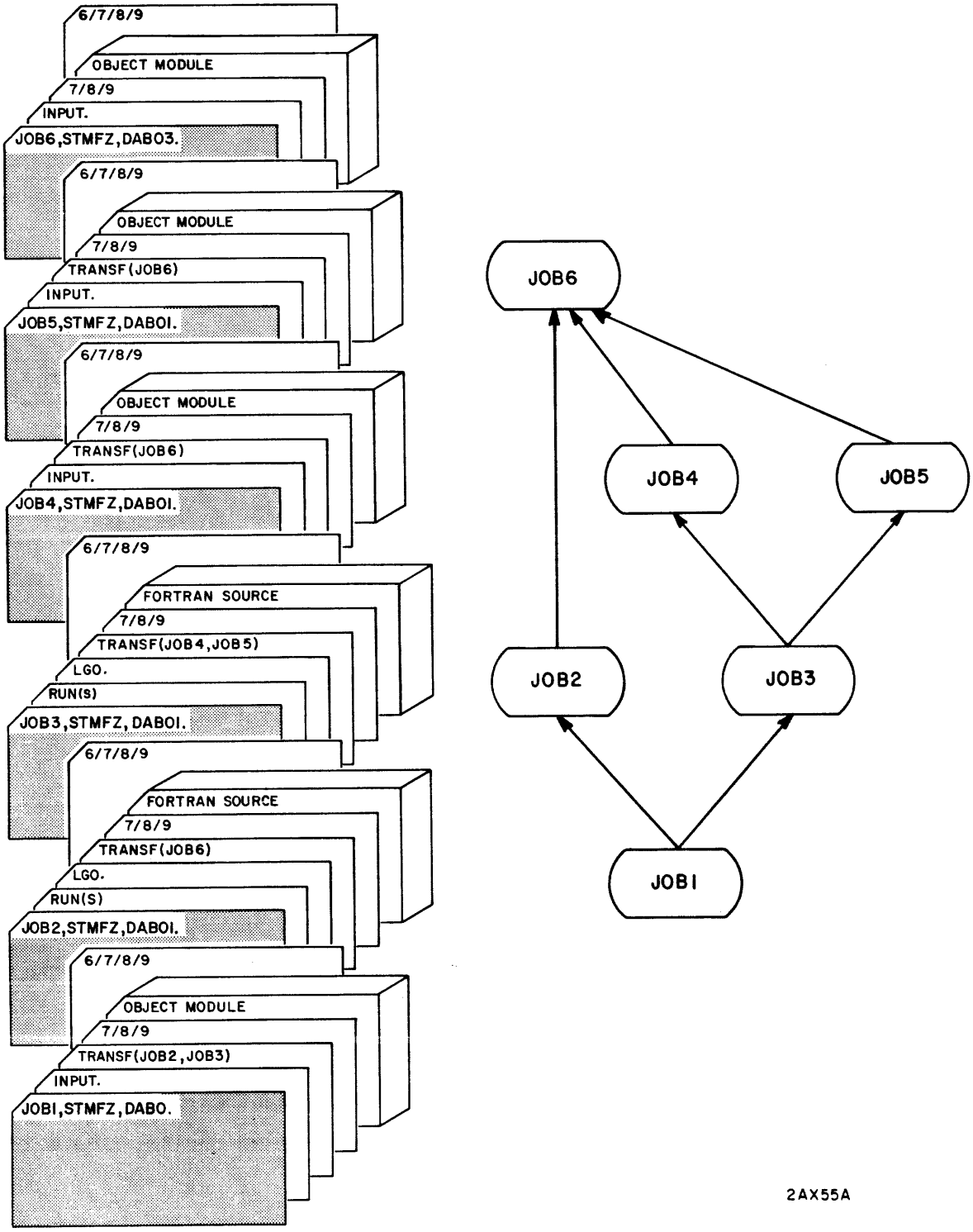
Parameters of the TRANSF statements consist of the names of jobs in the dependency string for which the job meets some need. Only the first five characters of the job name are relevant.

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 posting so that when the required job is submitted, the waiting job 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), and so on. 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 recovery of the operating system, all currently executing jobs (except those that cannot be rerun due to the conditions noted in the following text) are automatically rerun, although most executing jobs can be recovered from the point of interruption. 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 conditions.

- The job has attached a SCOPE 2 permanent file with extend or modify permission.
- The job has cataloged a permanent file under either SCOPE 2 or SCOPE 3.4.
- The job has executed a TRANSF.
- The job has purged a permanent file.
- The job has opened a connected file or has processed a CONNECT macro (refer to the SCOPE 2 Reference Manual).

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,STMFZ,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,STMFZ,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.

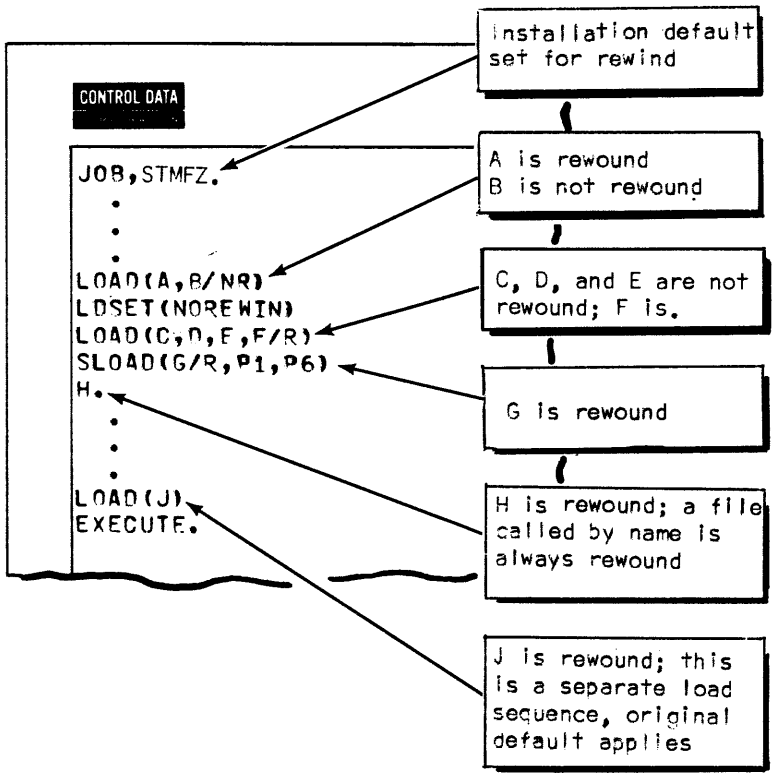
Rules for rewinding:

- Before loading from the file, the loader always rewinds a file name call (for example, the LGO file). INPUT is an exception. It is not rewound.
- 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.
- 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.

- 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.

- There is no need to rewind libraries.

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



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.

Logical file name

File organization (sequential, word addressable, or library)

Record type and specifications relevant to record type

Blocking type, if any

Processing direction (input, output, or input/output)

Labeling requirements (blocked files only)

End-of-data exit options

Error exit options

Other, optional information

Generally, the programmer provides the file name and can rely entirely on the compilers and assemblers to generate an FIT in the SCM field using system default values to fill in the file description. For example, the FORTRAN compilers generate an FIT for each file noted in the PROGRAM statement. The COBOL compilers generate an FIT for each file assigned, using the file statement (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 before the job step that requires the file specification. The first parameter must be the logical file name (lfn).

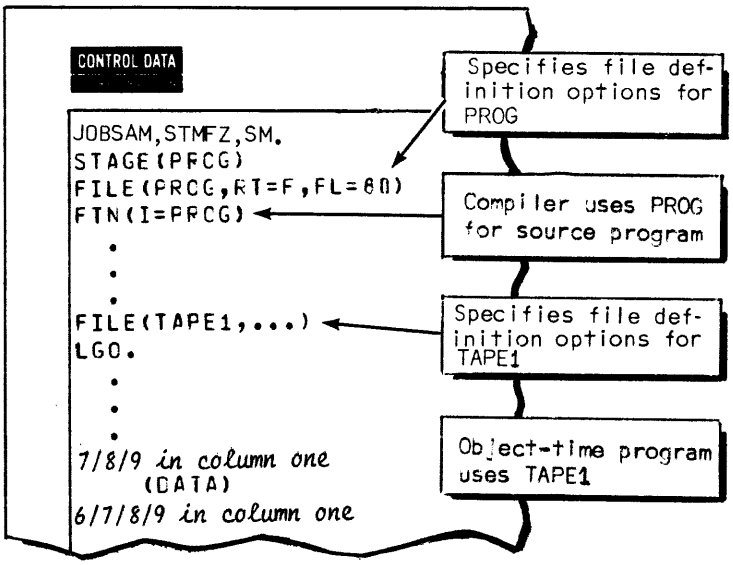
```
FILE(lfn,...)
```

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 refer 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 source input file PROG precedes the compilation; the FILE statement that redefines TAPE1 precedes the load-and-go statement.



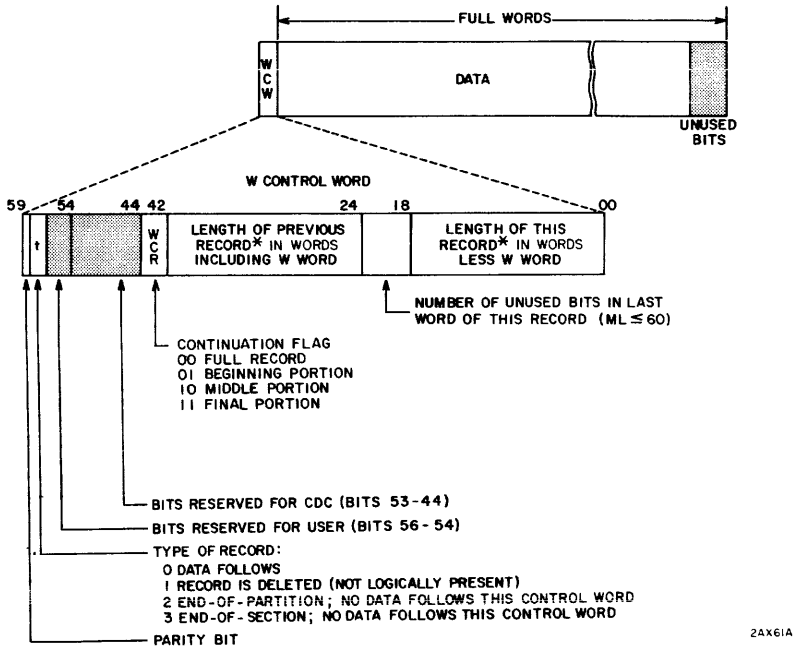
Example 5-1. Placement of FILE Statement

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

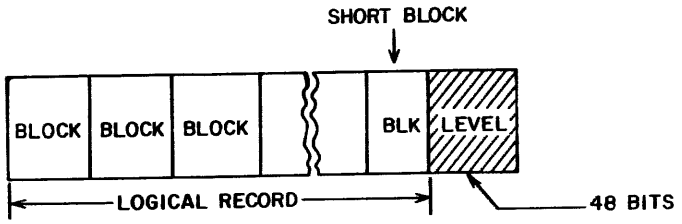
W Control Word: The first word of each record is a control word header containing sizes of current and previous records. This is the system and FORTRAN (RUN and FTN) default. To specify, place RT=W on the FILE statement.



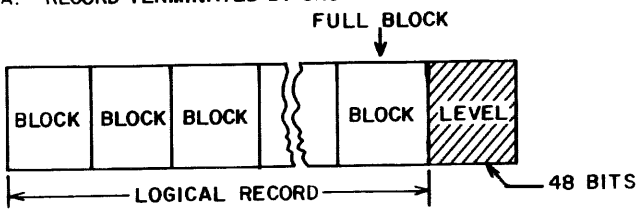
* OR PORTION OF RECORD, IF DIVIDED INTO PARTIAL RECORDS

A feature of W-records is that they may be constructed in portions. Each portion is prefaced with its own W-control word along with a flag (WCR) in the control word to indicate the portion is not a complete record but a partial record. The total record length (RL) then becomes the sum of the lengths of all the portions. The advantage is that W-type records can be constructed in portions without the RL being known at the time the first portion is constructed.

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. To specify, place RT=S on the FILE statement.

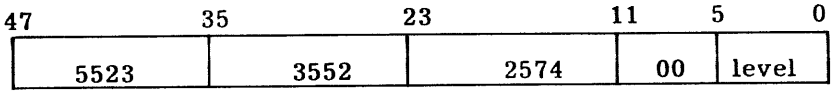


A. RECORD TERMINATED BY SHORT BLOCK AND LEVEL NUMBER



B. RECORD TERMINATED BY FULL BLOCK AND ZERO-LENGTH BLOCK CONTAINING LEVEL NUMBER.

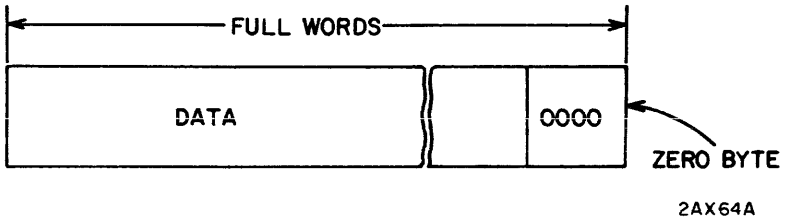
In binary mode, the 48-bit number contains the following information; a 4-bit level number is right-justified in the level field.



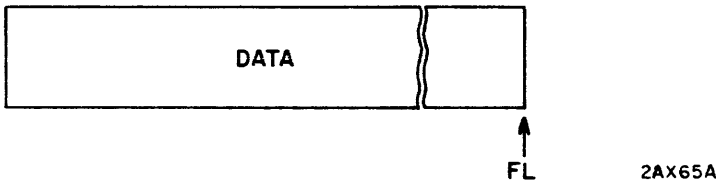
Level numbers in the 48-bit appendages indicate either end-of-record or end-of-partition as follows:

$0\text{-}16_8$	End-of-record
17_8	End-of-partition

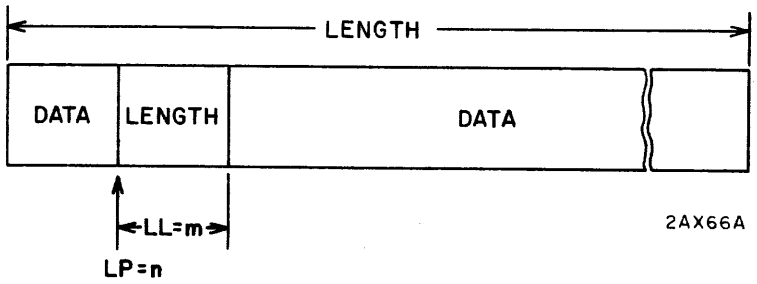
Z Zero Byte: Each record consists of an integral number of 60-bit words in which the last word has the low-order 12 bits set to zero, that is, contains a zero byte. The user must specify RT=Z and FL=n on the FILE statement, where n is a decimal count of characters. It must be large enough to accommodate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



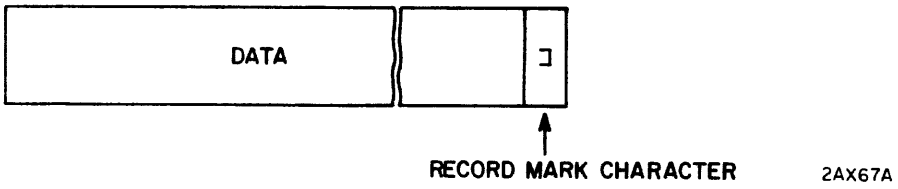
F Fixed Length: Each record consists of a fixed number of characters. This record type is specified by RT=F, FL=n on the FILE statement, where n is the decimal count of characters in each record. The default for n is 0.



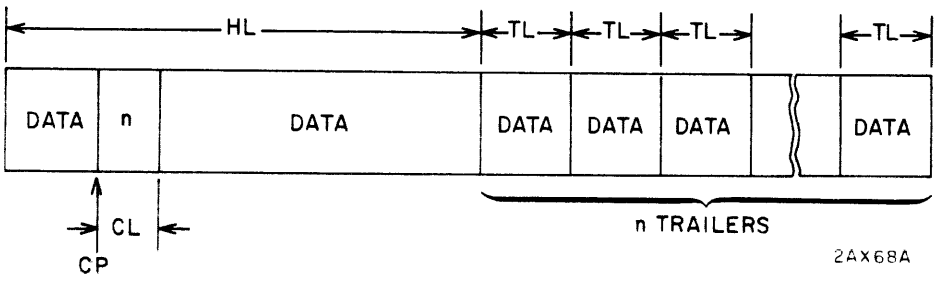
D Decimal Count: Each record consists of a number of characters specified in a decimal count field within the record. The record type is specified as RT=D. The position of the length field is specified in characters as LP=n; the length of the length field is specified as LL=m, where m is 1 to 6. The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



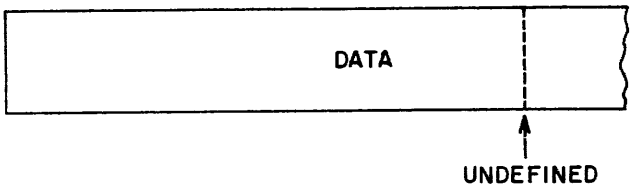
R Record Mark: Each record consists of a series of characters terminated by a character designated as the record mark character, conventionally]. The default for 7600 record manager is]. Use RMK=n, where n is the decimal or octal equivalent (octal equivalent suffixed by B) of the character in display code to specify a record mark character. 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 n 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.



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 SCOPE 2 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.

SCOPE 2 uses record type W as the default, regardless of the FORTRAN I/O statement used. When the file is assigned to magnetic tape, the file is blocked using I blocking (refer to Blocked File Format); otherwise, the file is unblocked. This usage is in contrast to FORTRAN for SCOPE 3.4 for which the default record type and block type is determined by the FORTRAN I/O statement used.

FORTRAN I/O Statement	SCOPE 2		SCOPE 3.4	
	Record Type	Blocking	Record Type	Blocking
READ/WRITE with a FORMAT statement	W	Unblocked unless file is staged or on-line tape	Z	C
READ/WRITE without a FORMAT statement	W		W	I†
BUFFER IN/BUFFER OUT statement	W		S	C

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.

In COBOL, because the record buffer area is defined in the source code, the user is prohibited from using a FILE statement attempting to specify MRL or FL larger than the source-language defined values. That is, MRL and FL can be used for shorter records but not longer records than are defined in the source language program.

† SCOPE 2 can read a tape created by SCOPE 3.4 I-blocked W record type only if it has been recorded in Stranger Tape format. Otherwise, the I/W file is embedded in SCOPE logical records; that is, the S record structure is superimposed over the I/W structure.

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.
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.

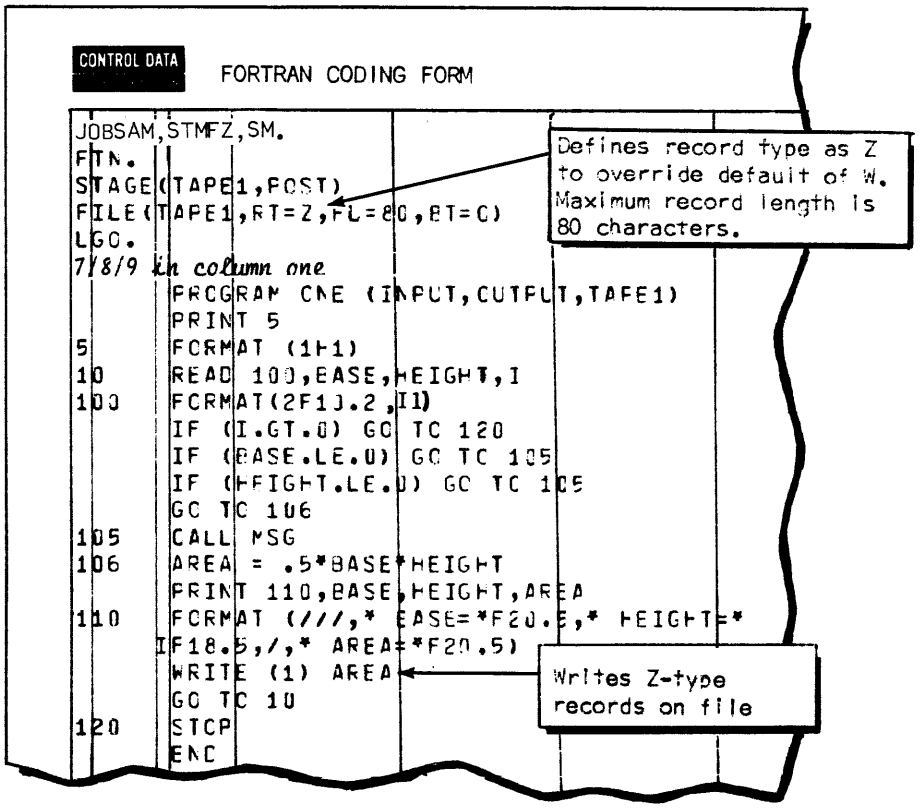
TABLE 5-2. COBOL SPECIFIED RECORD TYPES

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$_1$ TO integer$_2$ CHARACTERS DEPENDING ON data-name	D	D	illegal
RECORD CONTAINS integer$_1$ TO integer$_2$ 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.

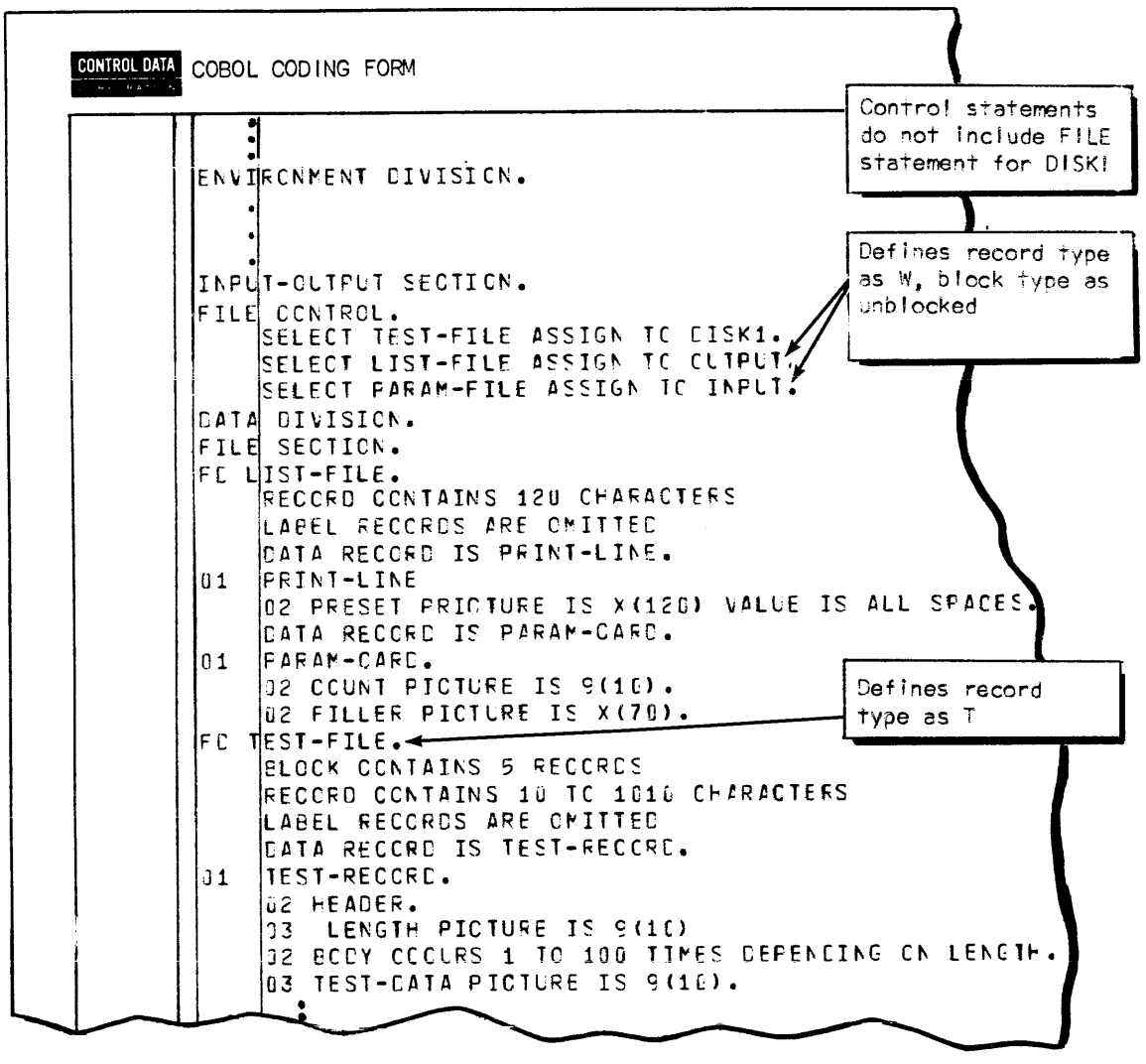
In Example 5-2, the FIT generated for TAPE1 by the FTN compiler defines record type as W. The STAGE statement, as will be described later, causes the file to be blocked. To change the description, the programmer inserts a FILE statement before execution causing record type to be Z with a record length 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.



Example 5-2. Overriding Default of W Record Type for FORTRAN Program

Example 5-3 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.



Example 5-3. 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. Records accessed by unformatted reads and writes have MRL set to 327,680 characters.

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 5120.

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

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

For block types K and E, MRL cannot exceed block size (MBL). (Refer to Blocked File Format in this section.)

UNBLOCKED FILE FORMAT

NOTE

The term "unblocked" used in the SCOPE 2 sense is essentially a gapless format supported only on 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 records cannot be unblocked.

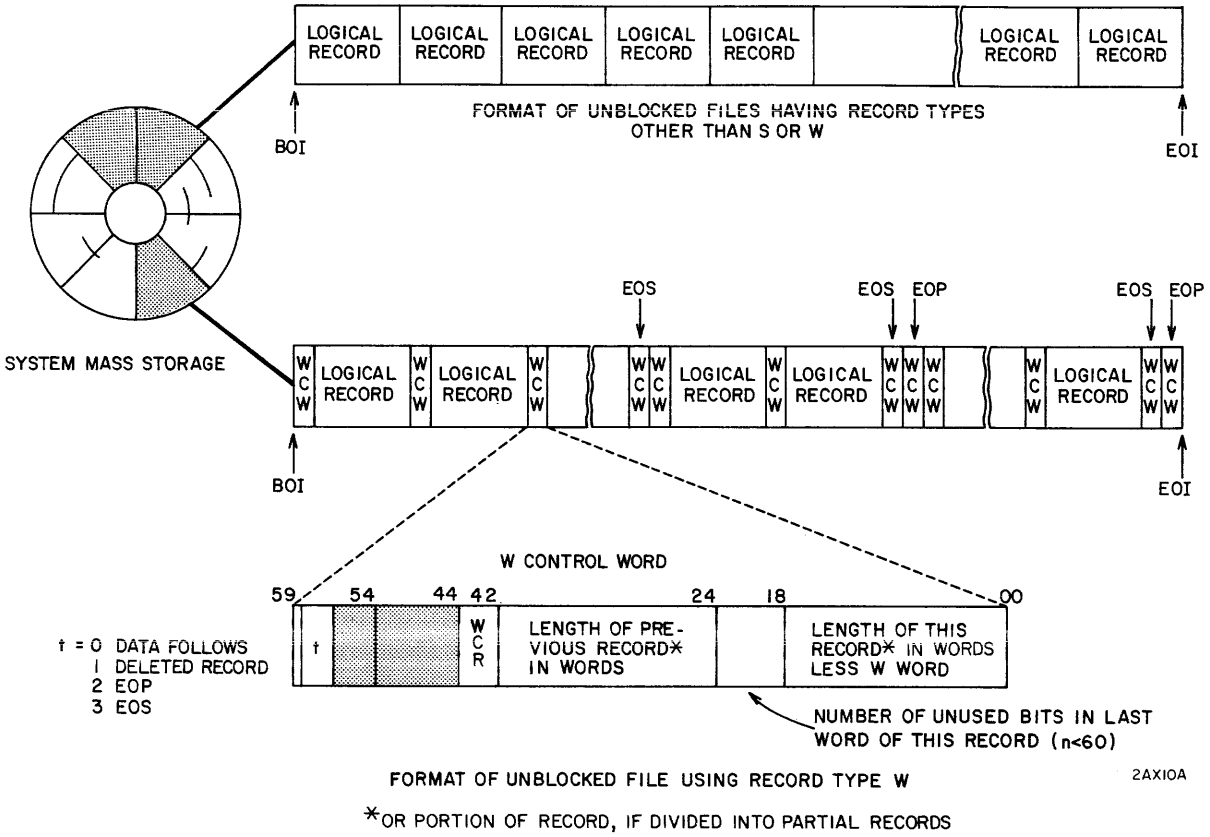


Figure 5-1. Unblocked File Format

RULES FOR ACCESSING UNBLOCKED FILES

- Unblocked files can be accessed using any of the file organizations. Remember, however, that SCOPE 3.4 does not allow sequential (SQ) files to be unblocked.
- Only unblocked files can be accessed as word addressable (WA) files.
- Only unblocked files using W record type can be accessed as library (LB) files.
- Only unblocked files can be loaded.

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

HOW TO SPECIFY UNBLOCKED

- 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).
- 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 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 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 memory use to a minimum. This technique is advisable in a heavily loaded multiprogramming environment.

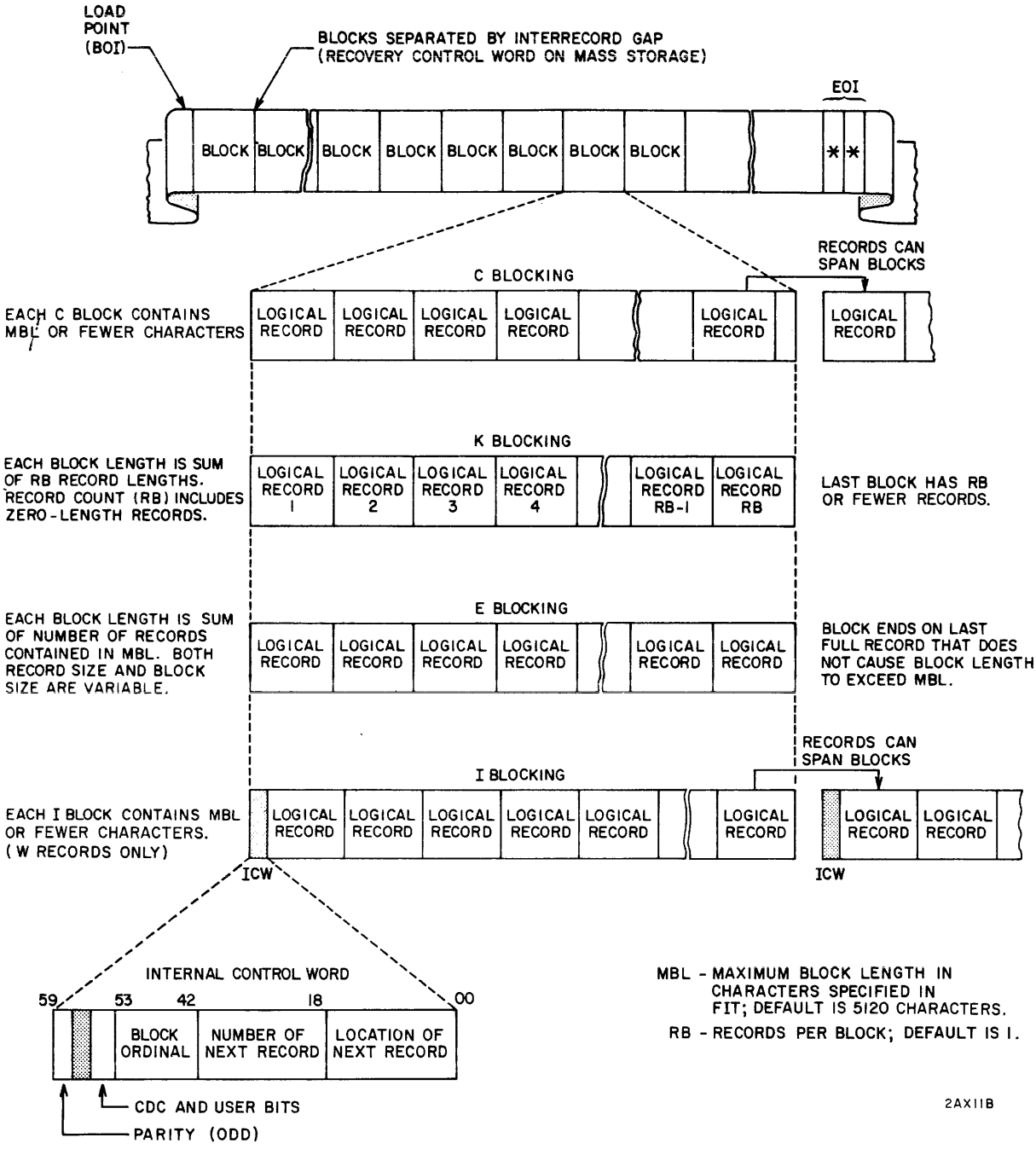


Figure 5-2. Blocking Types

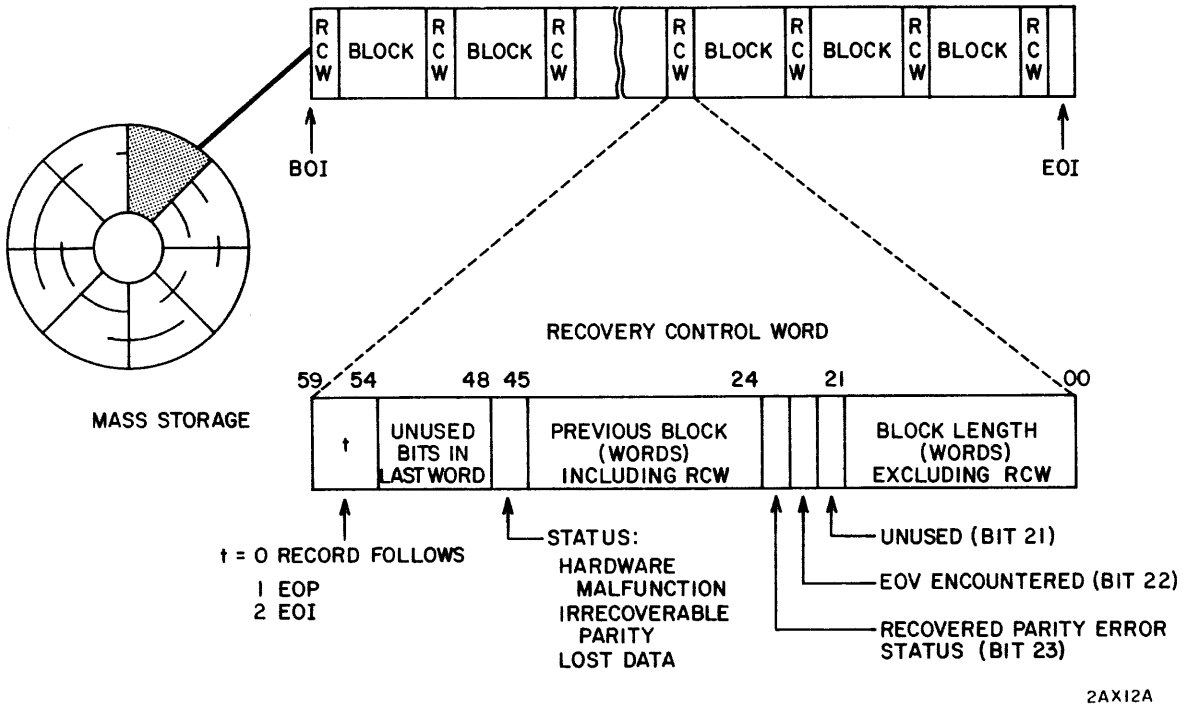


Figure 5-3. Blocked File Format on Mass Storage

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

- 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	C
other	K

- 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 the first rule under Rules for Specifying MBL.

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

- 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 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 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 records.

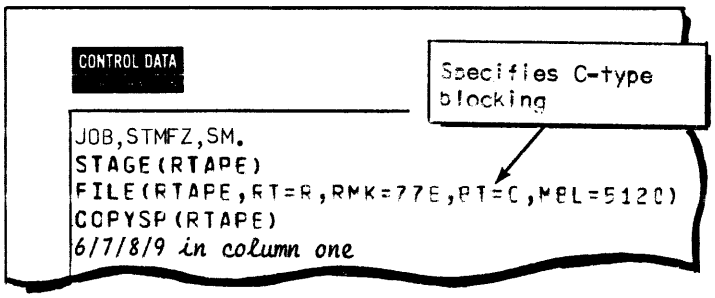
RULES FOR SPECIFYING MBL

- For I blocking, MBL must be 5120 to be compatible with SCOPE 3.4.
- For C blocking, MBL must be 5120 to be compatible with SCOPE 3.4 S/L devices.
- For S, Z, and W records, MBL must be a multiple of 10 characters (full words).
- MBL (for BT=K, MRL x RB) is limited by location of the magnetic tape unit as shown in Table 5-4. The CDC CYBER 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	
	6-Bit Characters	Words
On-line tape	262,140	26,214
Staged on-line tape	25,590	2559
CDC CYBER Station	25,590	2559
7611-1 I/O Station	5120	512

Example 5-4 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.



Example 5-4. 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:

- Blocked files with record types F, D, R, T, U, and W
- 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 6/7/8/9 card.

An end-of-partition is equivalent to the 7600 SCOPE 1 end-of-file card (6/7/8/9 multipunch in column 1) or a SCOPE 1 type 2 boundary control word.

The only unblocked record type for which partitions can occur is W record type.

The NOS/BE card reader can never read more than one end-of-partition card.

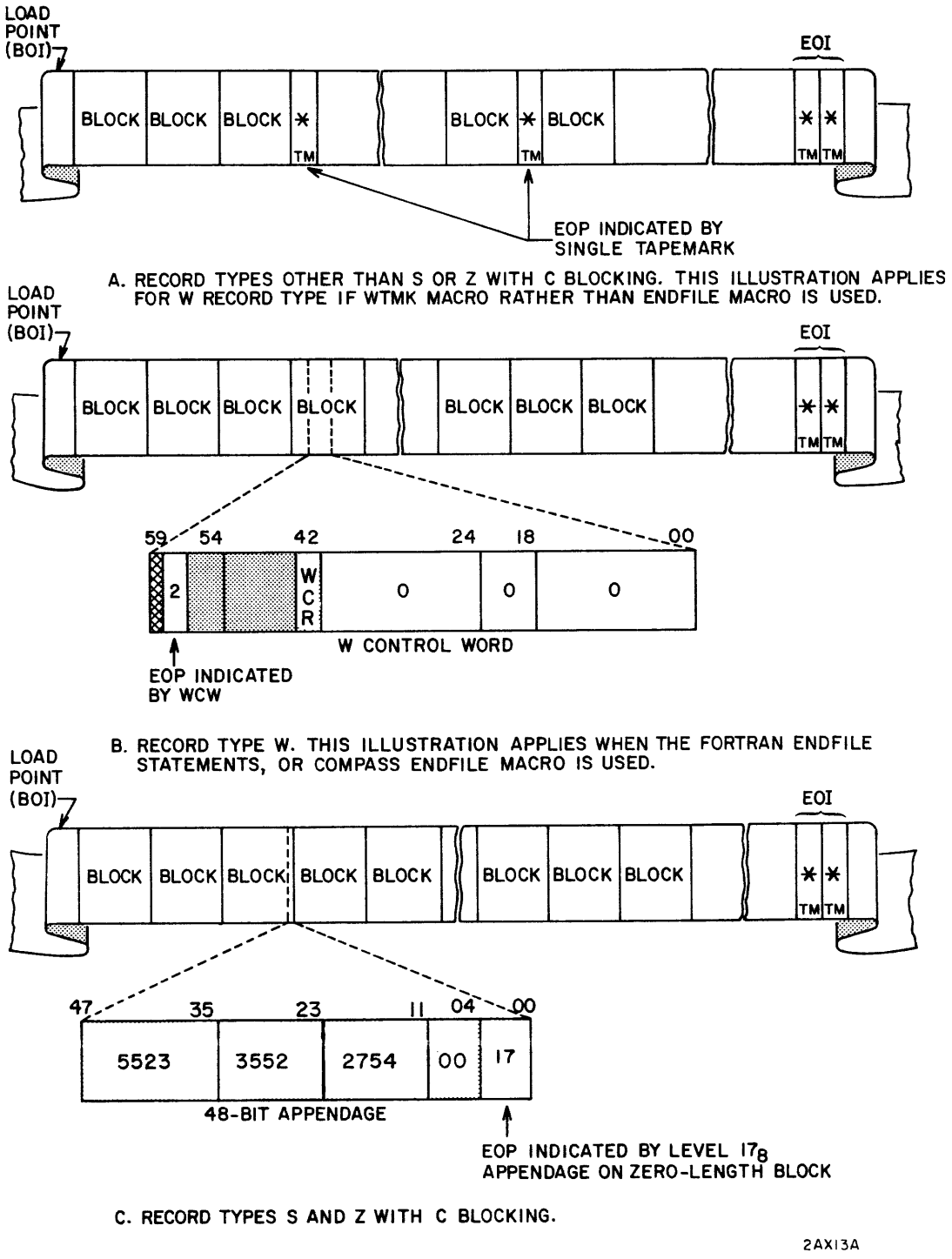


Figure 5-4. End-Of-Partition on Blocked Magnetic Tape

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 SCOPE 3.4 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 SCOPE 3.4. Example 5-5 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 SCOPE 3.4, but differs from SCOPE 3.3 and 7600 SCOPE 1 implementation, which always ignores the level 0 EOS.

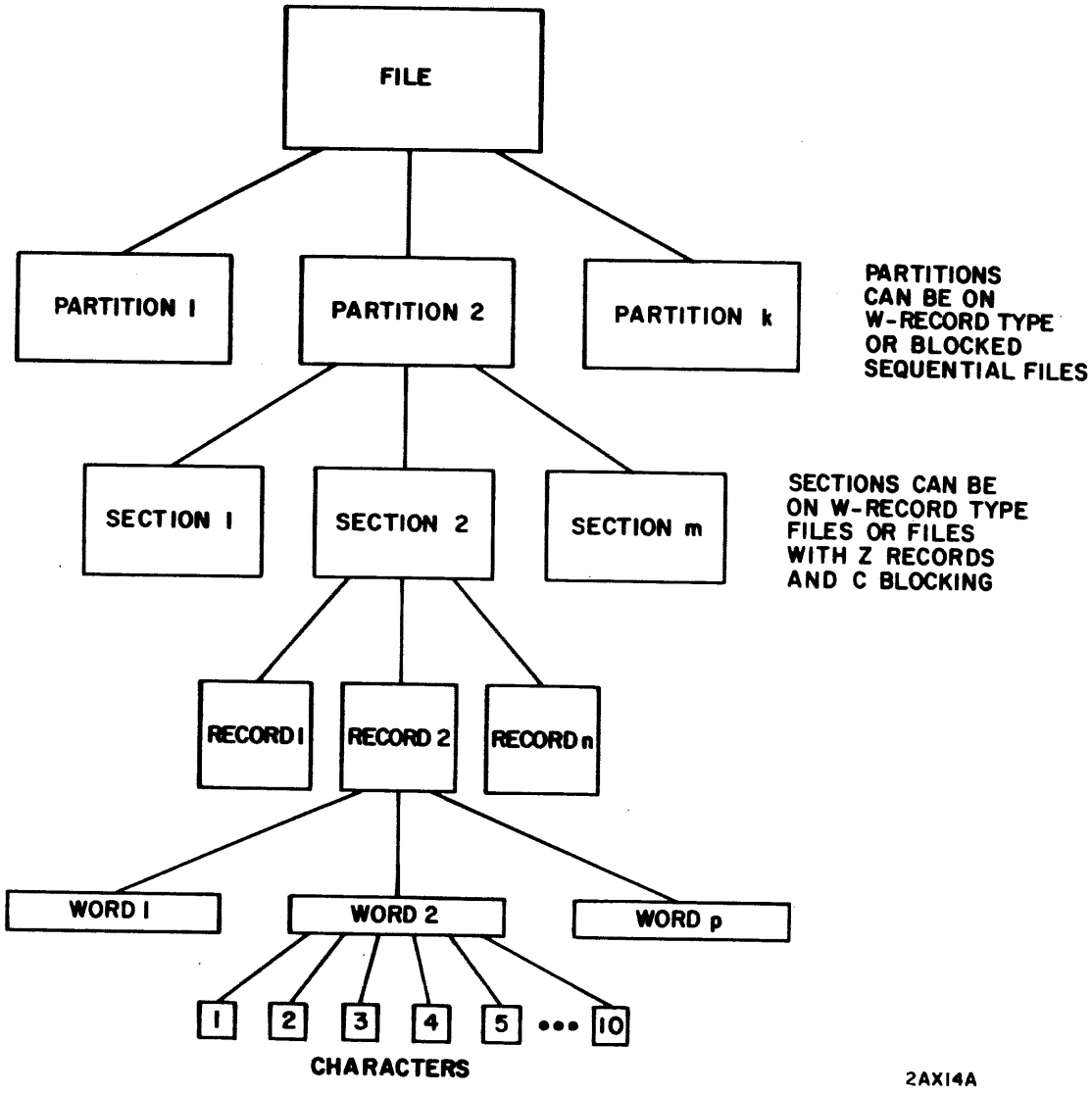


Figure 5-5. File Hierarchy

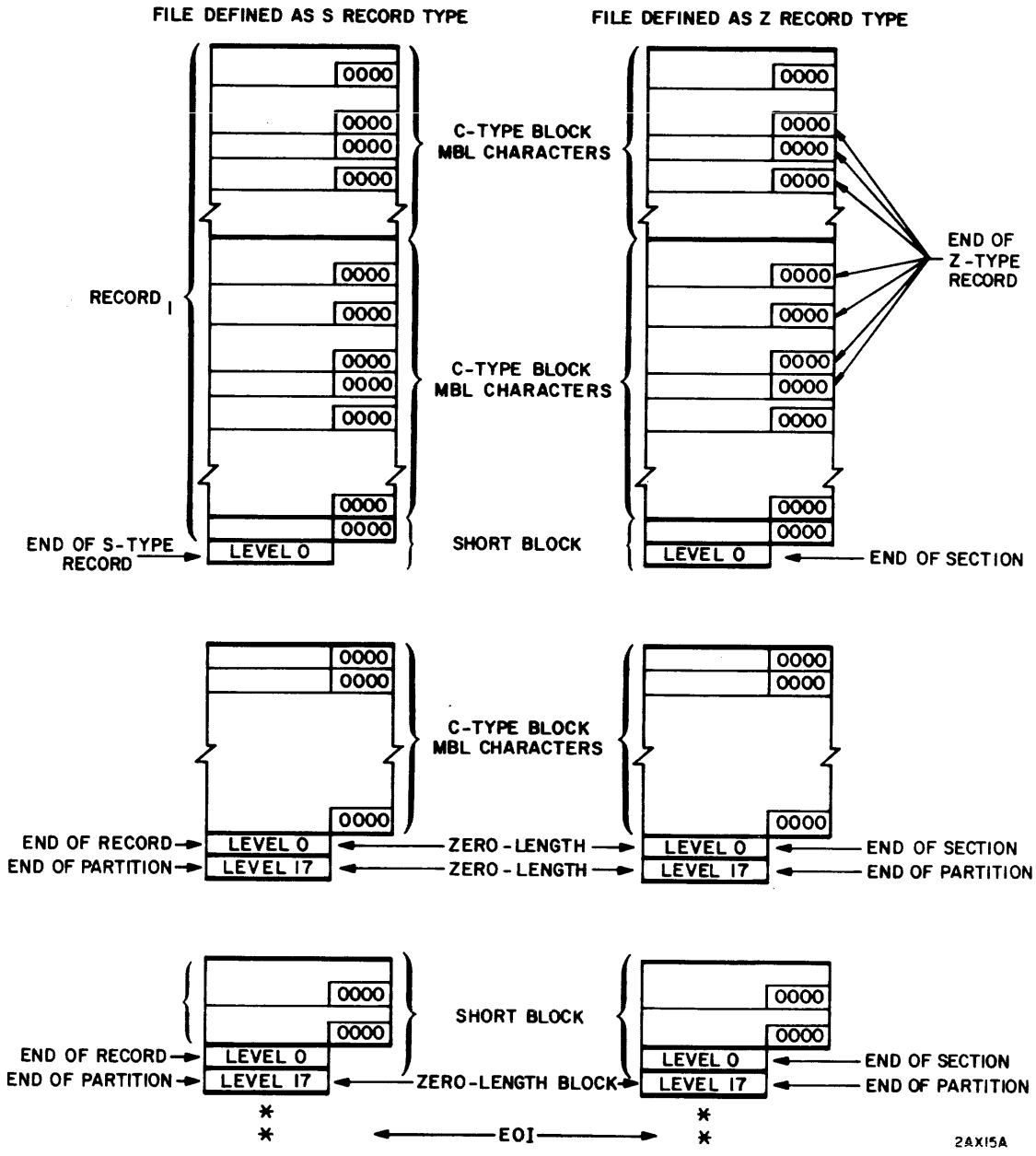
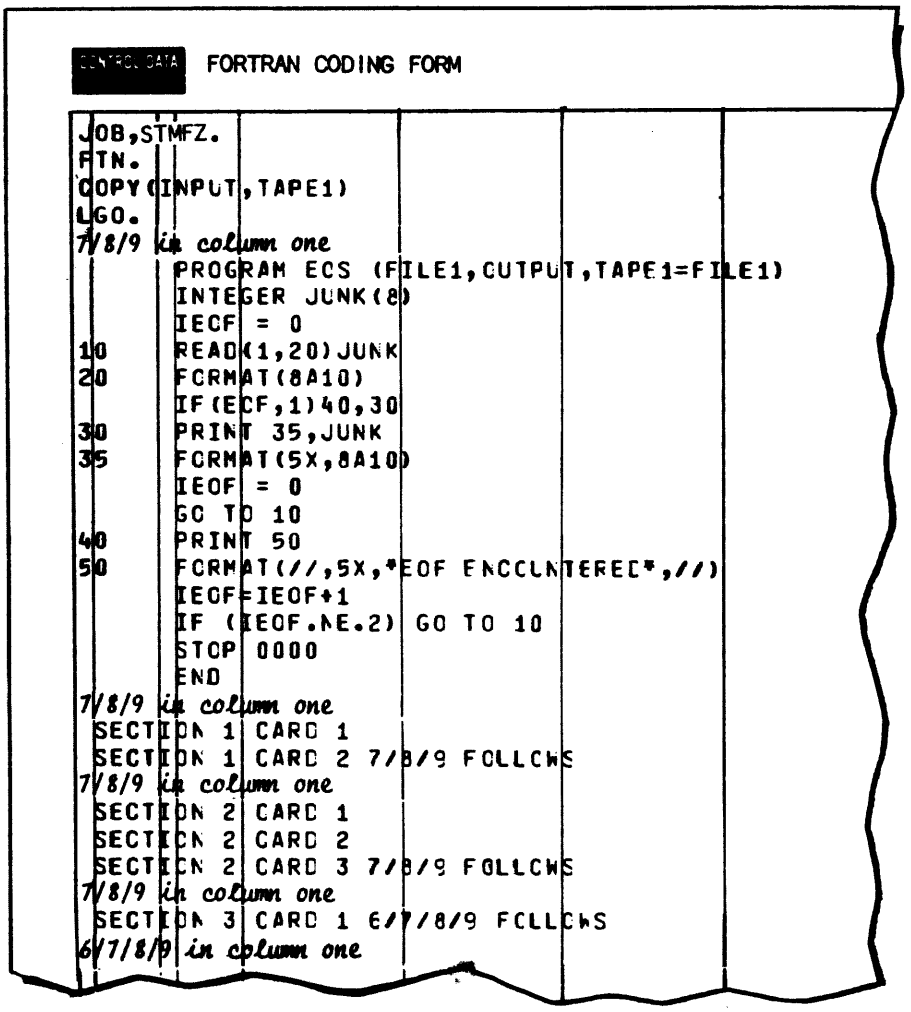


Figure 5-6. Relationship of S and Z Records (C Blocking)



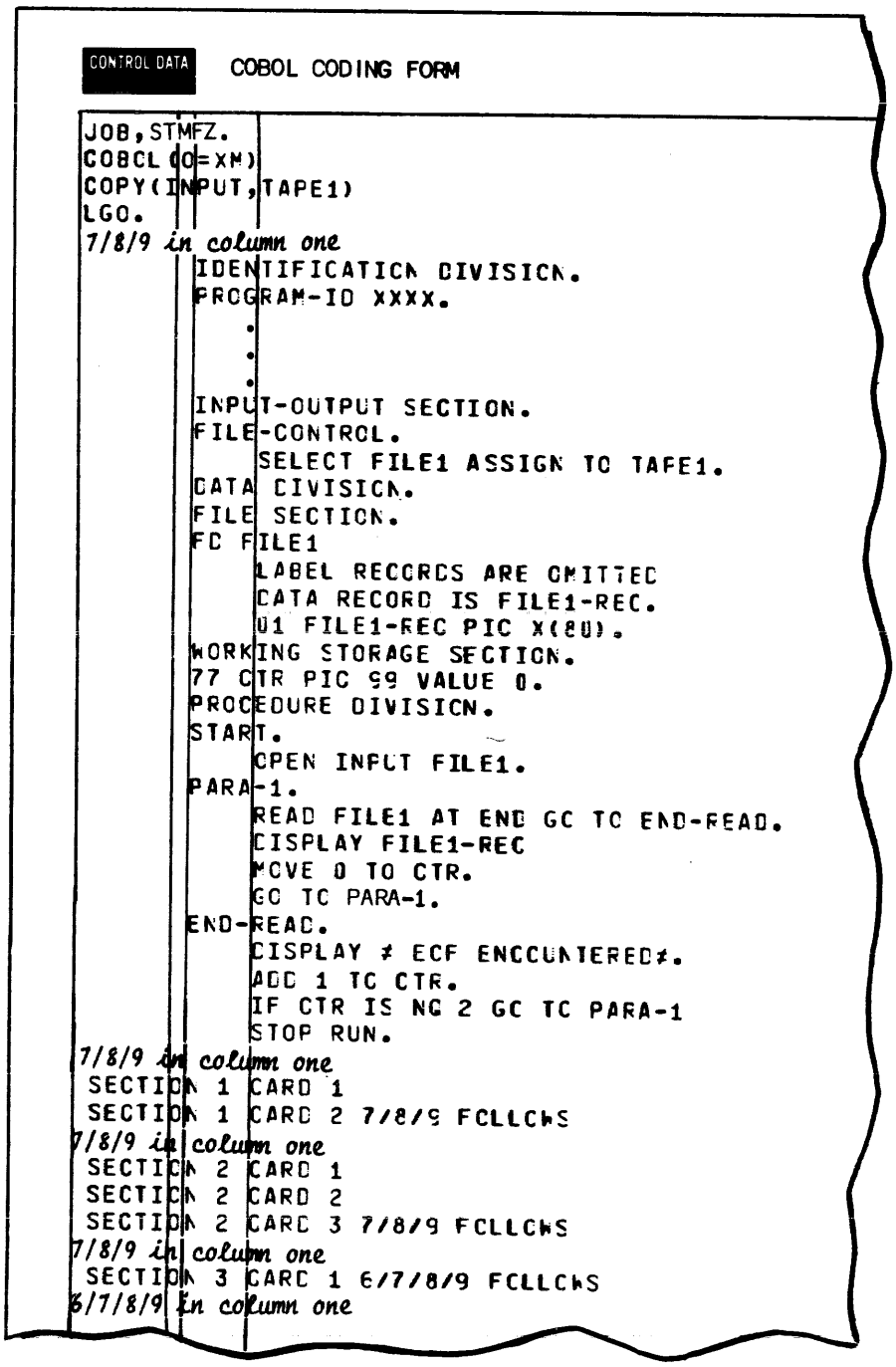
Example 5-5. FORTRAN Treatment of EOS on Input File

Example 5-6 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.



Example 5-6. 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 in 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 (refer to 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 time routines and the SCOPE 2 copy routines usually open a file as input/output. A prestaged file is opened for input; a poststaged 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 INPUT for input, OUTPUT for output, or I-O for input/output.

PROGRAM EXIT CONDITIONS

In general, following a call to the record manager, control returns to the user program at the next instruction after the call. Under certain conditions, control transfers to a location known as the exit address designated by the user.

Exit addresses can be specified only on record manager macros. Specification of an address implies the presence of a user owncode routine at the address.

Conditions causing exit addresses to be taken are:

- End-of-data conditions
- Label processing conditions (refer to Section 11)
- Error conditions

Prior to passing control to an exit address, the record manager stores a jump instruction at the exit address. This permits normal program execution to resume following execution of the user owncode routine. Control then passes to the exit address plus one.

END-OF-DATA EXIT

Control passes to a user end-of-data exit if one of the following conditions has been detected and the user has specified an end-of-data exit (DX=addr on FILE macro).

- A GET request encounters end-of-section, end-of-partition, or end-of-information.
- A READM request encounters an end-of-information.
- A SKIPBL or SKIPFL has encountered end-of-section, end-of-partition, beginning-of-information, or a tapemark.
- A SKIPBP has encountered beginning-of-information or a tapemark.
- A SKIPFP has encountered end-of-information or a tapemark.
- A SKIPBF has encountered beginning-of-information.
- A SKIPFF has encountered end-of-information.

When the end-of-data exit is taken (except for SKIPBx requests), the file is positioned immediately after the file delimiter encountered, that is, after the zero-length WCW, level number appendage, or tapemark. In the case of mass storage files at end-of-information, the file is positioned after the last word of data on the file. In the case of standard labeled tapes, the file is positioned after the first tapemark following the end-of-file trailer label group.

Where a record manager SKIPBx request encounters an end-of-data condition, the file position is immediately preceding the delimiter (except for BOI), that is, before the zero-length WCW, level number appendage, or tapemark. The next GET obtains data, however.

If no end-of-data exit is supplied by the user in the FIT and an end-of-data condition occurs, control returns to the user at the next instruction after the record manager macro. The file position flags are set in the FIT.

- End-of-logical record
- Beginning-of-information or beginning of volume
- End-of-section
- End-of-partition
- End-of-information or end-of-volume

ERROR EXIT

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), on whether or not an error exit address has been specified (EX field in the FIT), and on the type of error encountered.

By system default, the error option field is set for termination to occur upon encountering any record manager error. The FORTRAN compiler, however, generates an 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, use the EO=x parameter on the FILE statement, where x can be T, D or A or TD, DD or AD as described in the following text.

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

TERMINATION ON ANY ERROR (EO=T)

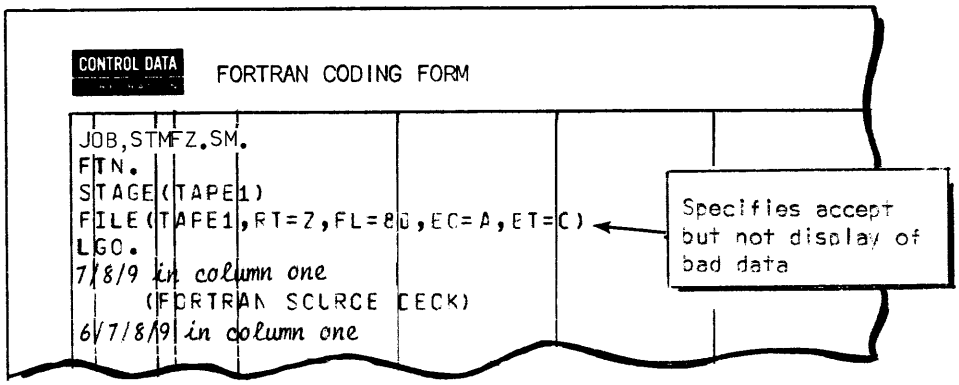
If the error option parameter in the FIT specifies terminate (EO=T), no error exit is taken even if the user has supplied one. The job is terminated if any record manager error is encountered.

If EX has been specified and the EO parameter specifies other than T or TD, control passes to the error exit. For unrecovered tape read parity errors, the action taken by the record manager before passing control to the user exit depends on whether accept (A) or drop (D) has been specified on the EO parameter. For all other errors, the action is the same for both A and D, that is, the error status is indicated in the FIT and control passes to the user.

ACCEPT ERROR (EO=A)

For an unrecovered tape read parity error for which EO=A or EO=AD has been specified, the record manager places the bad data in the user buffer and passes control to the user. The file is not repositioned. The next read causes the record manager to move the next record, possibly the same block, to the user buffer.

In Example 5-7, EO=A is specified on the FILE statement to override the FORTRAN-specified accept-and-display.



Example 5-7. Specifying Error Option as Accept With No Display

NOTE

A parity error indication occurs on the first, and only the first, record in bad block, whether the record is good or not.

DROP BAD DATA (EO=D)

For an unrecovered tape parity error for which drop has been specified (EO=D or EO=DD), the record manager attempts to skip the bad data and resume file processing with the next record 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.

When no error exit address is provided and the EO parameter is not set for job termination, the action taken by the record manager is the same for A and D as described above except that control passes to the instruction following the macro request instead of to an exit address.

DISPLAY BAD DATA (EO=xD)

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 named ZZZZZEF which is automatically spooled 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.

Although labeled files are not described in detail until Section 11, it is helpful at this time to visualize the structures of both unlabeled and labeled files on reels of magnetic tape.

Figure 6-1 illustrates combinations for unlabeled tapes. Figure 6-2 illustrates combinations for labeled tapes. In the illustrations, each reel is a volume of magnetic tape; an asterisk represents a single tapemark.

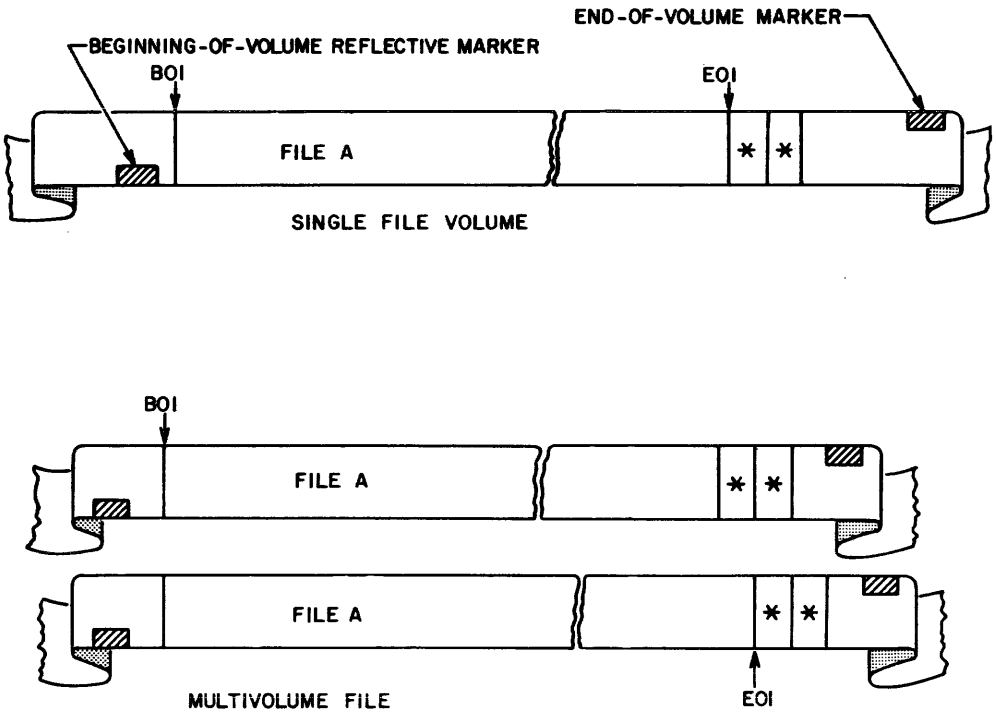


Figure 6-1. Unlabeled Magnetic Tape Files

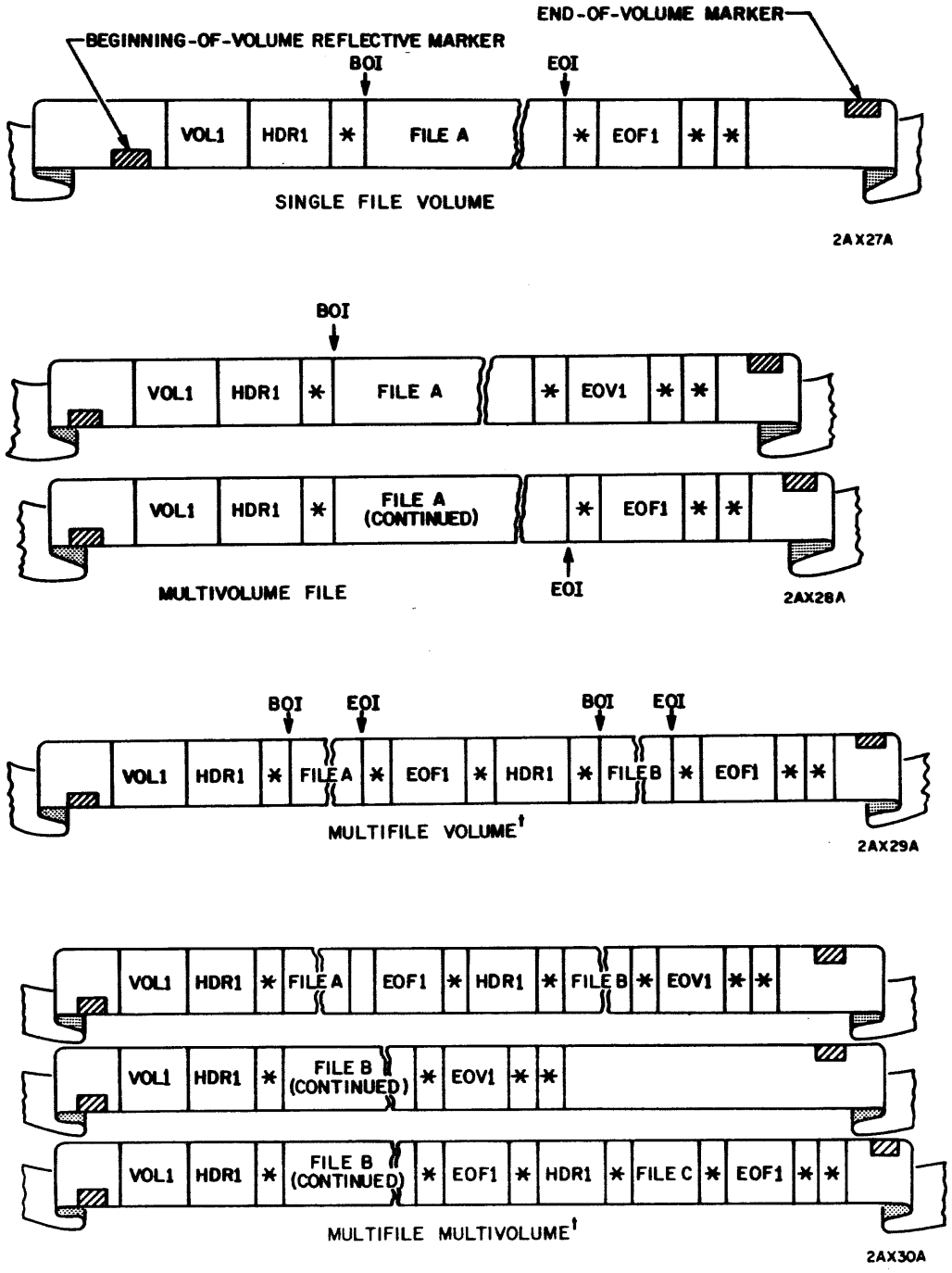


Figure 6-2. Labeled Magnetic Tape Files

† Standard labeled on-line tapes only

STAGING TAPES

Staging is the transfer of 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.

$\lceil \text{STAGE}(\text{lfn}, p_1, p_2, \ldots, p_n)$ and $\lceil \text{STAGE}(\text{lfn}, \text{POST}, p_1, p_2, \ldots, p_n)$

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
HD or PE	Used in place of NT to specify 9-track and indicate desired density
HY, HI, or LO	Used in place of MT or default to specify 7-track and indicate desired density
$\text{VSN}=\text{vsn}_1/\text{vsn}_2/\ldots/\text{vsn}_n$	Specifies volume serial numbers
ST=ggg	Specifies a 3-character physical or logical identifier for station

In addition, the A, and T, parameters described in section 7 can appear on the STAGE stagement to define mass storage characteristics such as maximum file size.

For prestaging, the lfn parameter is required as a minimum. For poststaging, 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. Prestaging occurs when a file is opened. Poststaging occurs when the job ends or when the file is returned or closed/unloaded. 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.

JOB STATEMENT PARAMETERS FOR TAPE STAGING

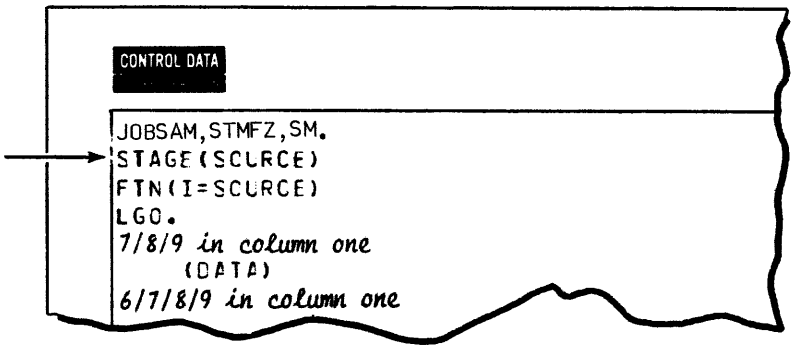
To indicate that a job may require staged tapes, add the following parameters on your job identification statement.

SM	Indicates that the job may require staged 7-track tapes
SN	Indicates that the job may require staged 9-track tapes

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 unlabeled reel (volume). The staging takes place when the compiler attempts to read from file SOURCE. At that time the job waits for the operator to mount the tape and for the tape transfer to take place. It resumes processing upon completion of the prestage operation.



Example 6-1. Prestaging an Unlabeled Tape

POSTSTAGING

NOTE

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

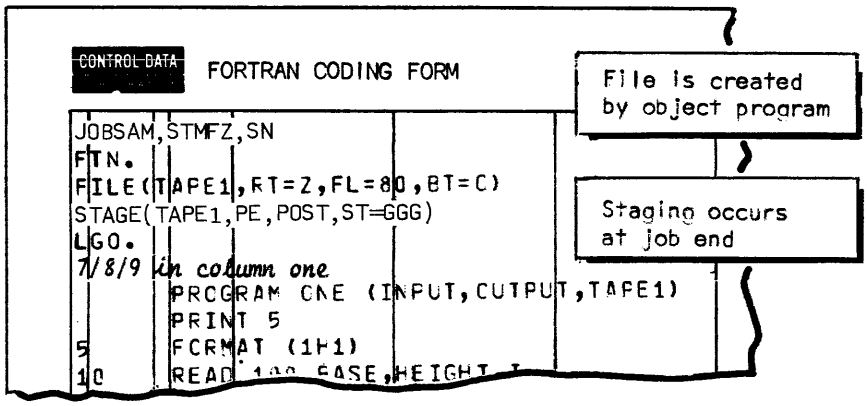
Poststaging is requested when POST appears on the STAGE statement. Upon job completion or return/unload of the file, the file is transferred to a station where it is written on one or more volumes of magnetic tape.

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 as a permanent file at a CDC CYBER station and poststaged.

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 poststaged. There is no way to poststage part of a file.

Example 6-2 illustrates poststaging 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 I.



Example 6-2. Poststaging an Unlabeled Tape

SPECIFYING TYPE OF TAPE UNIT AND DENSITY

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

NOTE

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 or a 9-track density (HD or PE) on a STAGE statement requests a 9-track unit. Otherwise, if the parameter is omitted, or if MT or a 7-track density is specified, a 7-track unit is used for staging. In Example 6-2, file TAPE1 is poststaged to a 9-track unit.

When no density is specified on the STAGE statement, the density is set by default. The default can be changed by the installation.

When reading 9-track tapes, the default is irrelevant because the hardware determines the tape density at load point and sets the density accordingly.

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

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	-
HD	-	800 bpi
PE	-	1600 bpi

The density for the label need not be the same as for the data for staged 7-track tapes. The station drivers search for a label at the beginning of a tape by trying to read the label record at each density in succession until they can read the label. The drivers then use the density specified on the STAGE statement 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=ggg on the STAGE statement. ggg identifies the station by either its logical or physical identifier.

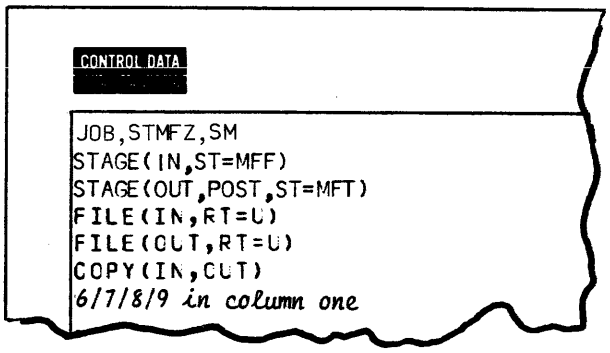
```
STAGE(lfn,...,ST=ggg)
```

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

If ggg is the physical or logical identifier of the CDC CYBER 76 mainframe on which the job is executing, staging occurs from an on-line tape unit.

† Not applicable to 667 units

In example 6-3, a CDC CYBER station has been logged in with the physical identifier MFF. Another CDC CYBER station has been logged in as MFT, a logical identifier specifying the station with magnetic tape files. The operator at station MFF is told to mount the input tape IN for prestaging and the operator at station MFT is told to mount the output tape OUT for poststaging. The prestaging operation begins when the file is opened by the copy. The poststaging operation occurs at job end.



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 whether it is binary or coded.

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 and is less dense on tape.

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. When the tape is read, 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 (section 10).

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

7-TRACK CONVERSION

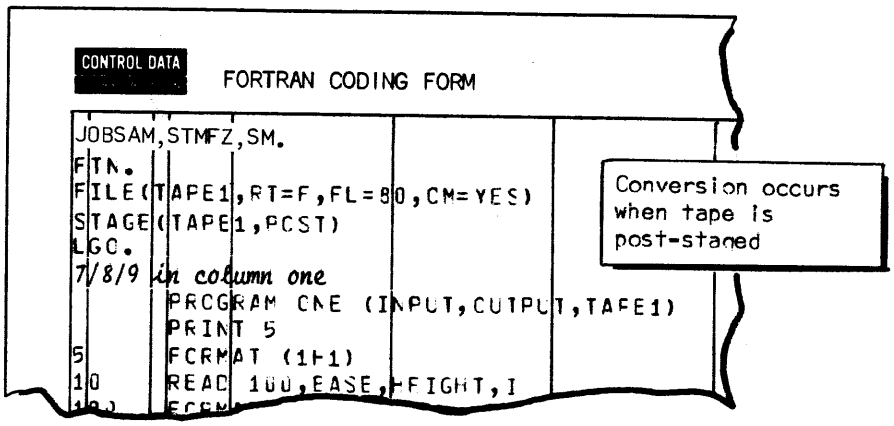
To select even parity, coded mode tape when using a 7-track magnetic tape unit, place the parameter CM=YES on the FILE statement.

```
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 code takes place.

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



Example 6-4. 7-Track Code Conversion for Poststaged 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 character using the system default character set installation 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,PE,US,...)
```

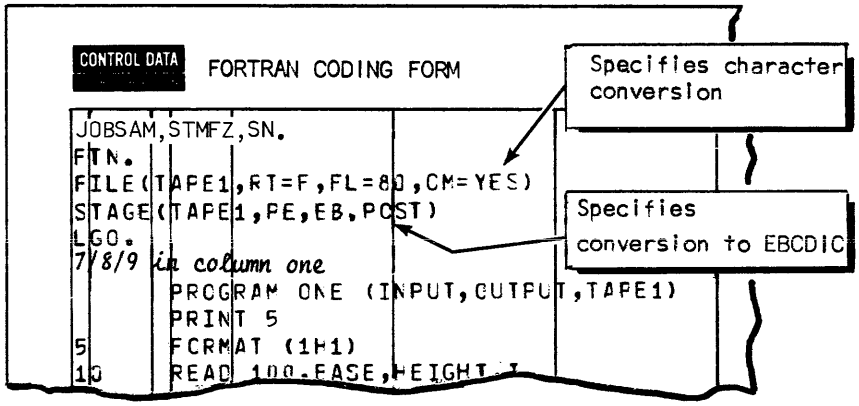
†ANSI Standard X3.4-1968 American Standard Code for Information Interchange
†† Extended Binary Coded Decimal Interchange Code

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,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.



Example 6-5. 9-Track Code Conversion for Poststaged 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:

- Stage-by-volume which permits an ordered progression of labeled reels (volumes) to be automatically prestaged upon completion of the previous volume.
- Stage-by-partial volume, which permits a selected portion of a single volume to be prestaged through station.
- 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 system 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 releases the storage for the current volume and requests staging of the next volume.

When a file is staged by volume, a rewind positions the file to the beginning of the current volume. Backspacing or skipping backwards is not possible 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 SCOPE 3.4 are terminated by the end-of-volume label. Automatic volume switching is possible when using tapes containing end-of-volume labels.

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=vsn1/vsn2/.../vsnn)UL.
```

vsn_i is a 1 to 6 alphanumeric character identifier to aid the operator in locating the correct tape. UL is used to specify unlabeled tape if the installation chooses to make labeled tape the default value.

LABELED TAPES

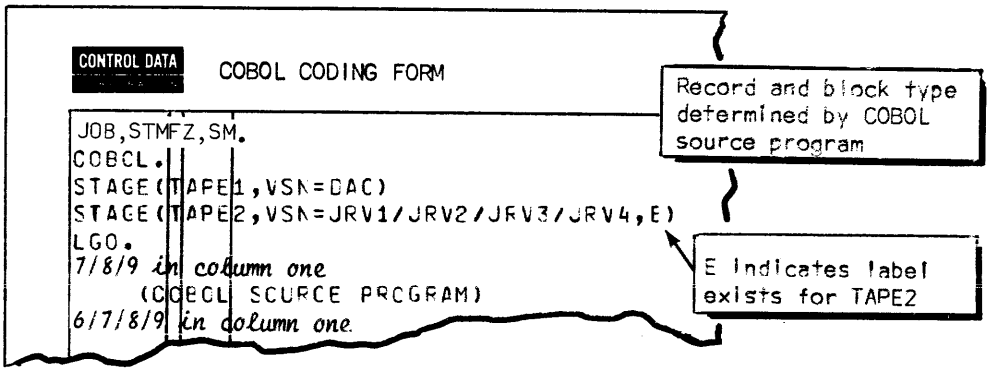
For a labeled tape volume, the vsn refers to both the external sticker of a reel and the contents of a field in the VOL1 label (appendix C). When no VSN parameter is supplied on the STAGE statement, the contents of the vsn field in the VOL1 label for the

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 (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.



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 generally increases throughput by decreasing mass storage requirements and staging time.

Only station staged tapes can be partially staged and only portions of one volume can be staged. Refer to the SCOPE 2 Reference Manual for a table illustrating the restrictions for partial staging.

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

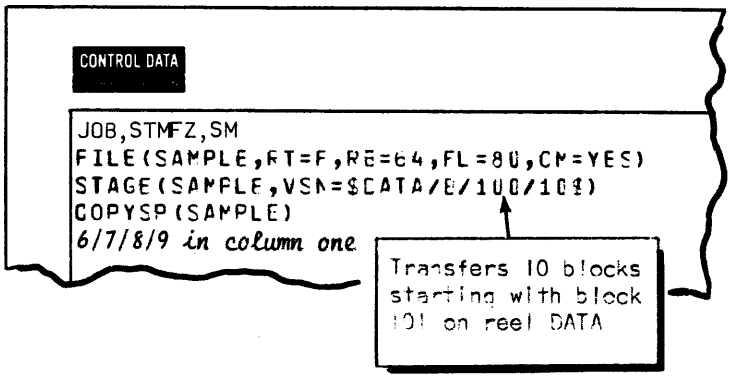
PRESTAGING BY BLOCKS

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.



Example 6-7. Partial Staging by Blocks

PRESTAGING BY TAPEMARKS

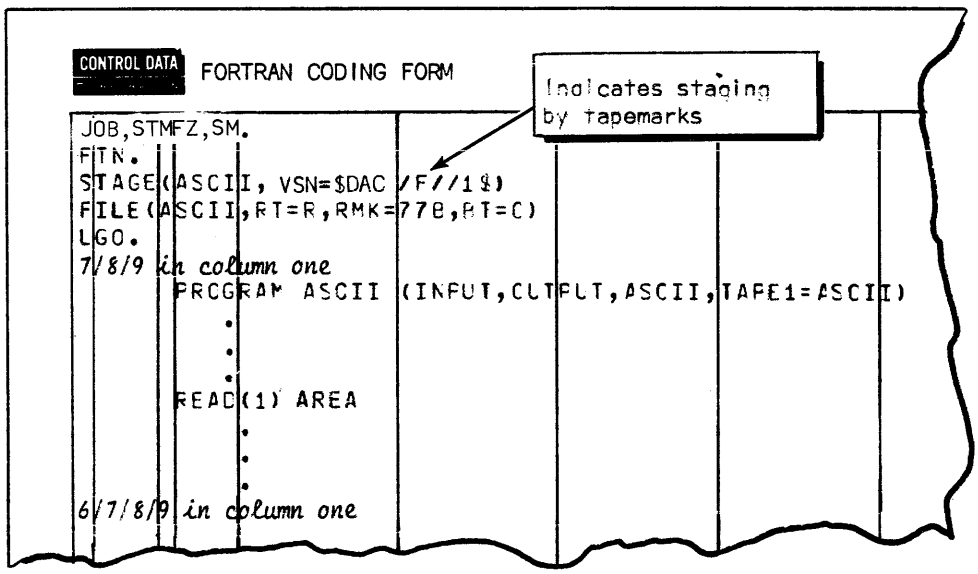
To stage by tapemarks, use a VSN parameter with the following format.

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, 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.



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.

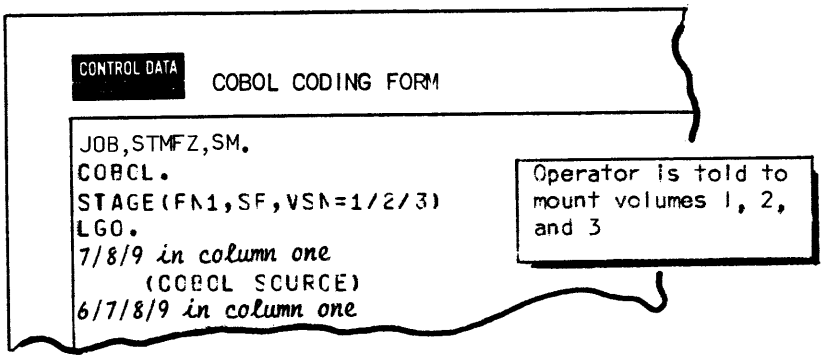
```
STAGE(lfn,....,SF)
```

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

An unlabeled file with no vsn list has one reel staged.

For a multivolume unlabeled file prestaged SF, the two terminal tapemarks for each end of volume are embedded in the data. A user-executed CLOSEM VOLUME (COMPASS language) macro is required after the first of two consecutive tapemarks have been read to advance over the second tapemark and continue processing data on the next volume.

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



Example 6-9. Staging Entire File

USING VOLUME SERIAL NUMBER FOR POSTSTAGING

When poststaging 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:

$$VSN=vsn_1/vsn_2/vsn_3/.../vsn_n$$

For a labeled tape, be sure that there are enough vsn's specified for all volumes of the poststaged 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 3.4.

† This section applies only to sites having on-line tapes in the configuration.

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.

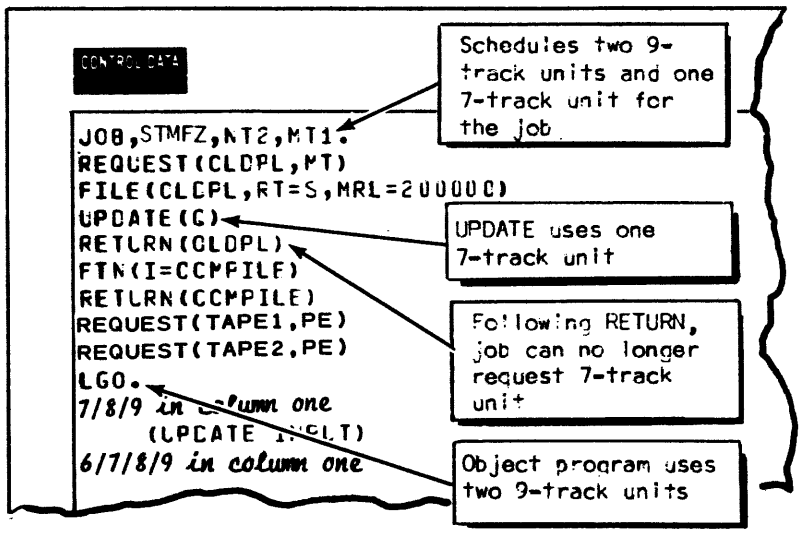
```
REQUEST(lfn,MT/NT,...)        or        REQUEST(lfn,density,...)
```

The MT/NT or density parameter must be present to distinguish the REQUEST from a mass storage REQUEST. MT or a 7-track density (LO, HI, HY) requests a 7-track unit; NT or a 9-track density (HD or PE) requests a 9-track unit.

A new request may be issued for a file after the previously requested file has been unloaded. Otherwise, subsequent REQUEST statements are ignored for a file. A STAGE statement and a REQUEST statement cannot be used concurrently for the same file, except to specify transfer unit size, allocation unit size and/or permanent file residence for the staged file.

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

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



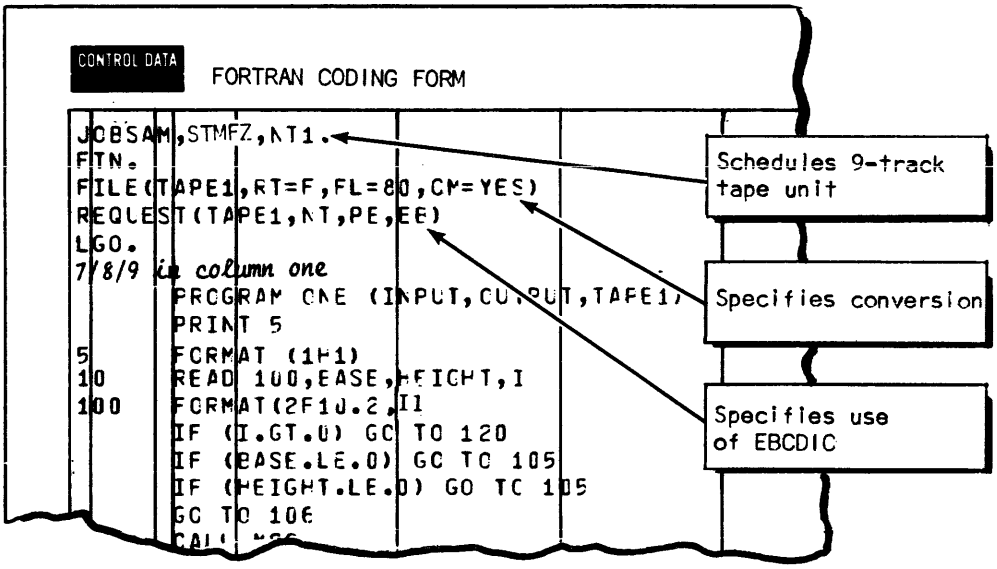
Example 6-10. Scheduling and Requesting On-Line Tape Units

CHARACTER CONVERSION AND PARITY

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

CM=NO and SPR=YES must be specified if READM/WRITEM macro operations are to be performed.

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



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. This also means a user cannot backspace or skip backward across volume boundaries. To rewind to the beginning of the first volume, close and unload the file and re-request the first volume.

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 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.

$$VSN=vsn_1/vsn_2/vsn_3/.../vsn_n$$

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

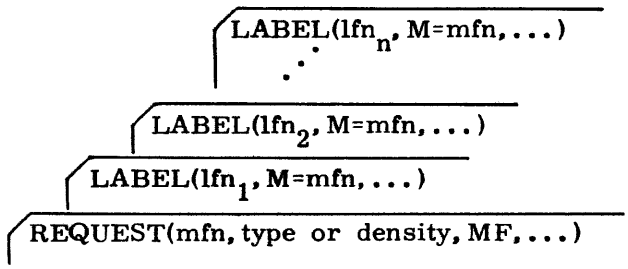
MULTIFILE ON-LINE TAPES

A multifile on-line tape (refer to Figure 6-2 is a volume or volume set containing more than one file. The file must be standard labeled (refer to Section 11).

A user specifies that an output file is to reside on a multifile set or that an input file resides on a multifile set through the use of the MF parameter on the REQUEST statement for the tape unit and through the use of LABEL statements.

On the REQUEST statement, instead of the first parameter being a logical file name, it is a multifile name (mfn) which is a maximum of 6 alphanumeric characters.

The object program or the LABEL statements provide the lfn's for the volume set. The LABEL statement for each file on the multifile set contains an M=mfn parameter where mfn is the same as on the REQUEST statement.



Only one file on a multifile set can be accessed at a time. Because multifile processing requires automatic volume switching, this feature is available only through the COBOL language and through use of COMPASS language macros.

The system prohibits a user from positioning a file beyond the labels.

When he accesses a file, the user specifies the first VSN for the multivolume set, or if he knows on which volume a file begins, he specifies a volume later in the set. The system searches forward for the file that matches the LABEL requirements.

MOUNT OPTION

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

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

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.

SPR=YES and CM=NO must be specified if READM/WRITEM macro operations are to be performed. Refer to the Record Manager Reference Manual.

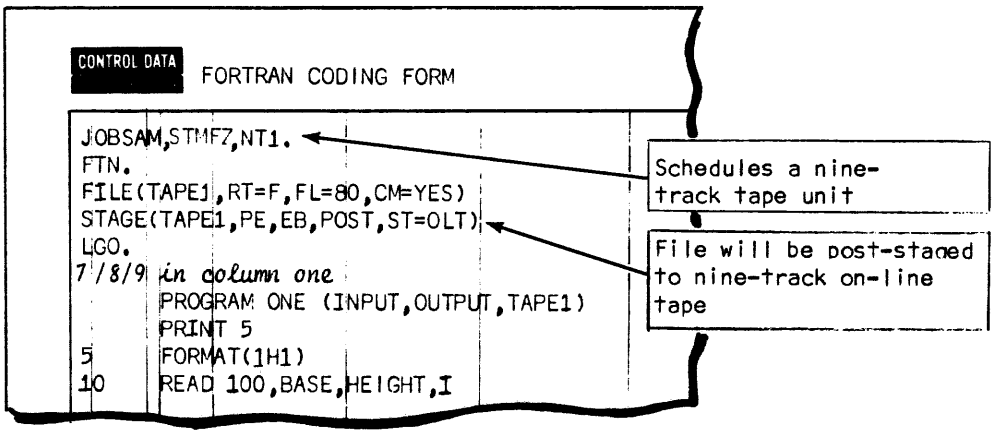
USING ON-LINE TAPE UNITS FOR STAGING

When using magnetic tape files, staging provides more efficient use of the tape units and the CPU than on-line usage does. Thus, a user should specify staging in preference to on-line tape use whenever possible.

To increase the staging capability of the system, SCOPE 2 allows a user to request staging from an on-line unit. Staging occurs as for any other tape unit at a station, that is, prestaging occurs when the file or volume is opened, and poststaging occurs when the file is closed/unloaded.

Staging from an on-line tape unit requires that the on-line tape be scheduled on the job statement and that the job contain a STAGE statement containing an ST parameter indicating that an on-line tape unit is the source or destination of the staged file. The identifier for an on-line tape unit is either the physical identifier or one of the logical identifiers of the SCOPE 2 Operating System on which the job is being processed.

In Example 6-12, file TAPE1 is poststaged to a 9-track on-line tape unit. The presence of NT1 on the job statement schedules the unit; the unit is assigned to the job when the file is closed/unloaded and the operator is requested to mount the tape.



Example 6-12. On-Line Staged Tape

For file staging by on-line tapes, the on-line tape unit is assigned to the job for each volume and released to the system after the volume is staged.

NOTE

When an installation site chooses labeled tape as its default, the UL parameter on the STAGE statement means unlabeled tape is used. Refer to the SCOPE 2 Reference Manual for a description of UL.

UNLOADING/RETURNING ON-LINE 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 on-line tape units scheduled for your job on the job identification statement, use the UNLOAD statement.

UNLOAD(lfn$_1$,lfn$_2$,...,lfn$_n$)

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(lfn$_1$,lfn$_2$,...,lfn$_n$)

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 CDC CYBER station generally follow CDC magnetic tape error recovery standards. Nonstandard recovery is attempted by the 7611-1 I/O Station.

For an unrecovered read parity error on a staged tape, the operator is given control to retry the procedures, to accept the data, or to terminate the job. If the station is a 7611-1 and the operator accepts the bad data, the block is dropped by the station. For all other stations, staged tapes for which the operator accepts bad data or for an on-line tape, further processing depends on the EO parameter on the FILE statement or macro. See Program Exit Conditions.

SPECIFYING NO RECOVERY

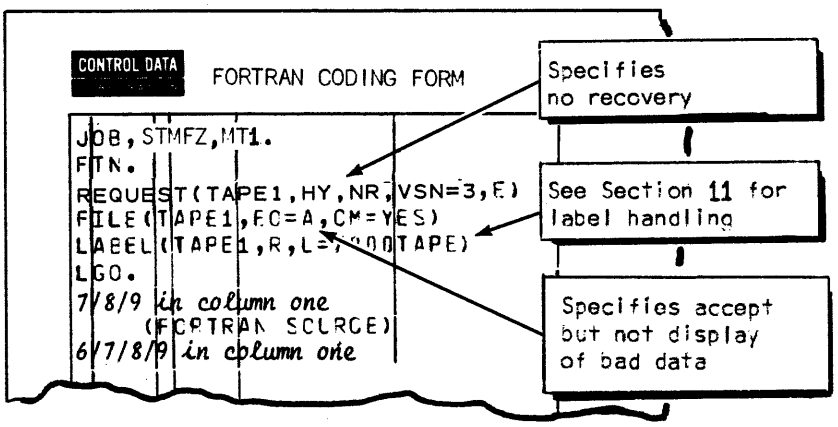
For on-line magnetic tape files, the user may specify that the recovery procedures not be performed if a read parity error is detected on a bad block. The block is transferred to the user buffer with the parity flag set in the first record. The action taken by the record manager depends on the setting of the error option (EO) field in the FIT for the file as described in section 5.

For staged magnetic tape files (including staged on-line tape files), the user may also specify no recovery but the parameter does not mean the same as for on-line magnetic tape files. In the case of staged files, specification of no recovery does not affect whether or not the recovery procedures are performed. In the event of read or write parity errors, no recovery means only that the operator is not notified each time an error is detected. If the error is unrecovered, the action taken by record manager depends on the contents of the EO field in the FIT when the record is read.

To specify no recovery, place NR on the REQUEST or STAGE statement, as follows:

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

In Example 6-13, NR is specified on the REQUEST statement and EO=A is specified on the FILE statement.



Example 6-13. 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 fewer 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 peripheral devices that a program can directly access include system and removable mass storage devices and on-line magnetic tape units. This section describes how files are assigned to mass storage and how space is allocated for the file.

INTRODUCTION TO REQUEST STATEMENT

When a file is to be created on mass storage, the user has the option of supplying a REQUEST statement to define characteristics relating to mass storage usage. This REQUEST statement differs from that for on-line magnetic tapes in that no device type specification is required. Through the REQUEST statement, the user may designate the type of mass storage device or can specify a particular mass storage device or set on which the file is to reside. If the user intends to make the file permanent, he must indicate on the REQUEST statement that the file is to reside on a permanent file device. Other parameters of the REQUEST statement permit the user to control the size of the transfers and the amount of mass storage allocated on each mass storage assignment.

The control statement has the following format.

REQUEST(lfn, p_1, p_2, . . . , p_n)

p_i	Significance
AF, AR, or AY	Device type (7638, 819, or 844-2 mass storage subsystem)
SN=setname	Setname
VSN=vsn_1/vsn_2/.../vsn_n	Volume serial number of set member
*PF	Permanent file device
An	Allocation unit
Tn	Transfer unit size
WCK	Write check

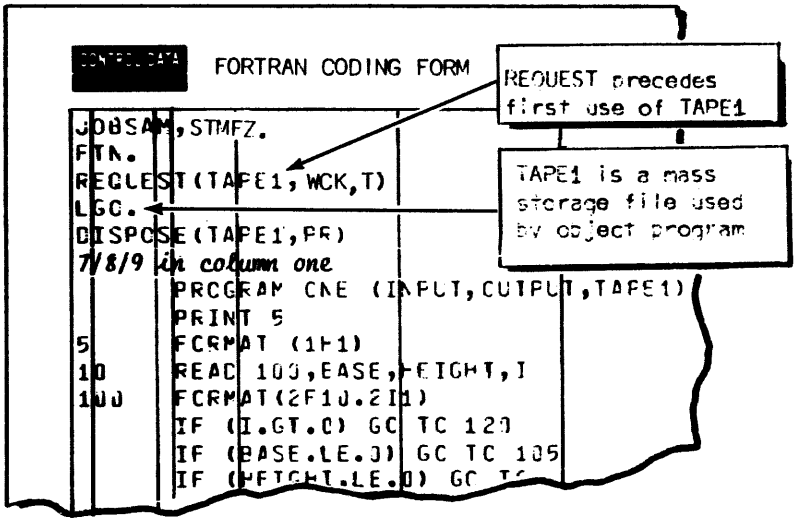
The parameters other than lfn can be in any order. They are described in greater detail later.

The An, Tn, and WCK parameters can also appear on a STAGE statment. (WCK applies only for poststaged files.)

To be effective, a REQUEST statement must be encountered before the file is opened for the first time. The record manager ignores subsequent REQUEST statements for a file until the file is closed/unloaded for the previous REQUEST. The only exception is a subsequent REQUEST that changes the transfer unit size.

Normally, REQUEST and STAGE statements are not used for the same file. Both can be used, however, when a REQUEST is used in conjunction with STAGE to specify transfer unit size, allocation unit size, and/or permanent file residence for the staged file.

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



Example 7-1. REQUEST Statement Placement

MASS STORAGE SETS

The three mass storage systems supported by SCOPE 2 are the 7638 Disk Storage Subsystem, the 844-2 Disk Storage Subsystem (on which are mounted Model 881 Disk Packs) and the 819 High Capacity Disk Subsystem. For purposes of equipment designation, each half of a 7638 is a separate device (sometimes referred to as an 817), each 881 disk pack is a separate device, and each 819 unit is a separate device.

Each device is a member of a mass storage set. A set is uniquely identified by a 1- to 7-character name known as the setname. Each set member is identified by a unique 1- to 6-character volume serial number (vsn). Any number of jobs can concurrently access a set.

The two types of mass storage sets are system and removable.

SYSTEM SET

SCOPE 2 requires the system set as a minimum. This set is created during system deadstart. Devices in the set contain the operating system, system permanent files, spooled files, staged files, libraries, and space for scratch and user permanent files. Because the system set must be present while the system is running, it is termed nonremovable. The system set can consist of any combination of mass storage devices.

Since the user has no control over the formation of the system set, the subject is not covered in this publication. Refer to the SCOPE 2.1 Installation Handbook for deadstart procedures.

REMOVABLE SETS

Depending upon the installation configuration, the user may have the option of defining and using removable sets. A job is not required to use removable sets; files are assigned to the system set by default.

A removable set is a set whose members may be logically mounted or dismounted. A logical mount is the association of a set member with a job. A set may be considered a removable set even though it resides all or in part on physically non-removable devices, such as a 7638 Disk Storage Subsystem. Or it may reside on a device that is physically as well as logically removable, such as the 881 Disk Pack on an 844-2 Disk Storage Subsystem. Every job requiring access to a removable set member must logically mount the member.

Each device is identified by a volume serial number (vsn), which identifies the device when it is to be incorporated into a set or is to be put into use. The vsn provides a means of identifying the device logically, and may only be assigned to a device through the use of the LABELMS control statement.

A user assigns the setname to the set when he creates the set or adds a device to it using the ADDSET control statement. The first device assigned to a set is known as the master device because it contains tables and information relevant to all the devices in the set. Creating a set does not automatically make the set available for use; the user must ensure that the master device is mounted before assigning any files to the set or referencing any files on the existing set. A file can be assigned to the set and can be limited to specific set members or can be allowed to overflow from one device to another according to REQUEST statement parameters.

Device scheduling is also a user consideration when using removable devices. For additional detail, refer to Using Removable Sets in this section.

HOW MASS STORAGE FILES ORIGINATE

Consider what happens when your job creates an output file. If there is a REQUEST statement assigning the file to magnetic tape, the file is written on an on-line tape, as described in section 6. If there is no REQUEST statement preceding the first reference to the file, all defaults apply and the system looks for room for the file on one or more of the system set members.

If a REQUEST statement assigns the file to a specific type of device in the system set or to a particular set member, the system looks for room for the file only on devices meeting the indicated criteria.

If a REQUEST statement assigns the file to a removable set, the system looks for room for the file only on members of the specified set that meet all of the criteria indicated on the REQUEST statement.

Space on a set member is assigned to a mass storage file as needed in units known as allocation units. Before writing on the file, the system also creates an intermediate buffer area for the file in large central memory. The size of this area can be the same as the allocation size or can be smaller. When a program performs a write, the data is first moved from small central memory to the LCM buffer area. When the buffer area is full of data, the buffer manager initiates a physical transfer of the data to mass storage. This minimizes the number of physical writes. The amount of data depends on the size of the LCM buffers and is referred to as a transfer unit.

After the file is created on mass storage, it can be used by the job in a number of ways.

- It can be used as a temporary scratch file that provides input to the same program or to some later job step and can then be released (destroyed). The LGO file is a common example of a scratch file.
- The file can be used as a scratch file and then cataloged as a permanent file before job end. In this case, the first reference to the file must be preceded by a REQUEST statement assigning the file to a permanent file device on either a system or removable set.
- A copy of the file can be routed to any linked mainframe (including the host mainframe) and be made permanent by using the SAVEPF control statement with the appropriate mainframe identifier.
- The file can be poststaged to magnetic tape. In this case, a STAGE statement requesting poststaging must precede the first reference to the file. The system always assigns staged files to system mass storage.
- The job can route the file to a printer or punch. In this case, the file must reside on system mass storage.

An input file referenced by a job is on mass storage unless the first reference to the file is preceded by a REQUEST for an on-line magnetic tape unit.

A file on mass storage may have originated in a number of ways.

- It may have been spooled in from punched cards as the INPUT portion of job deck.
- 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 the system to transfer all or part of the file from the station to system mass storage (refer to section 6).
- It may be a permanent mass storage file. In this case, a permanent file access control statement precedes the first use of the file. A permanent file may reside on system mass storage (system set), or it may reside on a removable set or originate from mass storage at a linked NOS/BE or SCOPE 2 mainframe. The first attempt to open a file staged from the linked mainframe causes the system to transfer a copy of the file from the linked mainframe to system mass storage.
- The file may be a scratch file created earlier in the same job.

USING THE SYSTEM SET

A REQUEST statement is not required to cause a scratch file to be assigned to the system set. However, a REQUEST statement with the *PF parameter is required for a file if it is going to be cataloged (section 8). A user may also supply a REQUEST statement to specify allocation unit and transfer unit sizes or to specify device type, setname, or vsn. However, the user does not normally specify device type, setname, or vsn for system sets because the default for these parameters applies for the system set. The scheduling algorithms for assigning the files to the various members of the system set are designed to equalize the use of the set members.

ASSIGNMENT BY DEVICE TYPE

If the installation has different types of devices (844s, 819s, or 7638s) in the system set, a user may indicate the type of device to which the file is to be assigned. AF on the REQUEST statement causes assignment to a 7638; AY causes assignment to an 844-2 device; AR causes assignment to an 819. When no type is indicated, the set member on which the file is to reside is determined by other criteria (for example, according to the scheduling algorithm or by setname and vsn).

ASSIGNMENT BY SETNAME

When the file is to be assigned to the system set, specification of the setname is not required because the normal default for the setname is the system set. The system setname is an installation deadstart option (normally SYSTEM). If the default for a job has been changed to a removable setname by the presence of a SETNAME control statement (described later in Using Removable Sets), the user must either return the setname to the system default through a SETNAME statement with no parameters or indicate that the system set is desired for file residence by including the parameter SN=setname on the REQUEST statement where setname is the name of the system set.

ASSIGNMENT BY VSN

A VSN parameter is not normally used in conjunction with the system set. When the user knows the volume serial numbers (vsn's) associated with the system set members, he can, however, specifically designate that a file is to reside on one or more set members. If the file is to become a permanent file, all the vsn's must be for permanent file devices; if the device type is specified, all the vsn's must be for devices of the specified type. Otherwise, job step abort occurs.

To specify a single vsn, use the parameter VSN=vsn, where vsn is the 1- to 6-character name of the device as assigned by the installation. To specify a number of vsn's, use the parameter VSN=$vsn_1/vsn_2/.../vsn_n$ where each vsn identifies a member of the system set. Space is assigned to the file from the members in the sequence listed.

USING REMOVABLE SETS

Use of removable sets involves the following steps.

1. Claiming on job statement the number of drives and specifying the number of packs to be used (844-2 Disk Storage Subsystem only)
2. Set creation
3. Mounting of set members

4. Requesting file assignment to the removable set or to specific members of the set

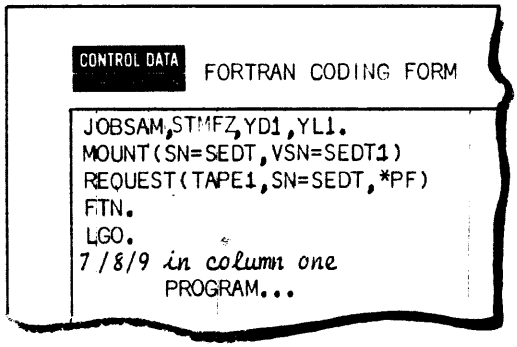
5. Dismounting of set members

DEVICE SCHEDULING - 844-2 DISK STORAGE SUBSYSTEM

A job can use or assign files to 844-2 removable set members only if it has claimed the 844-2 drives on the job statement. That is, the YDd parameter must be present, where d is an octal count of the number of drives to be used concurrently by the job.

The user must also specify the number of packs to be used. This count is specified as YLp, where p is the maximum number of 881 disk packs that can be used during the job. The default for the pack count is the number of drives claimed; conversely, the default for the number of drives claimed is the pack count. Each time a job dismounts a disk pack, the system decrements the pack count by one. When the current count for YL becomes less than the number of claimed drives (YD), the system returns a drive to the system from the job, making it available for other jobs.

Example 7-2 illustrates a request for the file named TAPE1 to be assigned to an 844-2 removable set. The job statement claims one disk drive to be used for one disk pack. The MOUNT and REQUEST statements are defined later.



Example 7-2. Scheduling of Removable Device

CREATION OF REMOVABLE SETS†

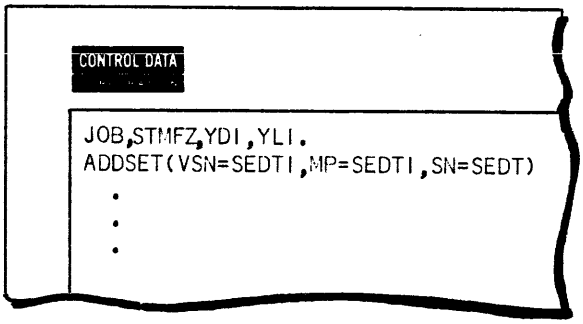
The first step in creation of a set is identification of the master device. This device contains information about the set for use by the system. To identify the master device for a set, use an ADDSET control statement with the VSN and MP parameters the same.

†Although any of the three types of mass storage supported by SCOPE 2 may be used as removable devices, the user will typically only be concerned with 844 drives/881 disk packs when creating removable sets. For this reason, examples involve the 844-2 Disk Storage Subsystem.

ADDSET(VSN=vsn, MP=vsn, SN=setname, DT=dt)

setname is the 1-to-7 alphanumeric character name through which the set will be recognized in subsequent references. dt is the device type. The default is an 844 device.

In Example 7-3, the master device for set SEDT is SEDT1.

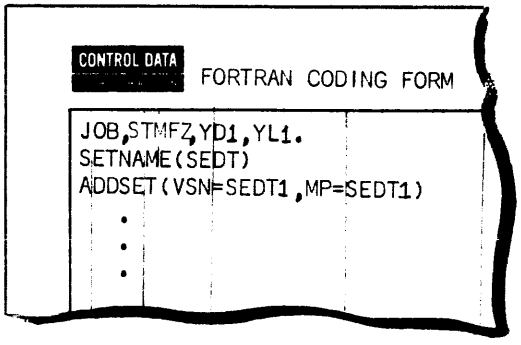


Example 7-3. Identify Master Device

A user could specify the setname through the SETNAME control statement as an alternative to using the SN parameter on any of the following statements.

ADDSET
MOUNT
DSMOUNT
REQUEST
ATTACH
DELSET

In Example 7-4, the default setname is set to SEDT. Following the SETNAME statement and until a SETNAME with no parameters is encountered, the default for the job is the removable set defined by the SETNAME statement and not the system set.



Example 7-4. Changing Default Setname for Job

The user should be aware that each use of an ADDSET control statement automatically decrements the YLp pack count, claimed on the job statement, by one. Thus, for each occurrence of an ADDSET control statement in the job, one pack should be added to the pack count. Also, after the set member is created, the ADDSET control statement explicitly dismounts the pack. If during the course of the job which creates a master pack the user intends to create files on that pack, he must first use a MOUNT control statement to remount the pack.

DESIGNATING A MEMBER AS A PERMANENT FILE DEVICE

When a member is introduced into a set, whether it is a master device or a member device, it is usually designated as a device on which the user can write permanent files.

Thus, if SEDT1, the master device in Example 7-4, were to be designated as a permanent file device, the control statement could be as follows, where *PF makes the member a permanent file device.

```
ADDSET(SN=SEDT,VSN=SEDT1,MP=SEDT1,*PF)
```

If a device is not designated as a permanent file device, no file on it can be cataloged as a permanent file. This means that when the set member is dismounted, any information on the file is not retained and is, therefore, irrecoverable.

Other parameters associated with the master device are the member limit, the retention period, and the permanent file limit.

MEMBER LIMIT

By default, the maximum number of members that can be grouped into a set is 50. The maximum value for maxmem is 50. To change this limit, add the NM=maxmem parameter to the ADDSET for the master device.

```
ADDSET(VSN=SEDT1,MP=SEDT1,...,NM=10)
```

RETENTION PERIOD

If not specified on the ADDSET for the master device, the retention period for the set is defined according to an installation parameter. To explicitly define the retention period, add the RP=rp, where rp is a decimal number of days, to the ADDSET statement.

PERMANENT FILE LIMIT

If not specified on the ADDSET for the master device, the limit on the number of permanent files that can reside on all members of the set is defined by an installation parameter. To explicitly define this limit, add the NF=maxpfn parameter to the ADDSET, where maxpfn is a decimal limit.

ADDING MEMBERS TO A SET

A set can consist of only the master device or can contain as many members as allowed by the member limit parameter.

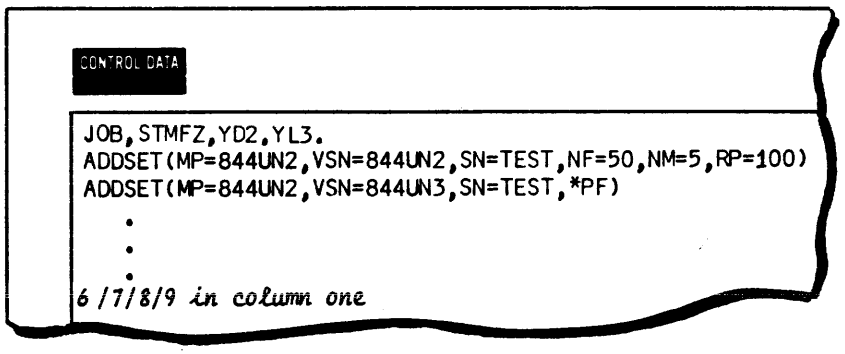
Adding a member to a set again requires the ADDSET control statement. However, for a new member, the vsn supplied by the VSN parameter and the vsn supplied by the MP parameter will be different.

```
ADDSET(SN=SEDT, VSN=SEDT2, MP=SEDT1, *PF)
```

This control statement would add a member to the set defined in Example 7-4.

Example 7-5 illustrates creation of a set containing two devices, 844UN2 and 844UN3. The first ADDSET identifies the master device, sets the maximum number of permanent files to 50, the number of set members to 5, and the retention period to 100 days.

The second ADDSET adds a member device that can contain permanent files.



Example 7-5. Adding a Member to a Set

MOUNTING SET MEMBERS

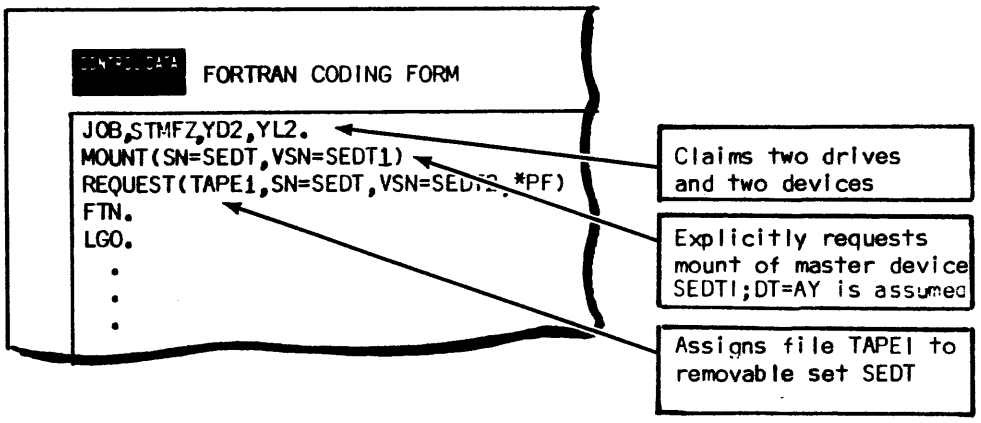
After the master device of a set has been defined, the set can be used for file assignment. Each job that assigns files to the set or uses any permanent files on the set must mount the master device prior to the first reference to the file. An explicit (control statement) mount is required for the master device only. Any other set members can be mounted by the system (implicit mount), as needed. If the removable set is on 844-2 mass storage, each pack used by the job must be included in the job statement YL count.

The MOUNT statement has the following format.

```
MOUNT(VSN=vsn, SN=setname, DT=dt)
```

vsn identifies the device to be mounted; the SN parameter can be omitted if the default setname has been altered through a SETNAME. Otherwise, it is used to identify the set for which vsn is a member. DT specifies the mass storage device type; AF for a 7638, AR for an 819, or AY for an 844-2. AY is assumed by default.

Example 7-6 illustrates a job that mounts a master device named SEDT1 on set SEDT and uses the set for the TAPE1 file. The file will be assigned to SEDT2, which must have been added to the set as a permanent file device. Set member SEDT2 will be implicitly mounted by the system.



Example 7-6. Mounting the Master Device for a Removable Set

REQUESTING USE OF REMOVABLE SETS

After a user has created a set and mounted the master pack, the set is ready to be used.

ASSIGNMENT BY SETNAME

When a file is to be assigned to a removable set, specification of the setname is required because the normal default for the setname is the system set. The setname can be specified either by changing the default to the name of the removable set through use of the SETNAME statement or can be specified through the SN parameter on each of the control statements that the user wishes to apply to the set (for example, the REQUEST statement, ATTACH statement, ADDSET statement, etc.).

ASSIGNMENT BY VSN

The VSN parameter on the REQUEST statement permits a user to designate which members of a removable set are to contain a file. This parameter is optional. If it is omitted, the file is assigned to any members of the set meeting other criteria specified on the REQUEST statement. The DT parameter must be included if the removable set is on either a 7638 or 819 Mass Storage System.

To specify a single vsn, use the parameter VSN=vsn, where vsn is the 1- to 6-character name of the device as determined when the device was initialized. To specify a number of vsn's, use the parameter VSN=$vsn_1/vsn_2/.../vsn_n$ where each vsn identifies a member of the removable set. Space is assigned to the file from the members in the sequence listed with the exception that if the master device is in the list, the file is always assigned to the master device first. If the list of vsn's is exhausted, members are selected as if there were no list. A set full condition occurs if no space is available.

Now, consider the REQUEST statement used in Example 7-6. The SN=SEDT parameter on the REQUEST statement causes the file to reside on the set named SEDT. Omitting the REQUEST statement or the SN=SEDT parameter would have caused the file TAPE1 to be assigned to the system set. The SN parameter is required if the REQUEST is not preceded by a SETNAME(SEDT) statement. The VSN parameter causes the file to be assigned to the device identified by the vsn SEDT2. This means that the job needs at least two 844-2 drives and will use two drives, one for the master device, SEDT1, and one for the member device, SEDT2. The *PF parameter on the REQUEST statement is described in detail in section 8. This parameter ensures that the file resides on a permanent file device and allows the file to be cataloged as a permanent file later in the job.

DISMOUNTING SET MEMBERS

The primary reason a user may wish to dismount a set member is to minimize the number of 844 disk drives required for a job. All set members used by a job are logically dismounted at job end.

A member device can be implicitly dismounted prior to job termination if the drive is needed by another device and if there are no open files on the device. A master device can be dismounted only by using a DSMOUNT control statement (explicit dismount) or as a result of job termination.

The following control statement can be used to explicitly dismount a set member.

```
DSMOUNT(VSN=vsn, SN=setname)
```

The vsn identifies the set member to be dismounted. The SN parameter can be omitted when the default setname has been altered through a SETNAME. Otherwise, it identifies the set for which the vsn is a member.

The dismount causes any open files on the device to be closed and unloaded. If the device being dismounted is the master device, the dismount causes all set members to be dismounted and all files that may have been opened to be closed and unloaded.

A drive holding a set member becomes available for reallocation when no job has the set member mounted, that is, all the jobs using the set member have explicitly or implicitly dismounted the set member.

A drive holding a master pack becomes available for reallocation when no job has the master pack mounted, that is, all the jobs that were sharing the master pack have terminated or explicitly dismounted the pack.

DELETING SET MEMBERS

The DELSET control statement is provided for use when a member is to be removed from the set or the entire set is to be deleted from the system.

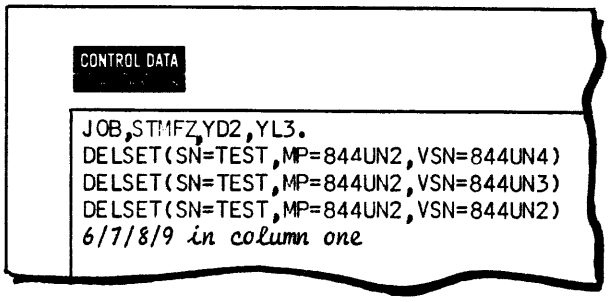
Format:

```
DELSET(VSN=vsn, MP=vsn, SN=setname, DT=dt)
```

Parameters can be in any order. The VSN, MP, and DT parameters identify the set member to be deleted, the master device for the set, and the device type of the master set (default device type is 844). When the VSN and MP parameters are the same, the master device is deleted. The SN parameter is required if the DELSET is not preceded by a SETNAME statement. Attempting to delete members of the system set results in job step abort.

Deleting the master device, in effect, deletes the entire set. A job step abort occurs if a DELSET attempts to delete a set member from which not all permanent files have been purged or not all local files have been returned.

In Example 7-7, three separate DELSET statements delete a set consisting of three set members. The same function could be performed with just the last of the three DELSET statements.



Example 7-7. Deleting Set Members

MINIMUM ALLOCATION UNITS (MAU)

Each mass storage file is automatically assigned space on an as-needed basis in units known as minimum allocation units (MAUs).

The number of MAUs assigned to a file at a time is determined by the following factors.

- For the input file and permanent files attached from any linked mainframe, the number of MAUs is determined by an installation option.
- For poststaged files, the number of MAUs is determined by the user through use of the allocation (An) parameter on the STAGE control statement or macro, REQUEST statement or macro, or by an installation option if the An parameter is omitted.
- For all other files, the number of allocation units is determined by the user through the allocation (An) parameter on the REQUEST control statement or macro. The number of MAUs is set to one by default.

The size of an MAU depends on the type of mass storage device, as follows:

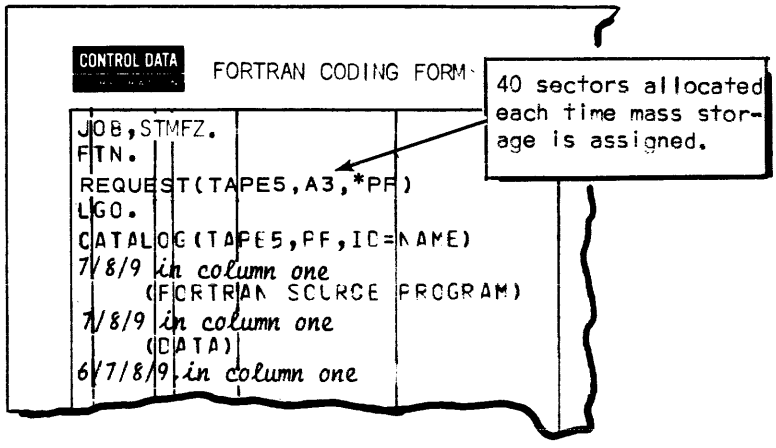
Device	Size of MAU
7638	25,600 characters (five sectors)
844-2	35,840 characters (1/8 of a cylinder)
819	25,600 characters (5 sectors)

TABLE 7-1. ALLOCATION OF MASS STORAGE

		Allocation for 7638			Allocation for 844-2			Allocation for 819		
An	MAUs	Sectors	Tracks	Characters	Sectors	Cylinders	Characters	Sectors	Head Gaps	Characters
A0	1	5	1/8	25,600	56	1/8	35,840	5	1/4	25,600
A1	2	10	1/4	51,200	112	1/4	71,680	10	1/2	51,200
A2	4	20	1/2	102,400	224	1/2	143,360	20	1	102,400
A3	8	40	1	204,800	448	1	286,720	40	2	204,800
A4	16	80	2	409,600	448	1	286,720	40	2	204,800

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 7-8 shows a REQUEST statement for a FORTRAN output file (TAPE5) which the user knows is going to be very large.



Example 7-8. Using the REQUEST Statement Allocation Parameter

TRANSFER UNIT SIZE

The amount of data transferred at a time between an I/O buffer in LCM and a mass storage device is known as the transfer unit. The transfer unit is used for determining the size of the LCM buffer and can be specified on a REQUEST or STAGE control statement or macro. The smallest unit that can be requested is 5120 characters. The largest unit depends on the device type. Transfer units are specified as the minimum (T) or as power-of-two multiples of minimum allocation units (Tn). T0, for example, indicates a multiple of 2^0 or one MAU. The allocation parameter (An) affects the size that can be specified for Tn because the amount of data transferred cannot exceed the amount of space allocated at a time. However, less data can be transferred than is allocated at a time in order to reduce LCM requirements. A large transfer unit size reduces the number of physical data transfers.

REQUEST(lfn,...,Tn)

Transfer Parameter	Buffer Area in Characters			
	7638	844-2	819	
T	5,120	5,120	5,120	Allocation can be A0, A1, A2, A3, or A4
T0	25,600	35,840	25,600	
T1	51,200	71,680	51,200	Allocation can be A1, A2, A3, or A4
T2	102,400	143,300	102,400	Allocation can be A2, A3, or A4
T3	204,800	286,720	204,800	Allocation can be A3 or A4
T4	409,600	-	-	Allocation must be A4 (7638 only)

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 7-1.

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 any linked mainframe, the transfer unit size equals the allocation unit size defined by system default.

A. TRANSFER UNIT SIZE FOR 7638 DISK STORAGE SUBSYSTEM

An \ Tn	T	T0	T1	T2	T3	T4
A, 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

B. TRANSFER UNIT SIZE FOR 844-2 DISK STORAGE SUBSYSTEM

An \ Tn	T	T0	T1	T2	T3
A, A0	5120	35,840			
A1	5120	35,840	71,680		
A2	5120	35,840	71,680	143,300	
A3, A4	5120	35,840	71,680	143,360	286,720

C. TRANSFER UNIT SIZE FOR 819 DISK STORAGE SUBSYSTEM

An \ Tn	T	T0	T1	T2	T3
A, 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

Notes

T is 5120 characters.
T4 is not allowed for 844-2 or 819.
A4 defaults to A3 for 844-2 and 819.

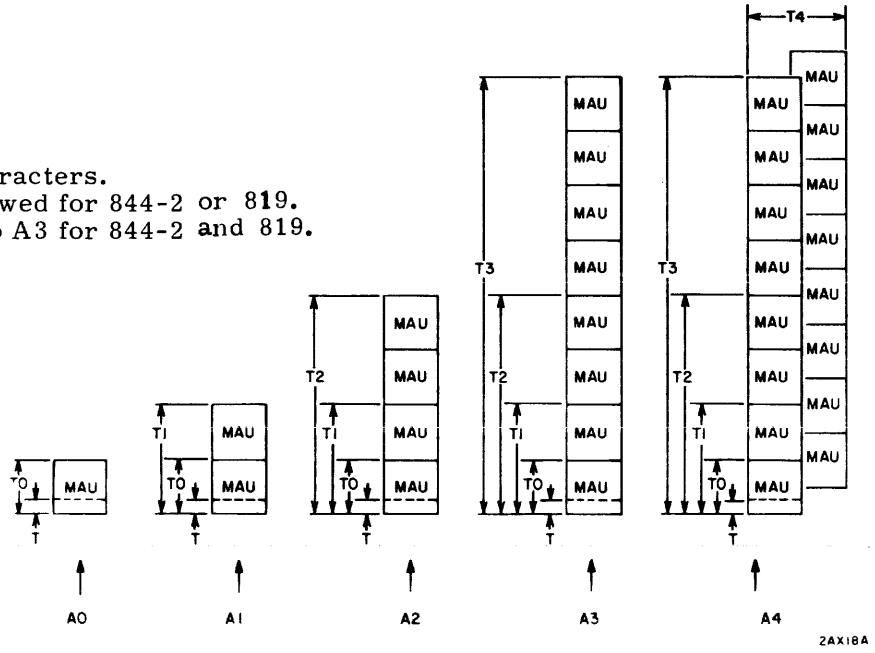
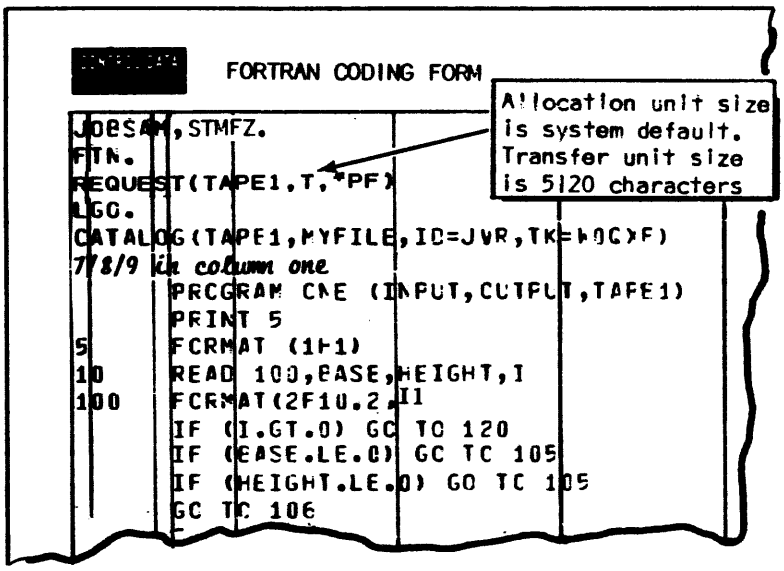


Figure 7-1. Relationship of MAUs and Transfer Unit Size

Usually, the system default is the most efficient. The Tn parameter can also be used on a STAGE statement for a staged file.

In Example 7-9, notice the REQUEST statement for TAPE1.



Example 7-9. Using the REQUEST Statement Transfer Unit Parameter

WRITE CHECK OPTION

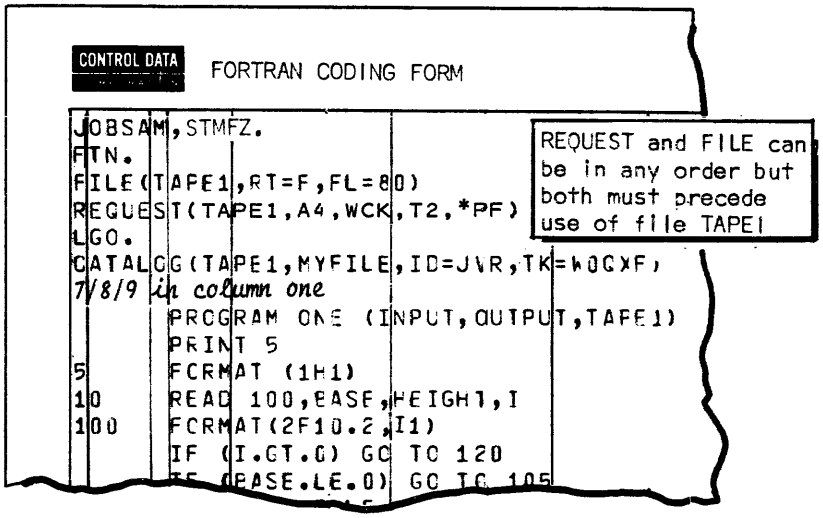
An option of the REQUEST statement permits a user to request that the record manager follow each write with three reads to ensure that the data can be read without parity error and that the checksum is correct. If any one of the three reads is successful, the write is successful. If all of the reads are unsuccessful, the write is unsuccessful, and the action taken by the system depends on the setting of the EO parameter on FILE statement or macro.

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 7-10 illustrates a FORTRAN program that uses WCK on file TAPE1. Here, use use of WCK could prevent cataloging of a useless file. If an irrecoverable read error is encountered, the job aborts. There is no way to recover the file following the abort.



Example 7-10. Using the RFQUEST Write Check Parameter

WCK is also allowed on a poststage 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.

- Receive a partial output of a large file
- Ensure that output is not lost if an abort occurs later in the job
- Reduce resources used by the job, thus reducing job cost and promoting efficient use of system resources
- Post a dependency (TRANSF)
- 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$_n$)

Type of File Being Returned	Action Taken on Return
Scratch	Mass storage assigned to the file is released.
OUTPUT, PUNCH, PUNCHB	The accumulated data is sent to the unit record device. 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 a job.
File with delayed disposition (use of * on DISPOSE statement)	The file is immediately disposed to a station unit record device.
Poststaged file	The file is immediately staged out.
Prestaged file	Mass storage assigned to the file is released.
SCOPE 2 permanent file	The file is detached from the job.
A linked mainframe permanent file	Mass storage assigned to the copy of the file is released.

An UNLOAD statement has the same effect as RETURN for mass storage files. When a file is returned, its LCM buffers are released.

The system returns all unreturned files when a job terminates.

For a removable device, returning all the files that the job has open on the device causes the device to be dismounted. When the job returns all the files open on the set, the master device is dismounted.

Example 7-11 illustrates a job that returns files ST, OLDPL, and COMPILE when they are no longer needed by the job.

CONTROL DATA

```
JOB,STMFZ,SM.
STAGE(ST)
ATTACH(OLDPL,PFNAME,ID=USER,...)
UPDATE(C,I=ST)
RETURN(OLDPL,ST)
FTN(I=COMPILE)
RETURN(COMPILE)
LGO.
6/7/8/9 in column one
```

Example 7-11. 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.

When a job reaches its mass storage limit, whether the limit is defined by default or by a LIMIT statement, the LIMIT statement cannot be used to extend the limit in order to continue the job.

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.

A mass storage file available to your job that is not already permanent can be made permanent (unless it is a poststaged tape file). A mass storage file is not made permanent unless you explicitly specify that it be made permanent through the use of the appropriate permanent file control statement or statements.

If you wish access to a permanent file that resides on the same computer system on which your job is running (that is, on the host computer system), you have the choice of using the ATTACH control statement and obtaining direct access to the file or of using the GETPF control statement and obtaining a working copy of the file. Similarly, you may either use the CATALOG control statement to directly catalog a file or the SAVEPF control statement to save a copy of the file. The advantages and disadvantages of direct and indirect file access are discussed below. If a permanent file resides or is to reside at a computer system in a linked mainframe environment other than the host mainframe (at a NOS/BE linked mainframe or another SCOPE 2 linked mainframe), the job always accesses a copy of the file which is staged to or from the linked computer system as a result of a GETPF or SAVEPF.

You may also purge a file at either location. In addition, you may modify a permanent file on the SCOPE 2 host system mass storage when you have the necessary permissions.

USING SCOPE 2 PERMANENT FILES

CYCLES

The SCOPE 2 permanent file manager maintains one to an installation defined number (default is five) of 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 999, 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 the next highest cycle.

LOGICAL AND PERMANENT FILE NAMES

A permanent file name is assigned to a permanent file when the permanent file is created and is used to identify the permanent file when a cycle of the file is attached. At all other times, the file is known by a logical file name.

The SAVEPF and CATALOG statements declare a scratch file identified by its lfn to be a permanent file under the permanent file name (pfn). Conversely, the GETPF and ATTACH statements declare that the permanent file having the specified pfn, user id, and cycle number is to be known to the attaching job by the lfn specified in the control statement.

Specify either pfn or both lfn and pfn. If both are specified, lfn is first and pfn is second. If only one of these parameters appears on a control statement, it is assumed to be pfn. The first seven characters of pfn are then used to form lfn.

A pfn is 1 to 40 alphanumeric characters. If characters other than 0 through 9 or A through Z are used, they must occur within 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 (ID)

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. The ID for a permanent file must be defined when the first cycle of the file is cataloged. Examples in this guide illustrate that ID parameters are required.

When no ID is defined on a SAVEPF or CATALOG, or when the ID parameter is omitted from an ATTACH or GETPF statement, the ID is assumed to be PUBLIC.

CREATING THE INITIAL CYCLE OF A FILE

Cataloging a permanent file may be accomplished by either of two processes; directly through the use of the CATALOG statement, or indirectly through the use of the SAVEPF statement. Either statement creates one cycle of a file, and also an entry for the permanent file in the system's permanent file directory. This entry includes the permanent file name, the cycle number, the creator identification, the retention period, 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 reading the file, increasing the size of the file, writing over existing data, cataloging a new cycle, or purging an existing cycle. If you specify all passwords but reveal none of them, only you as the creator have 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.

Before a file to be CATALOGed is created, it must be assigned to a permanent file device through a REQUEST statement containing the *PF parameter. A file saved using SAVEPF does not need to be preceded by a REQUEST statement; assignment to a permanent file device is automatic under SAVEPF.

The mass storage device on which a permanent file resides can be a member of either a system or removable set for CATALOG and ATTACH requests, but system set only for GETPF and SAVEPF. Refer to section 7 for more information on mass storage sets.

CATALOG

The first method of creating a permanent file is through the use of the CATALOG statement. CATALOGing a file merely means declaring a logical file associated with the creator's job to be a permanent file which is not destroyed at job end. An entry is created for the file in the permanent file directory. If no passwords are specified when the initial cycle is cataloged, any program that attaches the file has access to the file determined by defaults, which are discussed later. Passwords may be specified covering one or more operations; only those who know the passwords can exercise the permissions that the passwords represent.

A permanent file cataloged with or without passwords does not become available to other jobs until the job requesting the cataloging operation terminates or the job releases the file through a RETURN or UNLOAD statement. Until then, unless the MR=1 parameter is used, the file remains attached to the cataloging job with all permissions granted. If the parameter is included, it causes the file to be immediately available on a read-only basis to other jobs. Refer to Using the Multiread Parameter.

A CATALOG statement can be issued any time after the file has been created by the job and before the job terminates. A blocked file must be closed to be cataloged; if it is not, the job step aborts.

SAVEPF

The SAVEPF statement resembles CATALOG in that it also provides a means of creating a permanent file. The SAVEPF statement, however, causes a copy of a logical file associated with the creator's program to be created at any one of the mainframes in a linked mainframe environment, and cataloged as a permanent file at that mainframe. The mainframe is identified by the ST parameter on the SAVEPF statement, and may be either a NOS/BE linked mainframe or a SCOPE 2 linked mainframe. Cataloging of the copy of the logical file is accomplished at the time the SAVEPF statement is processed, and creates an entry for the permanent file in the permanent file directory at the mainframe specified in the ST parameter. If the ST parameter is not included in the SAVEPF statement, a copy of the logical file is saved at the host mainframe. The host mainframe may also be explicitly selected by including its logical identifier in the ST parameter. If the file is saved at some mainframe other than the host mainframe, any job which subsequently accesses the file must obtain a copy of it from that mainframe; the host mainframe on which the cataloging job runs does not keep a record of the file in its own permanent file directory.

The file cataloged as a permanent file at the linked mainframe becomes available to other jobs as soon as processing of the SAVEPF control statement has been completed. Upon completion of the SAVEPF processing, the logical file is still local to the cataloging job with all permissions granted.

A job may perform multiple SAVEPFs of the same logical file to create different or additional permanent files at a mainframe, subject to the restrictions on permanent files at that mainframe.

A SAVEPF statement can be issued any time after the file has been created by the job but before it is returned by either the RETURN statement or job termination. The file to be cataloged must reside on a system mass storage device. The job aborts if the file resides on a removable device.

The creator of the permanent file at a mainframe linked to the host mainframe can control the usage of the file in the same manner as a person who catalogs a file directly at that mainframe. Passwords may be supplied on the initial cycle to protect the file from unauthorized operations, just as on files saved at the host mainframe.

When you wish to save a file on SCOPE 2 system mass storage, it is advisable to do so by using the CATALOG statement rather than the SAVEPF statement when dealing with large files. Since SAVEPF saves a copy of the file attached to the job and not the file itself, there are two copies of the file on mass storage from the time the SAVEPF statement is processed until the job terminates or the local copy is returned, which unnecessarily ties up system mass storage resources. The CATALOG statement merely declares that the logical file is not destroyed when it is returned or the job terminates, and thus represents a more efficient usage of system resources. When performing multiple catalogs of a file on the same mainframe or when cataloging a file at a linked mainframe, SAVEPF should be used. (Refer to Using Permanent Files at Other Mainframes.)

ACCESSING THE INITIAL CYCLE OF A FILE

Permanent files may be accessed by using either the ATTACH or GETPF statements. File access methods are not determined by how the file was cataloged; that is, a file saved directly through the use of a CATALOG statement may be accessed either directly through an ATTACH request or indirectly through a GETPF request. The same is true for a file saved through SAVEPF.

ATTACH

If no passwords are specified when the initial cycle of a permanent file is cataloged, any user who knows the permanent file name (pfn), and creator identification (if specified) may access the file. All permissions to modify, purge or otherwise alter the file can be granted the attaching job when no passwords are specified. However, default values for the ATTACH statement specify that the file is attached to the job in multiple read mode, allowing read-only access unless the no-multiread parameter, MR=0, is included in the ATTACH statement.† Any number of users may simultaneously attach any number of cycles of a permanent file for read-only access. Multiread access is also possible when passwords for extend, modify, and control functions have been defined for the file, but no job currently has the file attached in any mode other than read mode. Any user who attaches a cycle of a permanent file for a purpose other than reading it must obtain exclusive access to that cycle. If a cycle is already in exclusive use when a request for the cycle is made, the requesting job waits until the job with exclusive access releases that cycle.

†Default multiread on ATTACH is an installation option; check with a site analyst if in doubt as to how ATTACH works at your site.

When an initial cycle of permanent file is cataloged with no passwords, the user desiring other than read access should specify MR=0 on his ATTACH statement. The file is then attached to his job with all permissions granted. He has unique access to that cycle of the file. The following control statement accomplishes this function. If MR=0 is omitted, the file is attached in multiread mode only.

```
ATTACH(lfn, pfn, ID=id, MR=0)
```

GETPF

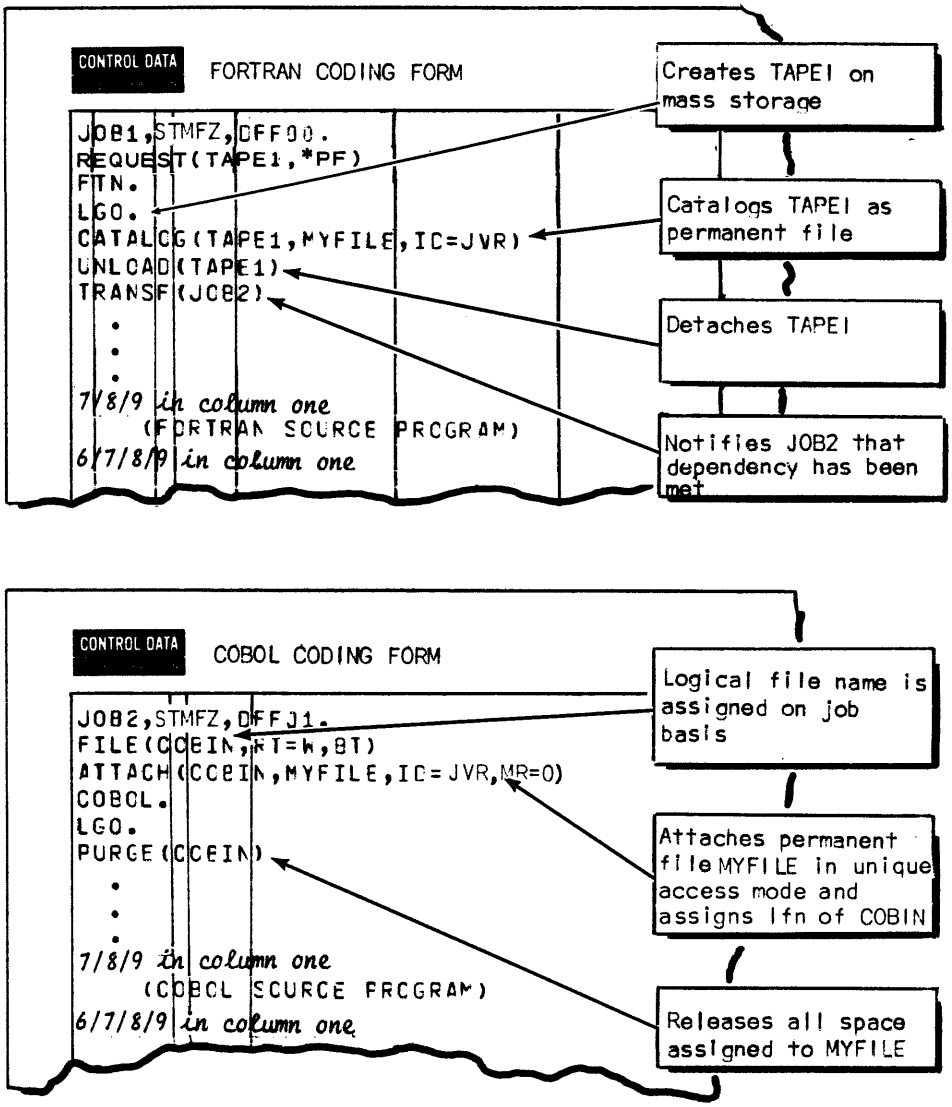
Since the GETPF statement creates a copy of the permanent file and attaches it to the job, multiread access is implicit. Any number of users may obtain their own copies of a file through the GETPF statement if they know the pfn and creator ID (and possibly the read password); the MR parameter is not relevant. However, if another user already has the desired cycle of the file exclusively attached to his job, that is, through the use of defined modify, control or extend passwords or the MR=0 parameter on an ATTACH statement, the GETPF request must wait until the job with exclusive access releases the file.

The copy obtained through GETPF becomes in effect a scratch file attached to the job. Thus, each user may modify his copy in any way he wants without affecting the access allowed to any other job. This is also the case when passwords have been defined for a file. You need only have the equivalent of read permission to obtain a copy of the desired file through GETPF, and then you may modify your own copy as you choose even though you do not have modify permission on the original file. However, since the file copy is effectively a scratch file, it is destroyed at job termination or when the file is returned or unloaded unless you take steps to preserve it through a CATALOG or SAVEPF statement.

As with SAVEPF, GETPF should be used carefully when dealing with large files locally, since two copies of the same file exist after the GETPF request is processed, and can unnecessarily tie up system mass storage. GETPF can be very useful, however, when you wish to modify a file but do not want to risk destroying it in the process, or wish only to modify it temporarily. A copy can be obtained through GETPF and modified without harm to the original. The following control statement will attach a copy of a file cataloged with no passwords.

```
GETPF(lfn, pfn, ID=id)
```

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.

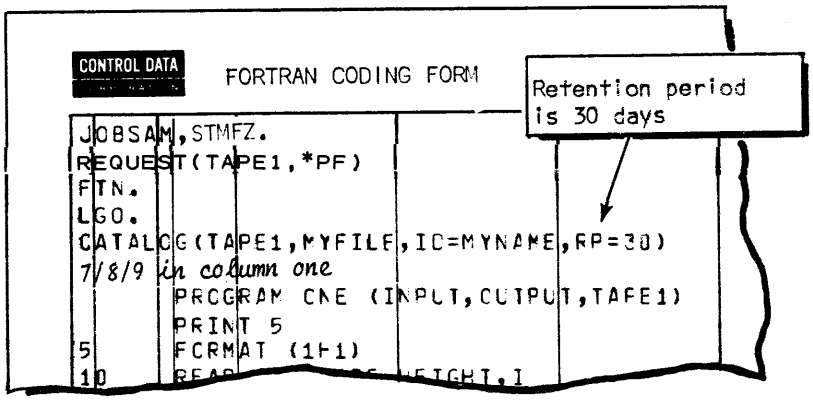


Example 8-1. Permanent File With No Password Requirements

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.



Example 8-2. Using the Retention Parameter

USING THE CYCLE NUMBER PARAMETER

When cataloging or attaching a permanent file, the user has the option of supplying a cycle number in the form CY=n, where n is a decimal value from 1 to 999. A number is seldom specified on a CATALOG or SAVEPF because when the first cycle of a file is cataloged, the default for cycle number is 1 and when subsequent cycles are cataloged, the cycle number defaults to one higher than the current highest cycle number for the permanent file.

If an existing cycle number (n) is specified on a CATALOG or SAVEPF, cycle n+1 is cataloged provided that cycle n+1 does not already exist or does not exceed 999. If the cycle n+1 exists or the cycle number exceeds 999, job step abort occurs for the control statement request.

If the cycle number is unspecified on an ATTACH or GETPF, the cycle with the highest number is attached unless the LC parameter is specified.

For certain applications such as purging a file, use of the LC (lowest cycle number) parameter is preferable to use of the CY parameter on an ATTACH. The LC=n parameter, where n is nonzero, causes the lowest-numbered cycle to be attached. If both LC and CY are specified, LC is ignored.

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.

The CATALOG or SAVEPF 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.

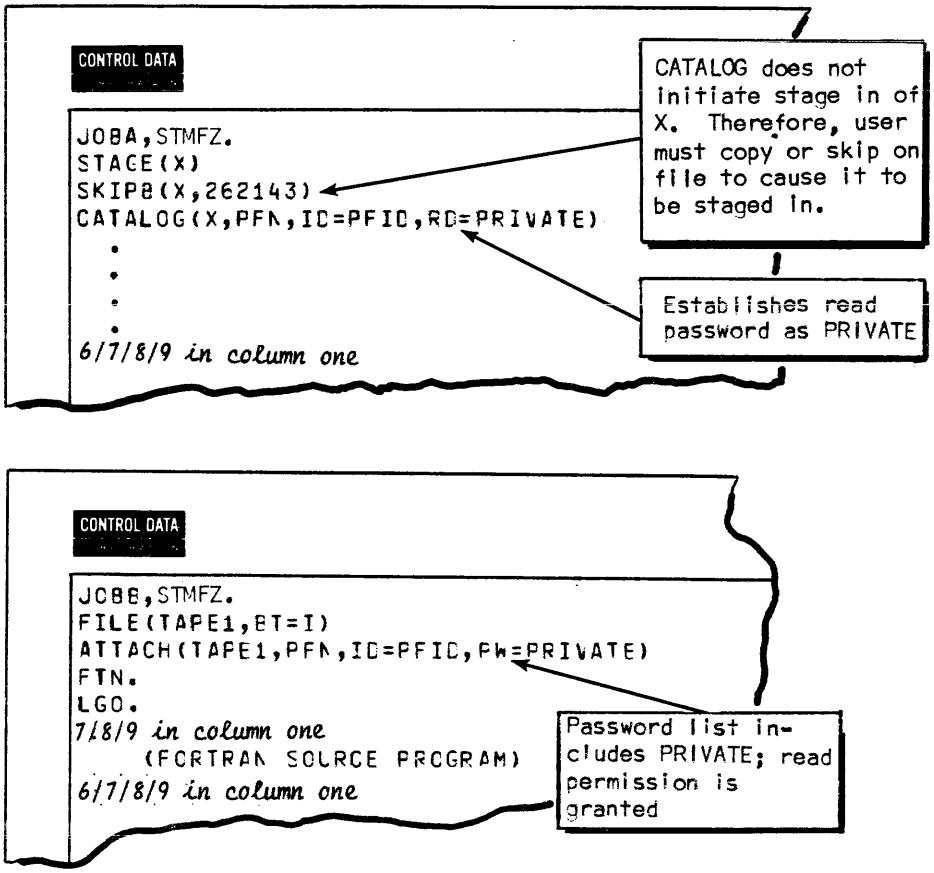
```
CATALOG or SAVEPF(lfn,pfn,ID=id,...,RD=rdpw,EX=expw,MD=mdpw,CN=dnpw,TK=tkpw)
```

USING THE READ PASSWORD

The read password (RD=rdpw, where rdpw is 1 to 9 characters), if specified when cataloging 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 or GETPF statement, as follows:

```
ATTACH or GETPF(lfn,pfn,ID=id,...,PW=rdpw)
```

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.



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 permanently increase the size of any cycle of a permanent file. It is applicable only to jobs requesting direct access to the file through an ATTACH statement, as the copy obtained through GETPF may be extended or otherwise modified as the user chooses. Changes made through GETPF, however, are not permanent unless the user takes steps to make them so.

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, even when accessing the file through ATTACH, 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)
```

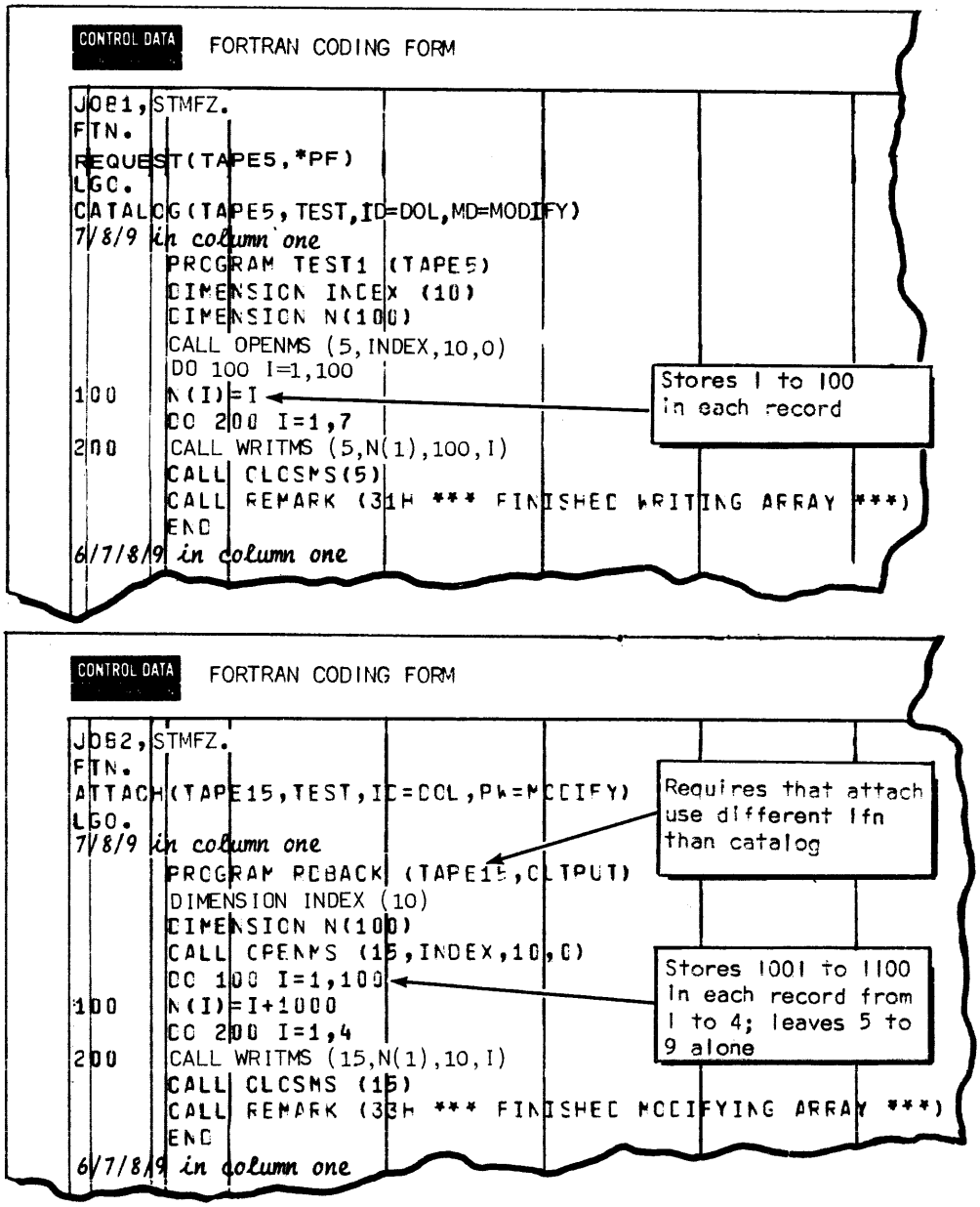
When a user attaches a file and obtains extend permission, he obtains exclusive access to that cycle of the permanent file.

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 permanently rewrite any portion of the existing data on any cycle of a permanent file. As in the case of the extend password, the modify password is not used when obtaining only a copy of, not direct access to, 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.



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.

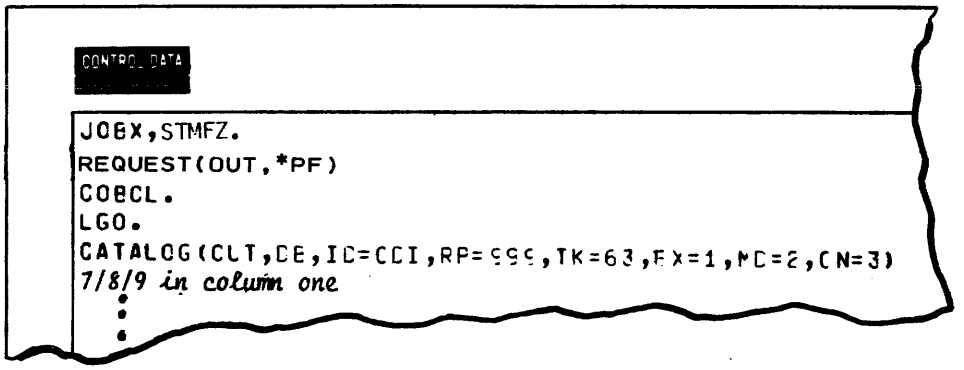
USING THE TURNKEY PASSWORD

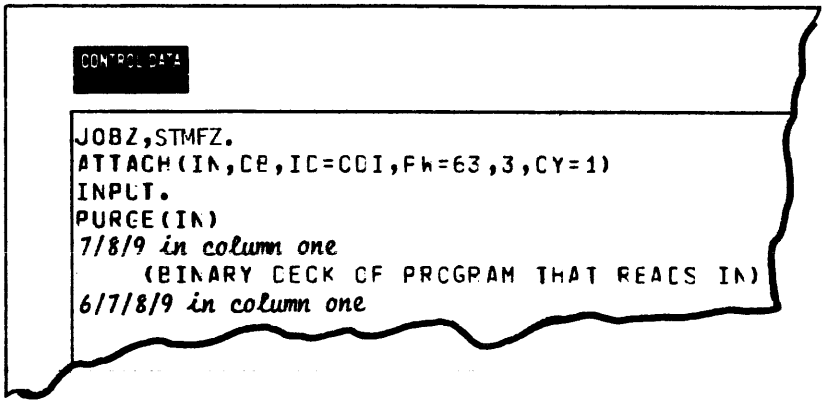
The turnkey password (TK=tkpw, where tkpw is 1 to 9 characters) restricts no specific permissions; however, if a turnkey password is defined on the catalog of the orignal cycle of a permanent file, only the user who knows the password can exercise permissions specified or defaulted for any cycle of the permanent file. The turnkey password is applicable to all five permanent file access control statements; CATALOG, SAVEPF, ATTACH, GETPF and PURGE.

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.





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. Remember, when obtaining indirect access to a cycle of a permanent file through GETPF, the read password and/or the turnkey password are the only ones applicable.

```
permanent file control statement(lfn,pfn,ID=id,...,PW=password1,password2,
...,passwordn)
```

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)
    or
SAVEPF
```

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
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.

USING THE EXCEPT READ PASSWORD

When cataloging the initial cycle of a permanent file, use of a single parameter, XR=password, is allowed in place of separate CN, EX, and MD parameters. XR defines a single password for control, extend, and modify unless they are explicitly and separately defined through CN, EX, and MD. The except read password is not applicable for GETPF requests.

```
CATALOG(lfn,pfn,ID=id,...,XR=password)
    or
SAVEPF
```

If the MD, EX, and CN parameters in the previous example were all replaced with the XR parameter, the statement would look like the following:

```
CATALOG(ALPHA, MF, ID=ME, XR=MEC, TK=TURNKEY)
```

Again, the only permission for which no password is defined is read permission. Thus, the following password lists on an ATTACH might be as follows:

Password list	Permission Granted
PW=TURNKEY	Read; multiread is allowed
PW=TURNKEY, MEC	Read, extend, modify, and control; multiread access is not allowed
PW=MEC	None; turnkey password is missing

USING THE MULTIREAD ACCESS PARAMETER

Normally, when a file is cataloged with passwords, it remains attached to the job performing the catalog with all permissions granted. This means that other jobs cannot obtain access to the file, even for reading, until the file is released by the job. By using the MR=1 multiread option, a user can specify that following the catalog that the file be attached to the job with read permission only, thus allowing immediate access to the file by other jobs as soon as the file is listed in the permanent file directory.

```
CATALOG(ALPHA, MF, ID=ME, XR=MEC, TK=TURNKEY, MR=1)
    or
SAVEPF
```

Following processing of this control statement, file ALPHA remains attached to the job with read permission only. If the MR=1 parameter had been omitted the file would remain attached with control, modify, extend, and read permissions, thus prohibiting multiread access to other jobs.

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 or SAVEPF when adding a new cycle to an existing permanent file.

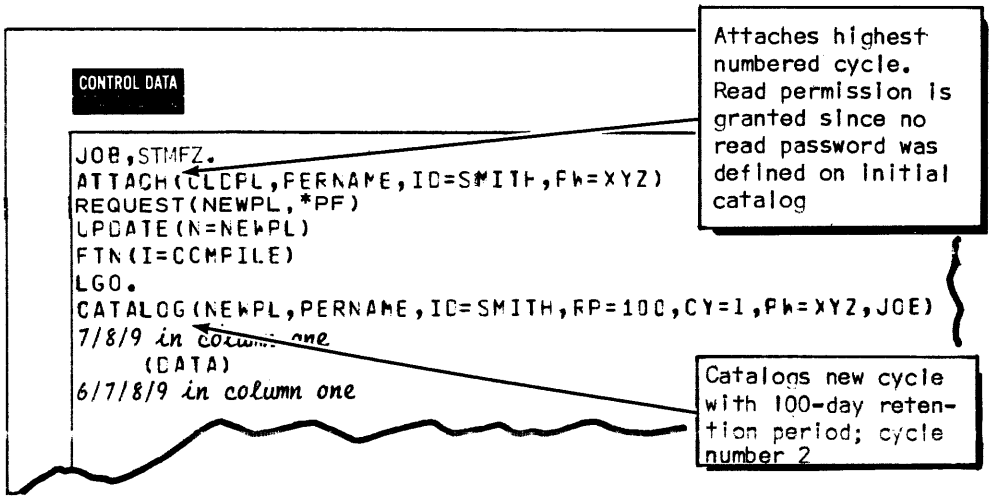
```
CATALOG(lfn, pfn, ID=id, RP=rp, CY=n, PW=password1, password2)
    or
SAVEPF
```

The parameters for lfn, pfn, RP, and ID have the same meanings as in the previous descriptions. The default for the cycle number is the highest numbered cycle plus one for a new cycle catalog.

A convenient technique in cataloging a file (if the same job is used to do the initial as well as subsequent cataloging) is to specify the permissions on the catalog statement in both forms. For example:

```
CATALOG(lfn, pfn, CN=cn, EX=ex, MD=md, TK=tk, PW=cn, tk)
```

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. A cycle numbered one is currently in the permanent file directory so the file is cataloged with cycle number two.



Example 8-6. Cataloging a New Cycle

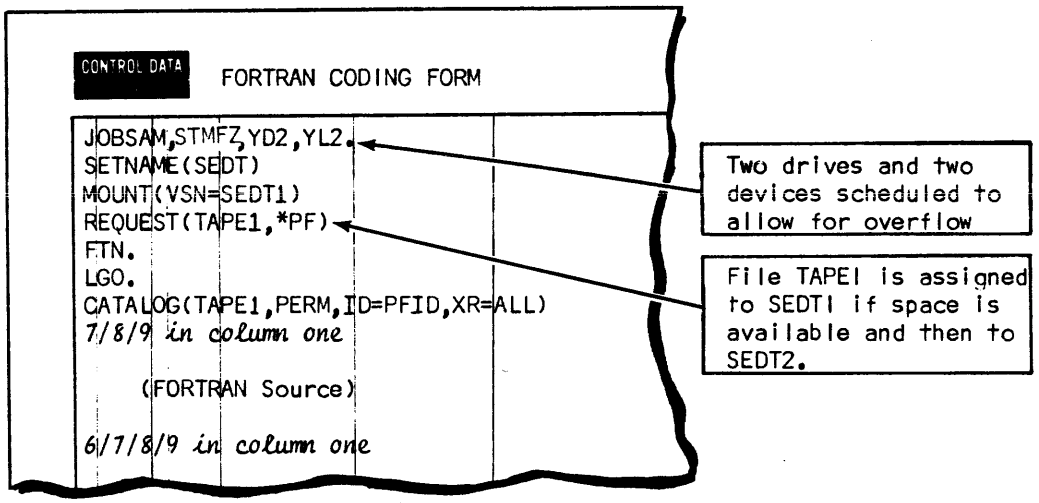
CATALOGING A FILE ON A REMOVABLE SET

So far, we have considered only using files on the default (system) set. Now, consider the possibility that you have your own removable packs and wish to maintain your permanent files on them.

First, you must create the removable set with at least one set member defined as a permanent file device. This procedure is described in Section 7. In Example 8-7, we assume that a removable set named SEDT exists and that the set contains two members; master device SEDT1 and another member identified by the vsn SEDT2. Both are permanent file devices.

In example 8-7, the SETNAME control statement changes the default setname for the job to SEDT. The MOUNT mounts the master device for the job. The REQUEST statement assigns file TAPE1 to the set by default; both vsn's SEDT1 and SEDT2 can be used for the file. When the file is cataloged, the permanent file directory on the master device is updated with information concerning the location of the file.

CATALOG must be used to create permanent files on a removable set; SAVEPF can only be used in conjunction with the system set.



Example 8-7. Cataloging File on Removable Set

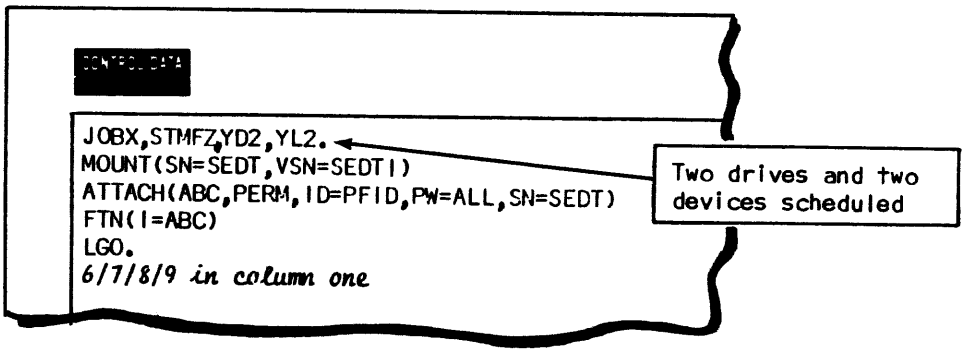
ATTACHING A FILE ON A REMOVABLE SET

Suppose that you wish access to a file on a removable set. This means that you must identify the set on which the file resides and ensure that the master device for the set is explicitly mounted.

There are two ways to identify the set. First, you can change the default setname for the job through a SETNAME control statement (Section 7) or second, you can supply the setname through the SN=setname parameter on an ATTACH. In either case, the master device must be mounted before the ATTACH.

The GETPF control statement cannot be used to access a file which resides on a removable device.

Example 8-8 illustrates attaching a permanent file on a removable device. If the permanent file is on a set member other than the master device, the system implicitly mounts the required device.



Example 8-8. Attaching a Permanent File from a Removable Set

ALTERING THE SIZE OF PERMANENT FILES

While a user can make any changes he desires on his copy of a cycle of a permanent file obtained through a GETPF request, the changes affect only his copy, and not the file itself. The only way the file itself can be altered is through an ATTACH request, where the proper permissions are granted by default or through defined passwords.

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. ALTER and EXTEND move the file's end-of-information. 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. When the control statement is processed, the EOI is moved to the current position of the file.

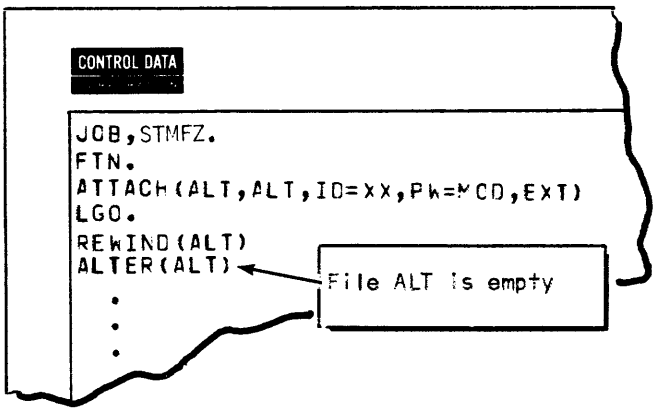
```
ALTER(lfn)
```

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)
```

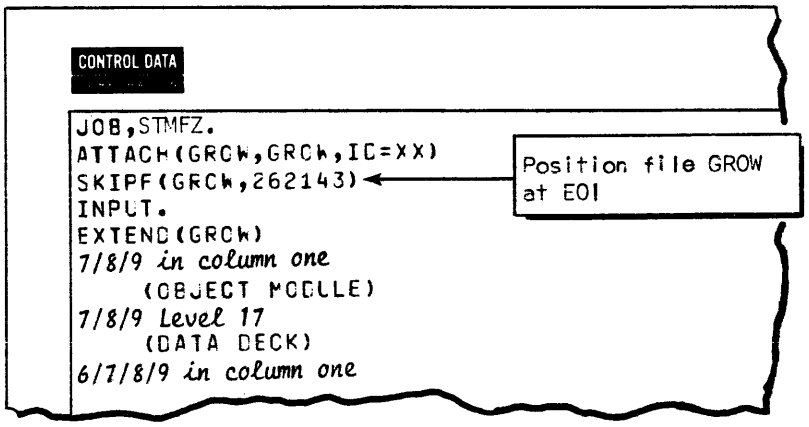
Because the attached file is used as a logical file, the only parameter required following the ALTER or EXTEND statement is the logical file name used when the file was attached. If a file is to be extended only, it must be positioned at end of information before any writing occurs. If it is a blocked file, it must be closed.

Example 8-9 illustrates how ALTER can be used to create an empty cycle of a permanent file.



Example 8-9. Using the ALTER Control Statement

In Example 8-10 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.



Example 8-10. Using the EXTEND Control Statement

PURGING PERMANENT FILES

If you have control permission on a file, you have the authority to purge any cycle of that file from the permanent file directory. Two forms of the PURGE control statement are available to a user for deleting a cycle of a permanent file. In the first form, an ATTACH statement must precede the PURGE statement; lfn is the only parameter required on the PURGE statement. All other parameters and passwords are specified on the ATTACH. The control statement has the following format.

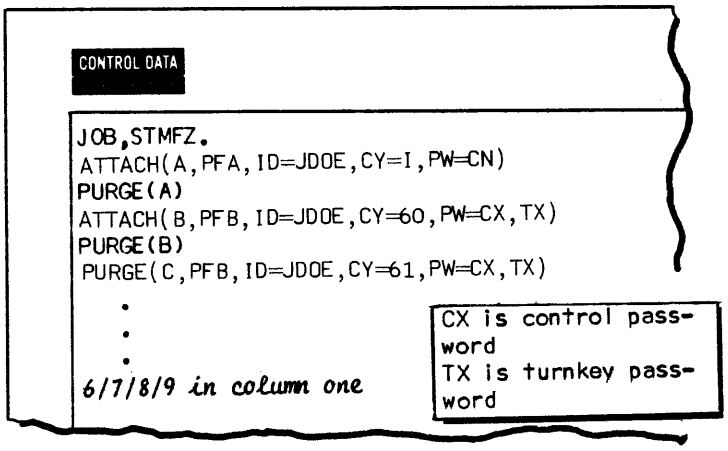
```
PURGE(lfn)
```

In the second form, no ATTACH is required; all parameters and passwords can be specified on the PURGE statement. If the cycle number is omitted, the highest numbered cycle is purged. The control statement format is shown below.

```
PURGE(lfn,pfn,ID=id,CY=n,PW=password...)
```

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 8-11, a user with the ID of JDOE decides to clean out three obsolete cycles from his permanent files to make room for new files. The first two cycles are purged after first attaching them; the third is purged with no prior attach.



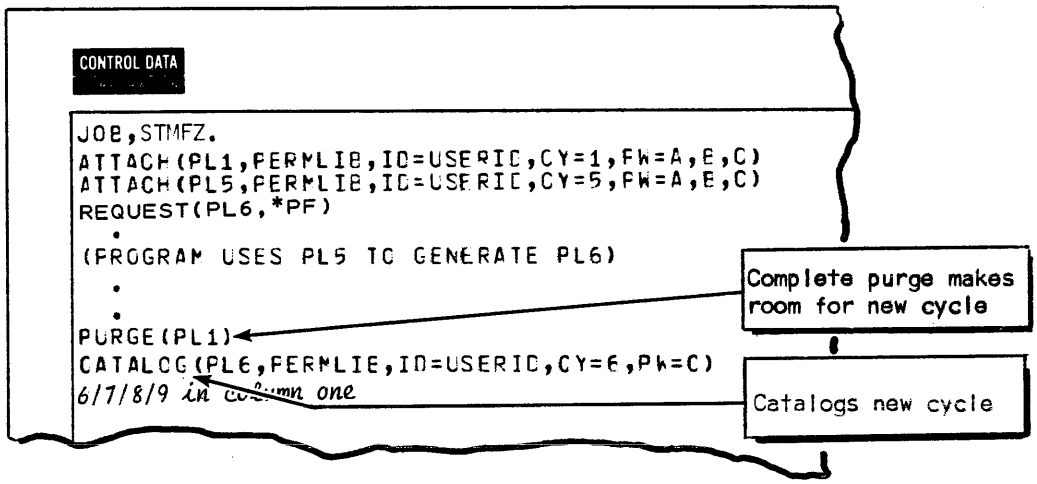
Example 8-11. Purging a Cycle of a Permanent File

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 8-11 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. A purge of a file without a prior ATTACH produces no local copy.

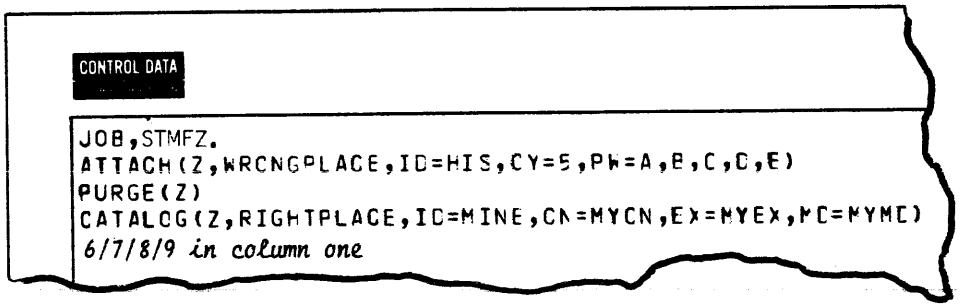
Example 8-12 shows a job that performs a complete purge of a cycle of a permanent file to make room for a new cycle.



Example 8-12. 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 8-13, 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.



Example 8-13. 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 PERMANENT FILES AT OTHER MAINFRAMES

So far, you have learned how to use permanent files on system mass storage on your local SCOPE 2 Operating System. Suppose your site has a linked NOS/BE or SCOPE 2 mainframe 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 another mainframe. SCOPE 2 allows both of these operations. You cannot, however, modify or extend a file at the other mainframe. These restrictions arise from the fact that when you attach a cycle of a permanent file residing at another mainframe, you do not obtain direct access to that file. Instead, the first time your attached file is opened, the other mainframe sends a working copy of the cycle requested over to the local mass storage where it resembles a temporary file more than it does a permanent file. Similarly, a catalog of a permanent file at another mainframe results in a copy of your file on the local mass storage being sent over to the other mainframe as soon as the CATALOG or SAVEPF command is issued.

Before attempting to use a permanent file at a linked NOS/BE mainframe, read the section on permanent files in the NOS/BE Reference Manual.

Restrictions on use of NOS/BE permanent files are:

- You cannot dispose or stage a file either attached from the station or cataloged at the station.
- The station permanent file is transferred to SCOPE 2 when it is opened by the job. If the operation to be performed does not open the file, the user must ensure that the staging of an attached file has taken place before his job refers to the file. One technique is to skip backward on the file (Example 8-15).
- The SCOPE 2 file organization library (LB) cannot be used under NOS/BE You can catalog files having this organization at the station but they cannot be accessed by a NOS/BE job.
- NOS/BE requires blocking for sequential (SQ) files.
- NOS/BE permanent files using SIS, SDA, and IORANDM are not interchangeable.
- If you do not supply a FILE statement for the attached NOS/BE permanent file, it is assumed by default to be record type W, unblocked.
- Files to be cataloged at the linked NOS/BE mainframe from the SCOPE 2 Operating System must be unlabeled.
- Although exceptions exist, relocatable binary files are, in general, not interchangeable.
- SP is required on the job card.

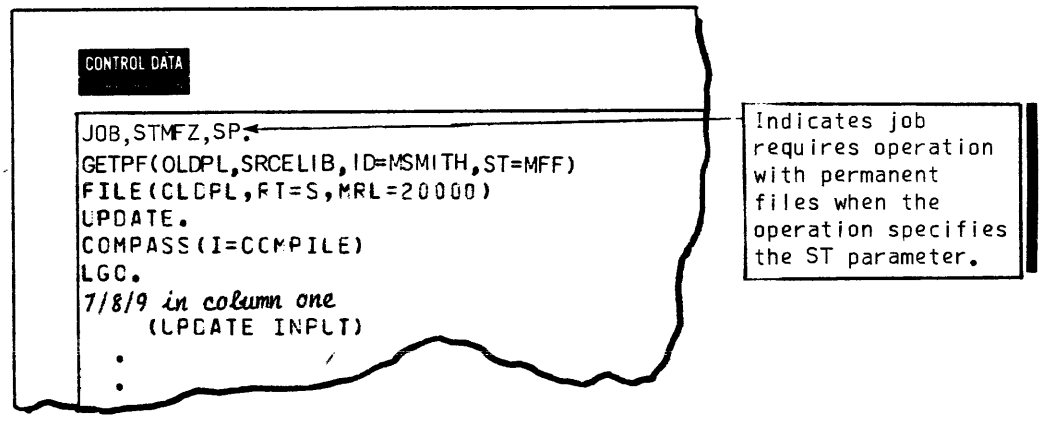
USING THE STATION PARAMETER

Use the GETPF control statement with the station parameter (ST=ggg, where ggg is a logical or physical mainframe id) to attach a copy of a file at another mainframe, and the SAVEPF statement with the ST parameter to save a copy of a file attached to your job at

the mainframe identified in the ST parameter. You can catalog initial cycles, or new cycles of existing files. Files must be attached from or saved on system mass storage at the linked mainframe. Removable sets at a linked mainframe cannot be accessed by the host system. When the ST parameter identifies a linked mainframe other than another SCOPE 2 Operating System, parameters generally are subject to the requirements of the NOS/BE permanent file manager.

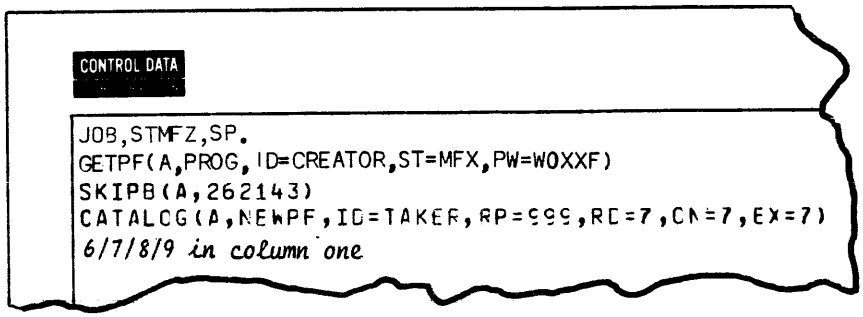
Use the ST=ggg parameter on the purge-with-no-attach form of the PURGE control statement to purge a cycle of a permanent file at another mainframe. All parameters and passwords must be specified on the PURGE statement. The specified cycle is removed from the permanent file directory at the mainframe identified by ggg.

Example 8-14 shows how to acquire an UPDATE old program library that exists as a permanent file at a linked mainframe for assembly and execution on the host mainframe. 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.



Example 8-14. Attaching a CDC CYBER Station Permanent File

In Example 8-15, a user wishes to obtain a copy of a cycle of a permanent file cataloged at another SCOPE 2 Operating System (ST=MFX) and recatalog it at the host mainframe so that the information exists under both permanent file systems. In this case, the user must initiate the transfer of the file from the linked mainframe through a copy or skip operation since the CATALOG statement does not open the file. The transfer to the host mainframe occurs when the user first opens the file.



Example 8-15. Cataloging a Permanent File Attached from a Linked 7600

Example 8-16 shows an UPDATE old program library being created under SCOPE 2 and cataloged under NOS/BE at the CDC CYBER station (identified as MFF). This cycle is then attached using a job that can be processed under either system.

CONTROL DATA

```
JOB1,STMFZ,SP.
FILE(PL,RT=S)
UPDATE(N=PL,W,L=1234)
SAVEPF(PL,PERMUPLIB,ID=JONES,ST=MFF,TK=XX)
7/8/9 in column one
        (UPDATE CREATION DECKS)
6/7/8/9 in column one
```

CONTROL DATA

```
JOB,STMFZ,SP.
FILE(OLDPL,RT=S)
FILE(NEWPL,RT=S)
GETPF(OLDPL,PERMUPLIB,ST=MFF,ID=JONES,PW=XX)
UPDATE(P=OLDPL,N=NEWPL,L=1234,H)
SAVEPF(NEWPL,PERMUPLIB,ST=MFF,ID=JONES,CY=2,PW=XX)
7/8/9 in column one
    (UPDATE CORRECTION DECK)
6/7/8/9 in column one
```

Example 8-16. Maintaining UPDATE OLDPL at a Linked NOS/BE Mainframe

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 card reader, 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 transfers of data, which are referred to as spooling.

CARD READER INPUT

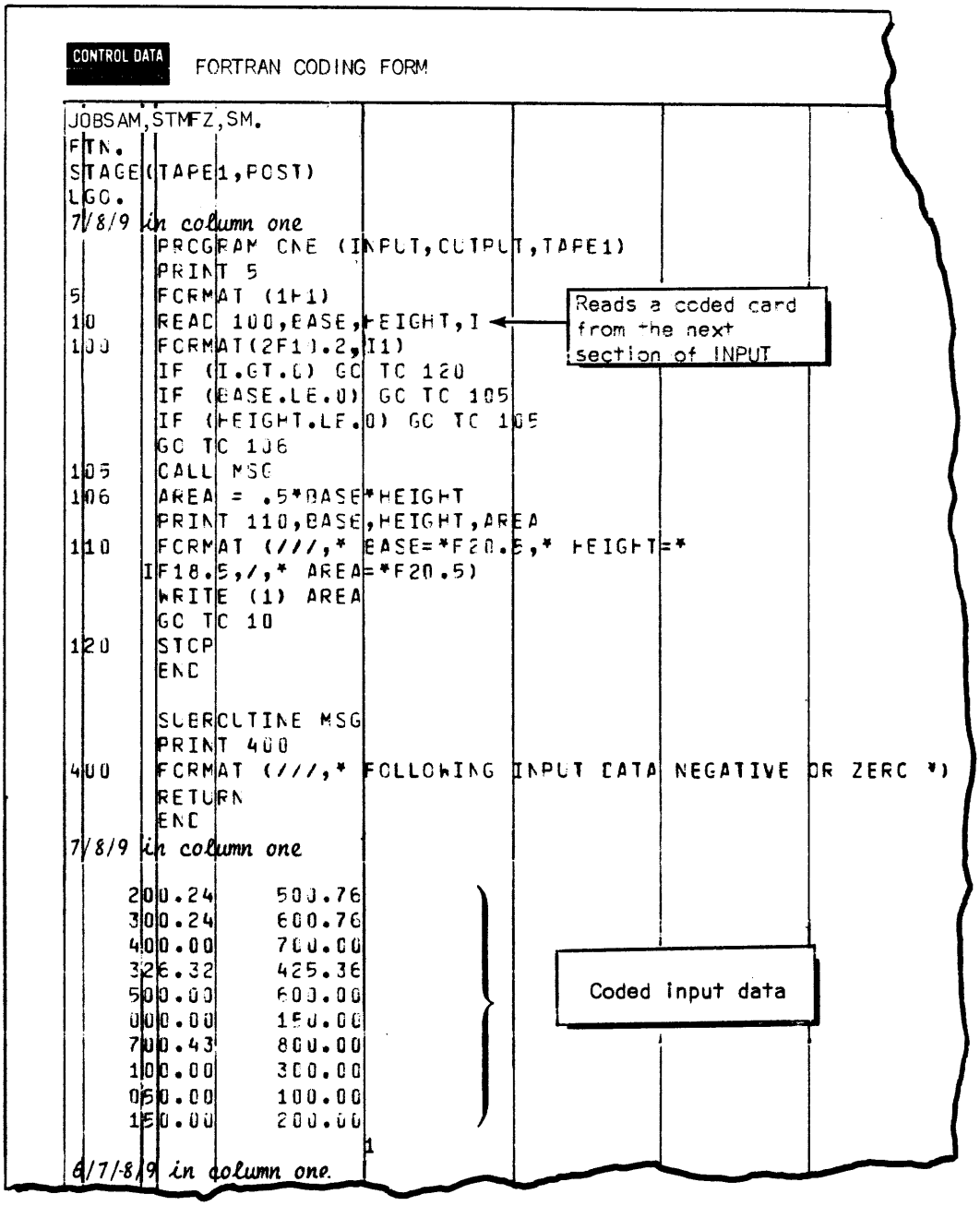
The INPUT file contains the portion of your job deck that most concerns you. Each section or partition in your job deck becomes a section or partition of INPUT. A section consists of sequential, unblocked W records that vary in size according to whether your deck section is coded cards, formatted binary 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 case there is no EOS preceding the EOP.

The first FORTRAN language READ fn,iolist statement automatically refers to the next record on INPUT (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.

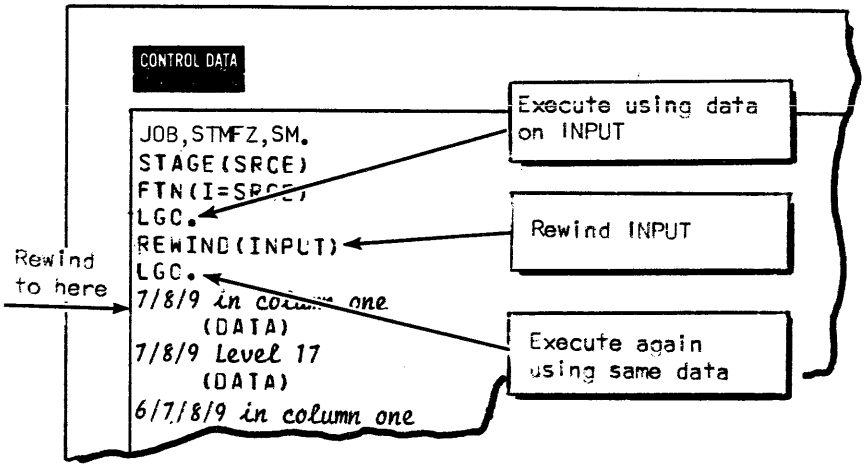
The INPUT file cannot be returned or unloaded. Although it is not protected from being redefined as some other file organization or record type, you should avoid redefining it. You can also change the maximum record length (MRL) between job steps.



Example 9-1. Reading Cards from INPUT

Remember, that unlike SCOPE 3.4 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.

The loader does not rewind INPUT before loading from it.



Example 9-2. Rewinding INPUT

CODED PUNCHED CARD INPUT

Not all 80-column punched cards follow 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 key-punching 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 CDC CYBER station.

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).

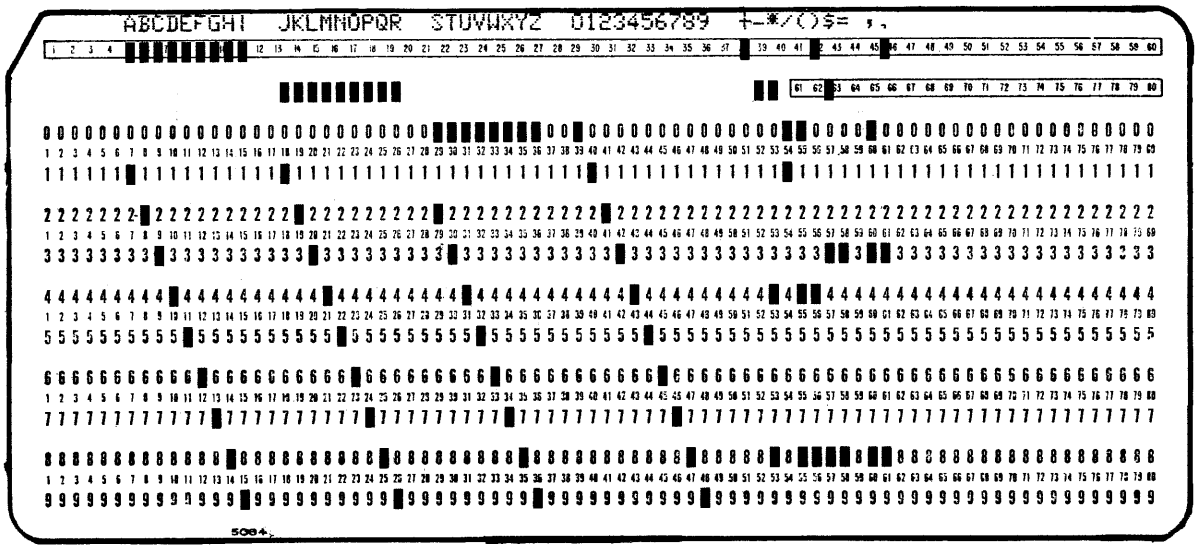


Figure 9-1. Hollerith (026) Coded Card

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.

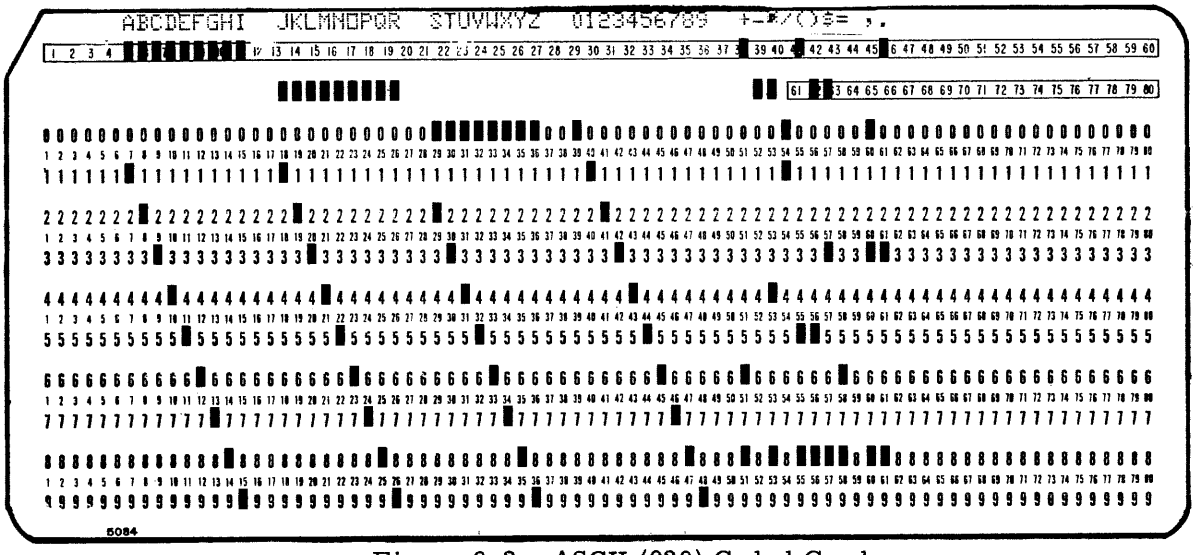
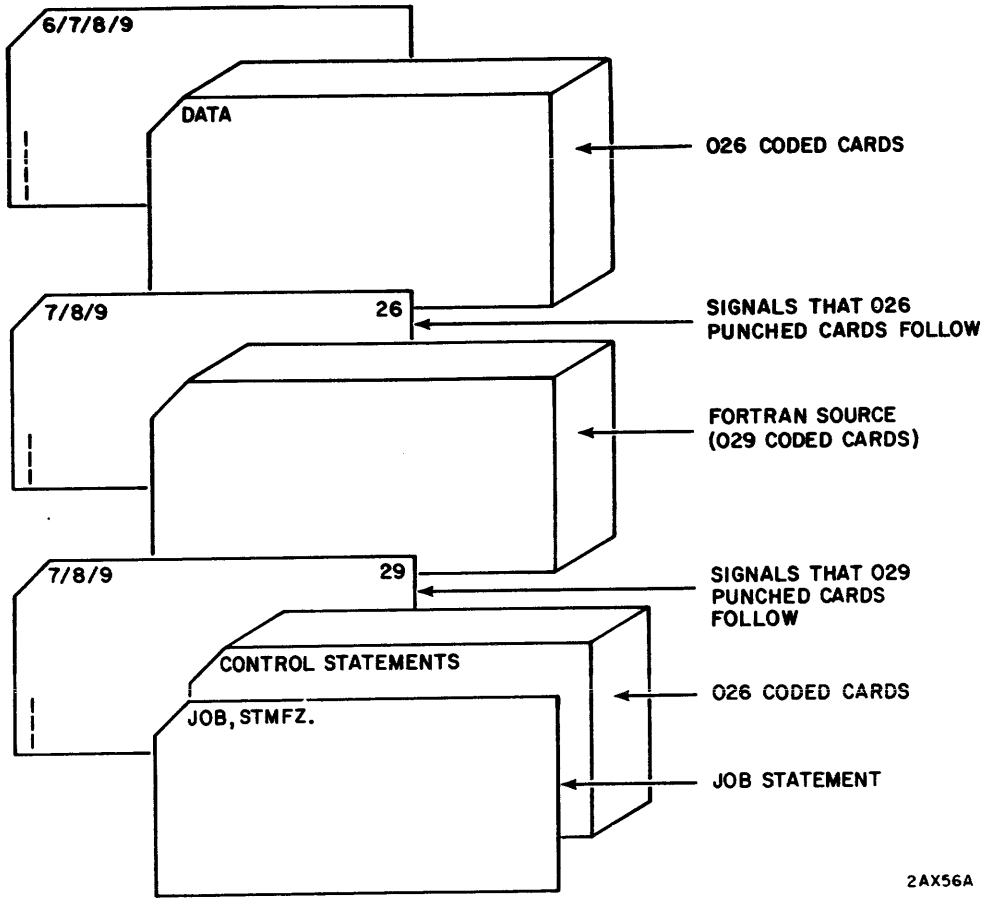


Figure 9-2. ASCII (029) Coded Card

† The 7611-1 Station recognizes the code on the job statement only.

Example 9-3 illustrates a job containing a FORTRAN source deck punched in the 029 punch format.



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, use care in arranging the data; there is no way to recreate them.

CDC CYBER STATION

A coded card from the CDC CYBER 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. However, a special test is made for the extra characters and they are removed when the record is sent to SCOPE 2 by the station. Thus, a coded card image has a minimum of 8 characters and a maximum of 80 characters (Figure 9-3). This procedure is required for the interface between the two systems, which use a zero byte terminator for a record.

7611-1 STATION

A coded card from the 7611-1 Station has all of the trailing blanks removed. If the card contains 1 or 80 characters of data, the record is 1 or 80 characters, respectively. Figure 9-3 illustrates coded card images on INPUT.

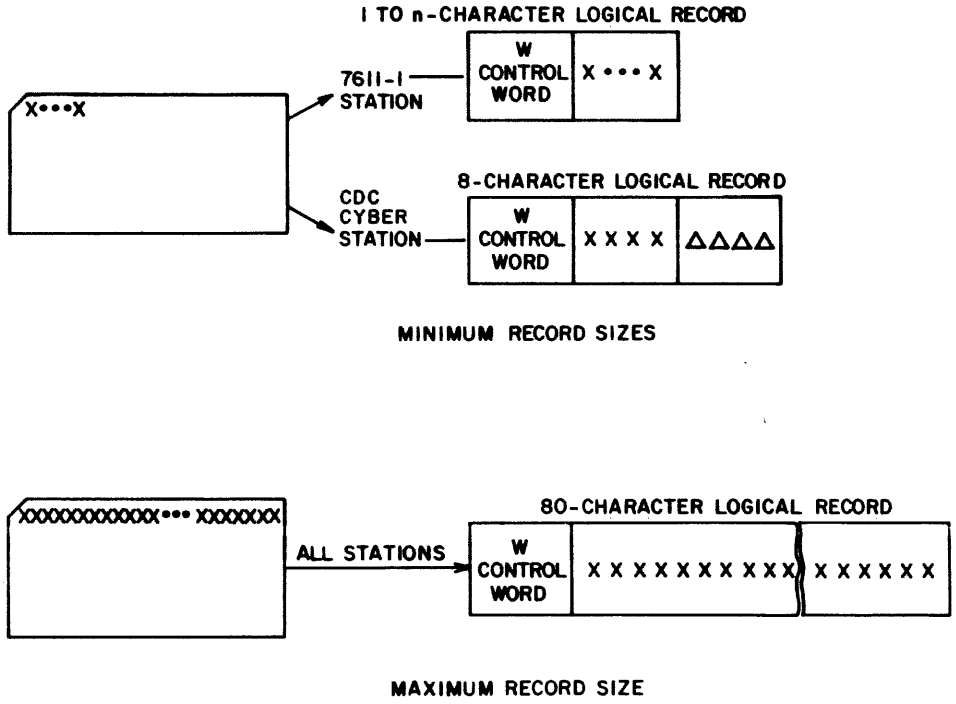


Figure 9-3. Coded Card Images as W Records on INPUT

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 CDC CYBER 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.

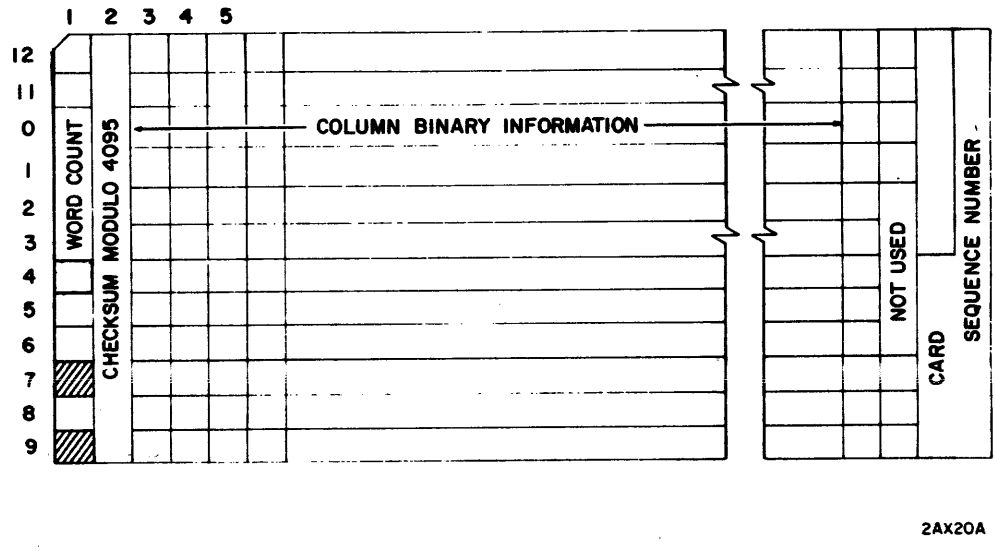


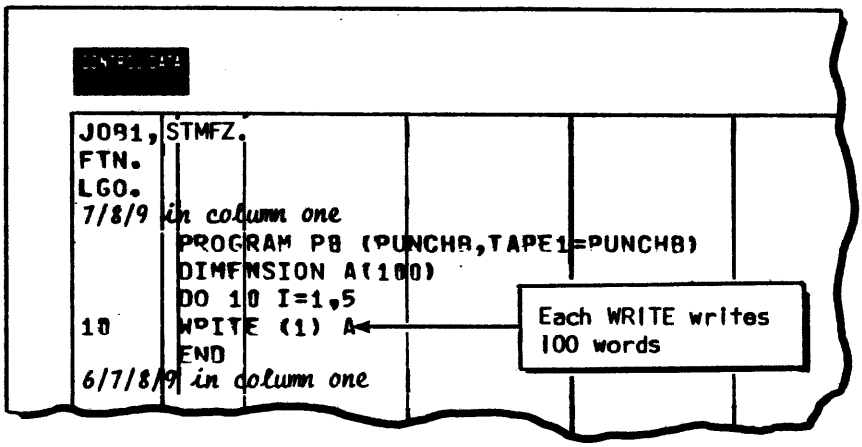
Figure 9-4. SCOPE Binary Card

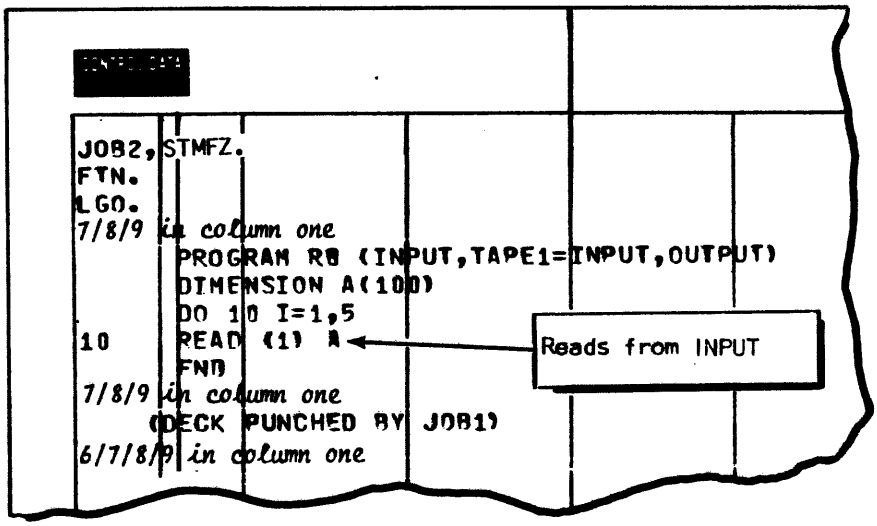
All of the decks to be loaded by the loader, that is, object modules and program image modules consist of SCOPE binary cards. For a more complete description of the loader tables that comprise the object module or program image module, refer to the Loader Reference Manual. The loader ignores the W record 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 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 CDC CYBER station, you receive 630 6-bit characters of data.

This does not promote a one-to-one correspondence between output statements and input statements. (See also, Punched Card Output, page 9-19.)





Example 9-4. FORTRAN Binary Input (CDC CYBER 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 (refer to 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.

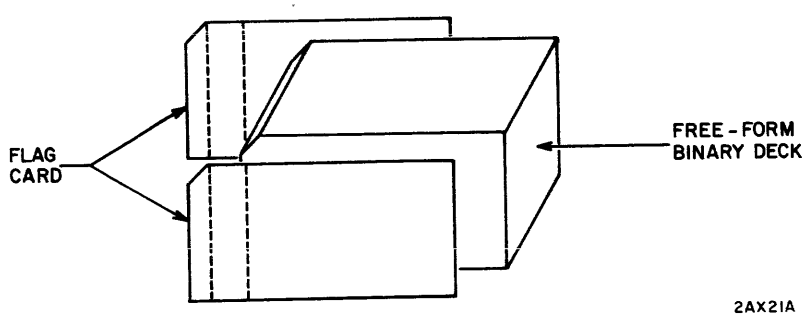


Figure 9-5. Flag Cards to Delimit Free-Form Binary Deck

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.

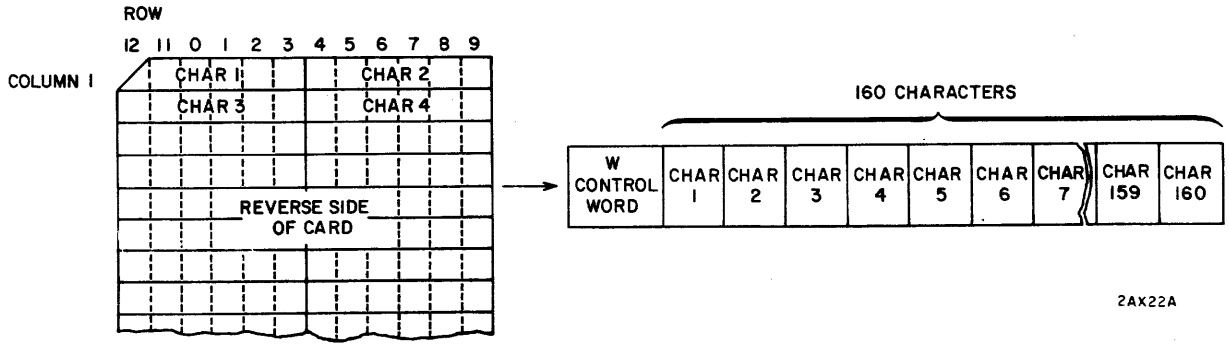


Figure 9-6. Free-Form Binary Card Translation

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, one card image, or 160 characters of data.

END-OF-SECTION LEVEL NUMBERS

Job decks transferred to SCOPE 2 by a CDC CYBER station can originate either at a local card reader or at a card reader at a remote terminal.

Columns two and three of each EOS in the job contain the level number associated with the type of section following the EOS. These level numbers are:

Level	Type of Section
0	Coded (default)
1	SCOPE binary
2	Free-form binary

The station, when it transfers the file to SCOPE 2 from the station mass storage, performs a conversion based on the level number, and if level is 0, on a test of the first card in the section.

Condition	Type of Section	Conversion
Level 0		
First card loader PRFX (77_8) table	SCOPE binary	Section converted S to W, unblocked
First card not PRFX table	Coded	Each card converted Z to W, unblocked
Level 1	SCOPE binary	Section converted S to W, unblocked
Level 2	Free-form binary	Each card converted F to W, unblocked; FL=160

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 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. (Refer to 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 CDC CYBER station, if a line is greater than 137 characters, additional characters are continued on subsequent lines. Only the first character of the line is interpreted as a printer control character. For the 7611-1 Station, characters beyond the 137th are ignored.

   When printing at the CDC CYBER 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 or to save the file in some other way (perhaps by staging it or cataloging it).

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 line to be printed must be a carriage control character. The character itself is not printed but is part of the line image on mass storage. Buffer areas must be large enough to accommodate this character. When the line 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.

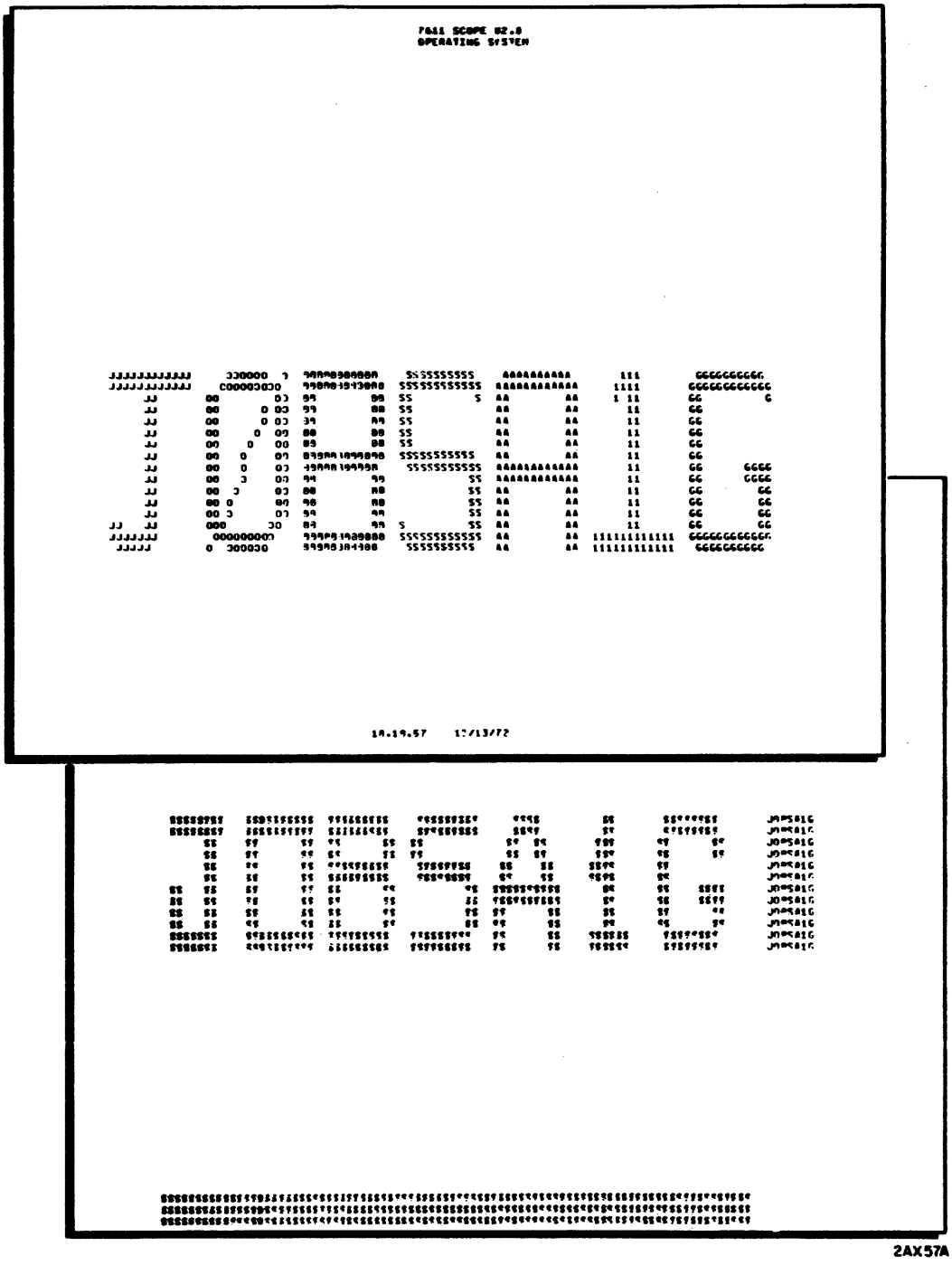


Figure 9-7. Printer Banner Pages (Two Styles)

TABLE 9-1. CARRIAGE CONTROL CHARACTERS

Character	Action Before Printing	Action After Printing
blank	Space 1	No space
A	Space 1	Eject to top of next page
B	Space 1	Skip to last line of page
C	Space 1	Skip to channel 6
D	Space 1	Skip to channel 5
E	Space 1	Skip to channel 4
F	Space 1	Skip to channel 3
G	Space 1	Skip to channel 2
H	Space 1	Skip to channel 1 (501) Skip to channel 11 (512)
I	Space 1	Skip to channel 7 (512)
J	Space 1	Skip to channel 8 (512)
K	Space 1	Skip to channel 9 (512)
L	Space 1	Skip to channel 10 (512)
M N O P	Space 1	No space
Q	No print; clear auto page eject	
R	No print; select auto page eject	
S	No print; clear eight vertical lines per inch (512)	
T	No print; select eight vertical lines per inch (512)	
U V W	Space 1	No space
X	Skip to channel 8 (512)	No space
Y	Skip to channel 9 (512)	No space
Z	Skip to channel 10 (512)	No space
PM	Display to the operator the rest of this line (up to display line size) and wait for operator action.	
0 (zero)	Space 2	No space
1	Eject to top of next page	No space
2	Skip to last line on page	No space
3	Skip to channel 6	No space
4	Skip to channel 5	No space
5	Skip to channel 4	No space
6	Skip to channel 3	No space
7	Skip to channel 2	No space
8	Skip to channel 1 (501) Skip to channel 11 (512)	No space No space
9	Skip to channel 7 (512)	No space
+	No space	No space
- (minus)	Space 3	No space

Issue the functions Q through T at the top of a page. S and T cause spacing to be different from the previous 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.

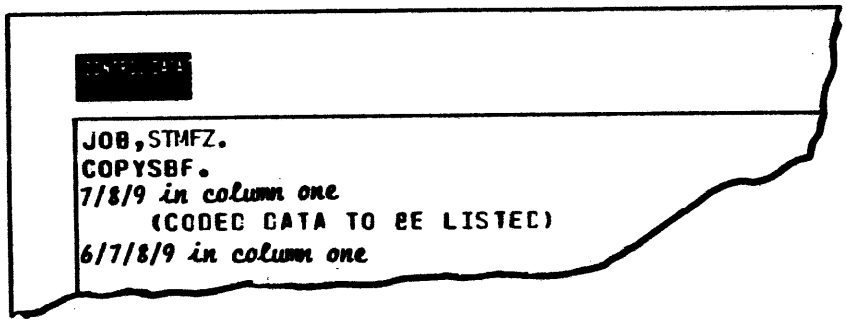
PRINTING A FILE THAT DOES NOT HAVE PRINTER CONTROL CHARACTERS

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     (lfn_in,lfn_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. Remember that the file must be unblocked W records to be disposed to a printer. 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.



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. Permanent files cannot be disposed.

Print blocked data by copying it to an unblocked file and then disposing the unblocked copy.

TYPE OF PRINTER

Designate the type of printer to use at the CDC CYBER station by using P1 instead of PR to indicate a 501 printer, by using P2 to indicate a 512 printer, or by using LR, LS, or LT to indicate a model of the 580 printer (LR is a 580-12 printer, LS is a 580-16 printer, and LT is a 580-20 printer). Your file will be placed in the queue for the printer of the requested type. The 7611-1 considers all six printer codes equivalent; all printing takes place on a 517 printer.

FORMS CONTROL

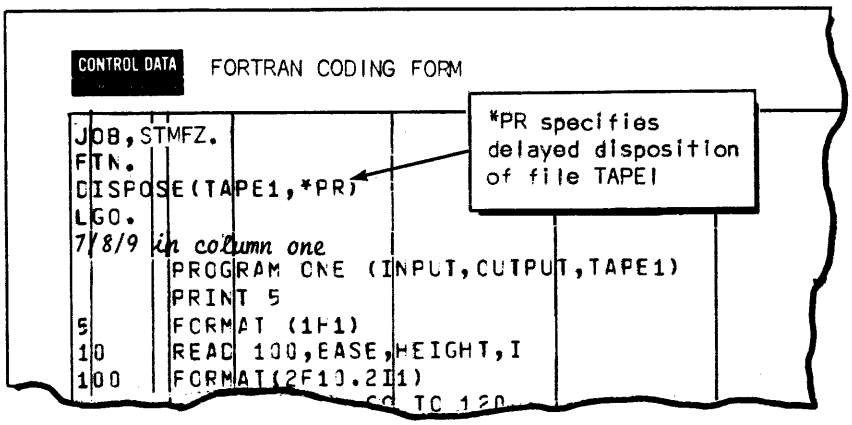
If you want the printing to take place on special forms, you can notify the operator by expanding the disposition code (dt) as follows:

```
DISPOSE(lfn, dt=Cyy)
```

dt is the printer type. 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.



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, dt, ST=gggttt)
```

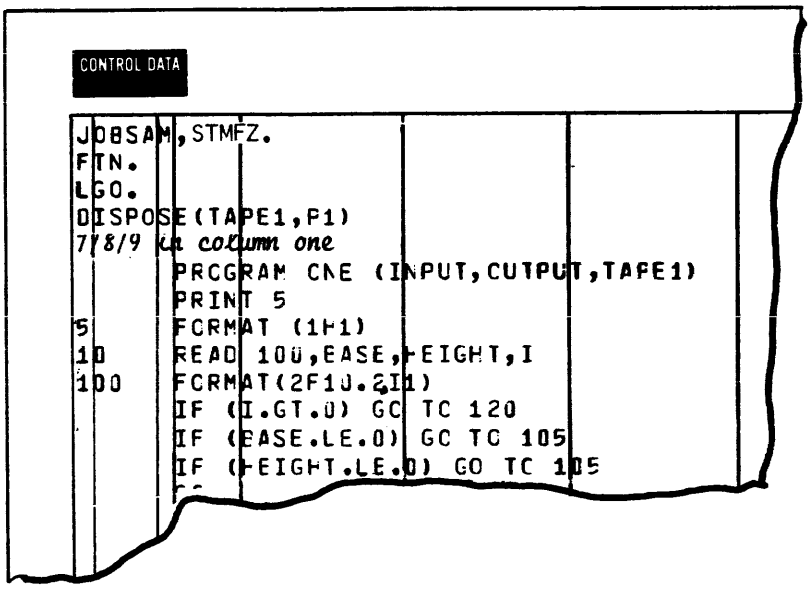
lfn is the logical file name and dt is the printer type. In this case, ggg is a 3-character station identifier assigned to the station when it was logged in. The terminal identifier ttt is optional, but when it is present, it identifies a terminal linked to the station designated by ggg. If ttt is present, the dt parameter cannot include forms control. A DISPOSE statement with no parameters except lfn causes the file to be closed/unloaded without being spooled to a station.

If you are routing to an INTERCOM terminal linked to a CDC CYBER station, you can designate the terminal by using Iyy on the ST parameter.

```
DISPOSE(lfn, dt, ST=gggIyy)          INTERCOM terminal
```

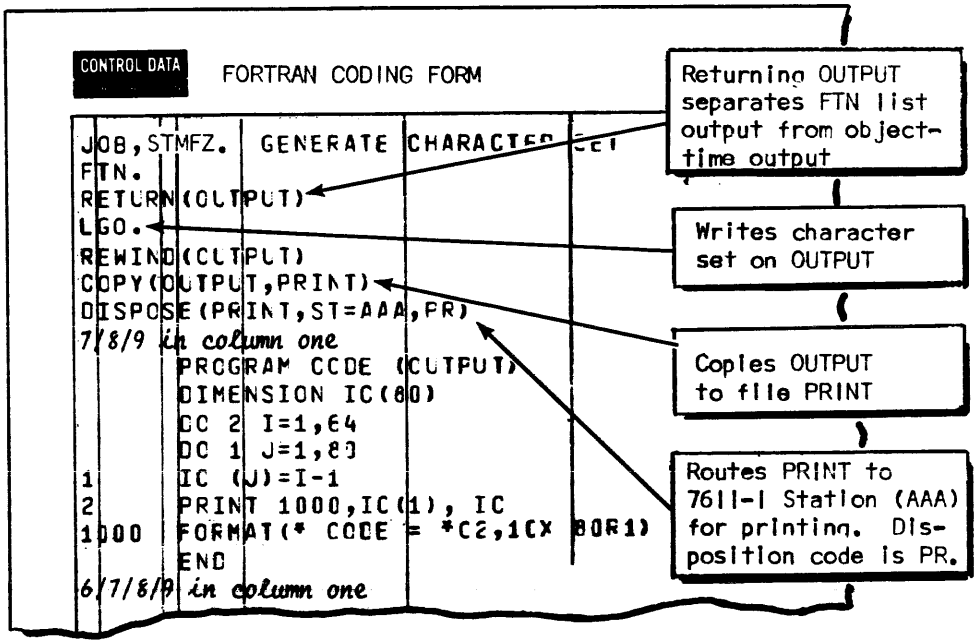
yy is a 2-character code defined by the installation as your terminal ID characters.

Example 9-7 illustrates a FORTRAN job that creates a print file named TAPE1 and disposes it to a 501 printer.

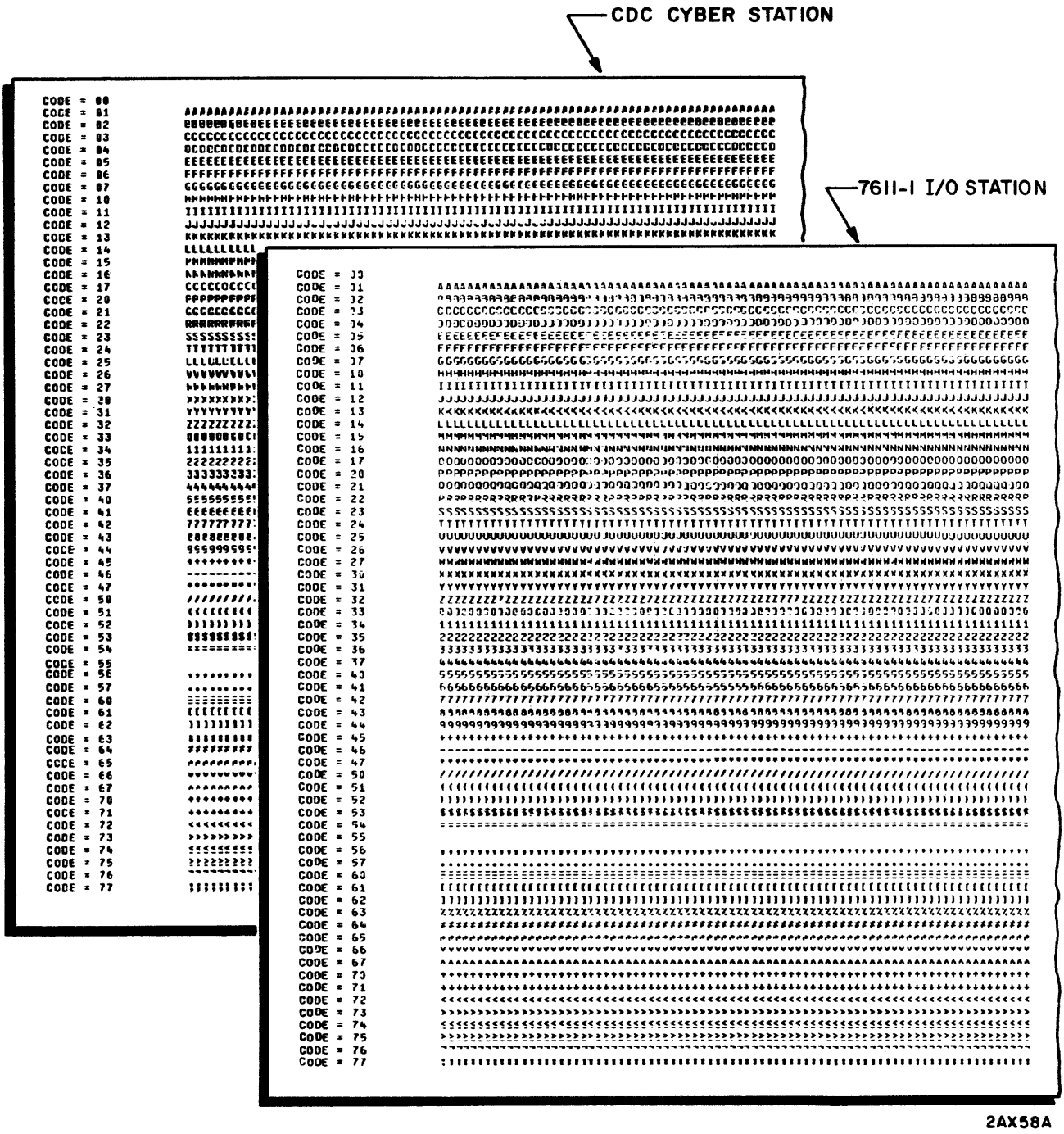


Example 9-7. Disposing Print File Created by FORTRAN Program

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 CDC CYBER 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 CDC CYBER station handles the % and : differently from the 7611-1 and that the FORTRAN program is unable to generate the 00 code.



Example 9-8. Generate Printer Character Sets



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.

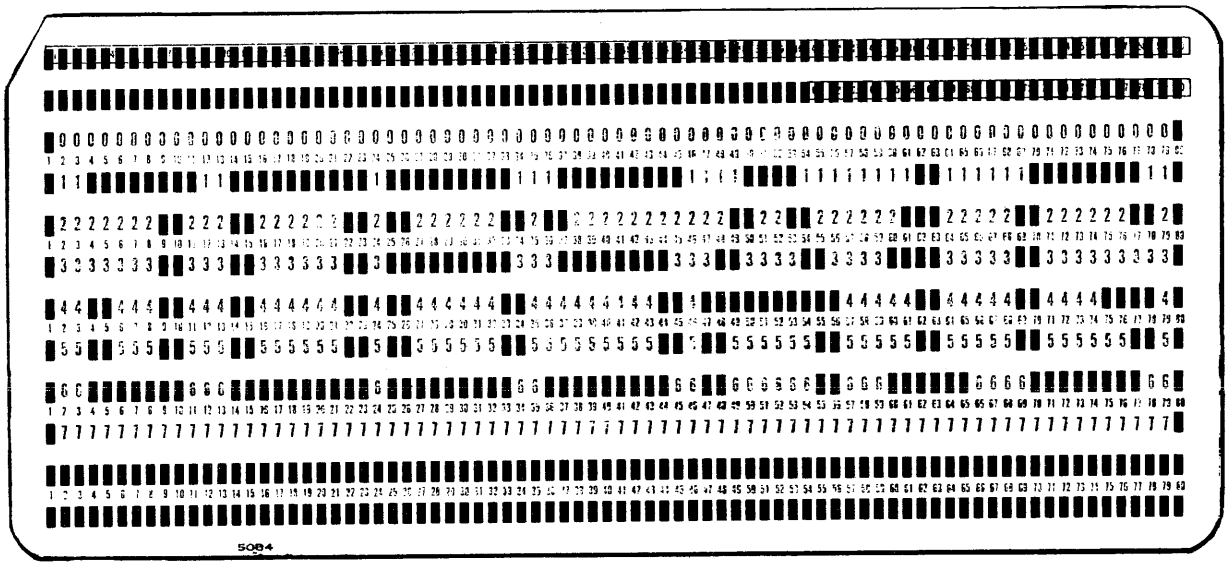


Figure 9-8. Lace Card

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 fewer characters which constitute a single punched card. If a record exceeds 80 characters, the CDC CYBER station continues the record on as many cards as are needed to hold the record. Each record begins on 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

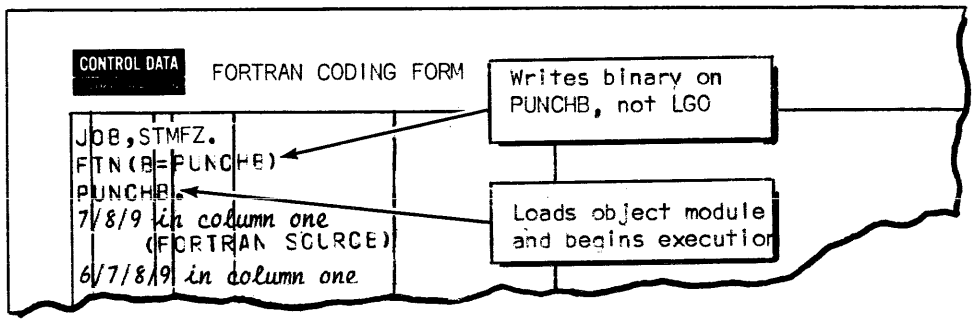
For a file to be punched as SCOPE binary cards, it must contain unblocked W records.

Normally, the record consists of a complete program 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.



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 PUNCHED 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 CDC CYBER 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, PZ, P9	Reserved for future use

FORMS CONTROL

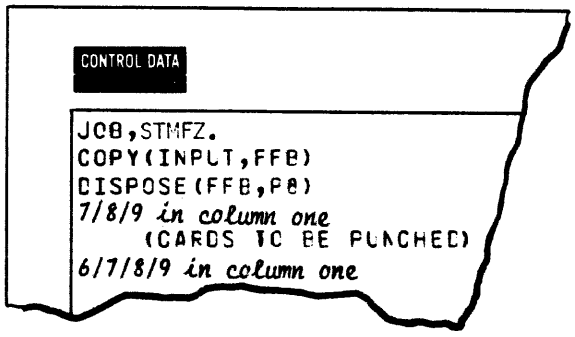
If you want to punch special cards, notify the operator by expanding the disposition code. For example,

```
DISPOSE(lfn, dt=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.



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. (Refer to 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 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.

The optional A parameter specifies that tape read parity errors are accepted by the copy routines. If the parameter is not included, any read parity errors cause the copy to be aborted. The errors are processed according to the error option (EO=x) specified on the FILE statement for the copy input file. For example, when the A

parameter is included on a copy control statement and the FILE error option is set for accept and display (EO=AD) any read parity errors encountered during a copy cause the bad data to be included in the copy, and written on a special file which is automatically printed at job end. If the A parameter is not included, the copy aborts, even though the FILE error option is set for accept. A user should check the dayfile for possible tape read parity errors whenever the A parameter is used to ascertain how good the copy is. Refer to Program Exit Conditions for more information.

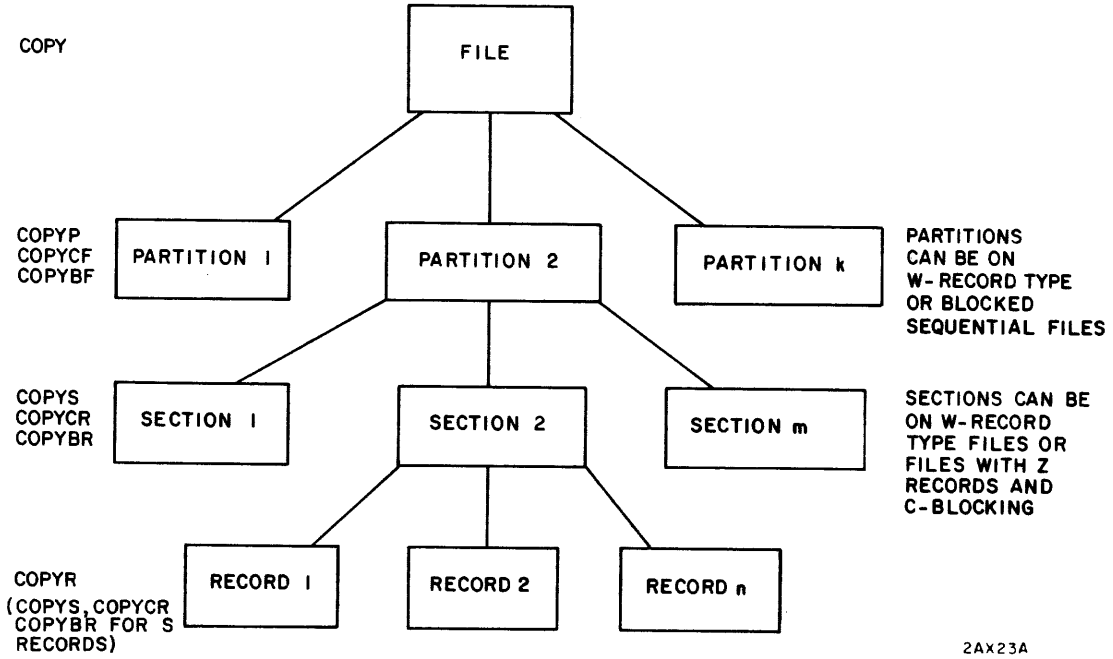


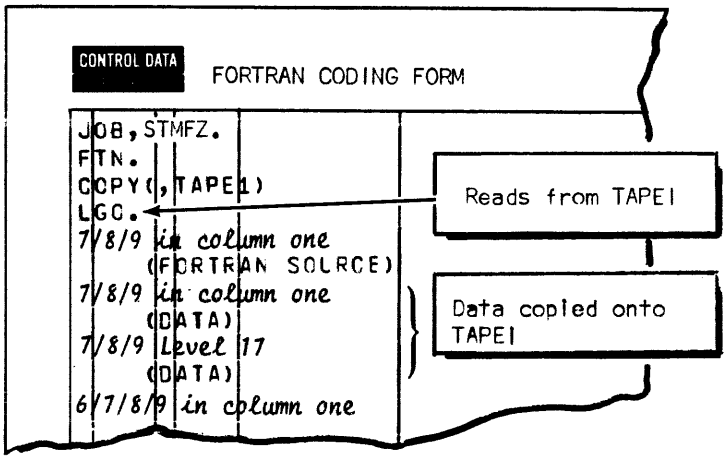
Figure 10-1. Selecting the Proper Copy Statement

FILE

To copy an entire file from its current position to end-of-information, use the following control statement.

COPY(lfn$_{in}$, lfn$_{out}$,,A)

Example 10-1 shows COPY being used to copy the data on INPUT to file TAPE1.



Example 10-1. File-to-File Copy

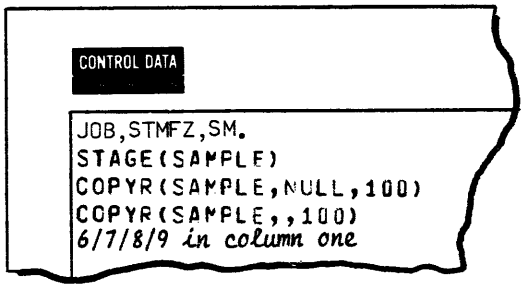
RECORDS

When copying records, use the following control statement.

COPYR(lfn$_{in}$,lfn$_{out}$,n,A)

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 records to the OUTPUT file.



Example 10-2. Copying Records

SECTIONS

When copying sections, use one of the following control statements:

COPYS(lfn$_{in}$,lfn$_{out}$,n,A)

COPYBR(lfn$_{in}$,lfn$_{out}$,n,A)

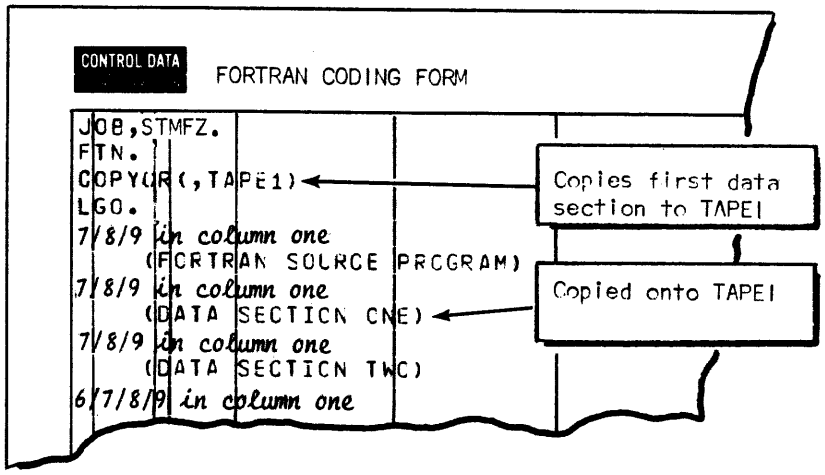
COPYCR(lfn$_{in}$,lfn$_{out}$,n,A)

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.



Example 10-3. Copying Sections

PARTITIONS

When copying partitions, use one of the following:

COPYP(lfn_{in}, lfn_{out}, n, A)

COPYBF(lfn_{in}, lfn_{out}, n, A)

COPYCF(lfn_{in}, lfn_{out}, n, A)

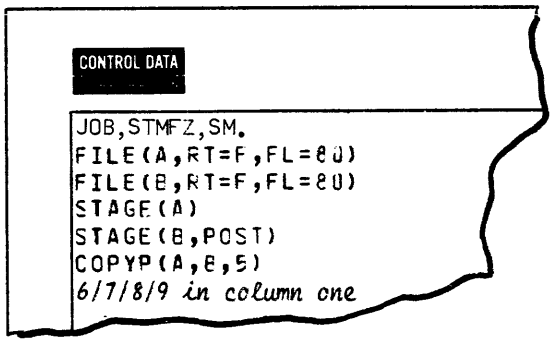
The results are identical regardless of the statement used. However, COPYBF and COPYCF provide a degree of compatibility with SCOPE 3.4. COPYP is not available under SCOPE 3.4.

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.

NOTE

The input file of jobs read through a NOS/BE card reader can never contain more than one end-of-partition card.

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.



Example 10-4. Copying Partitions

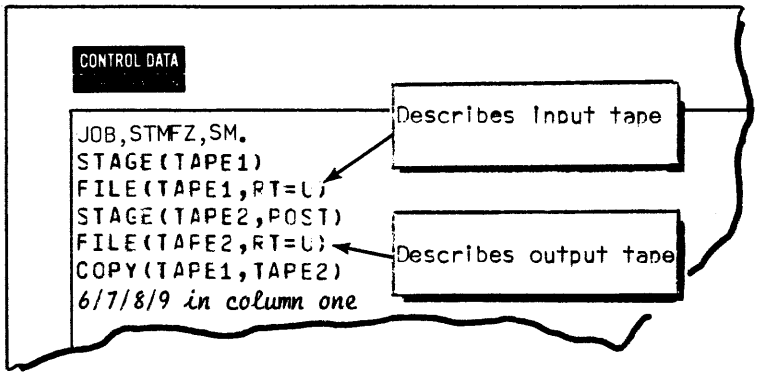
FILE DESCRIPTIONS FOR COPIES

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.



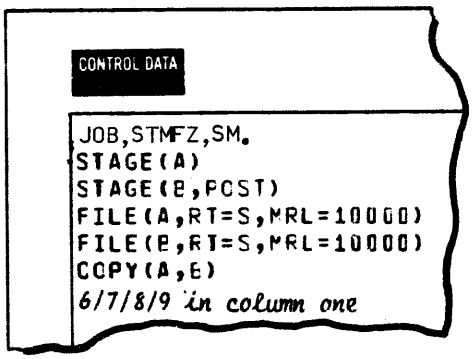
Example 10-5. Copying a Tape as Record Type U, Block Type K

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 memory 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.



Example 10-6. Setting MRL

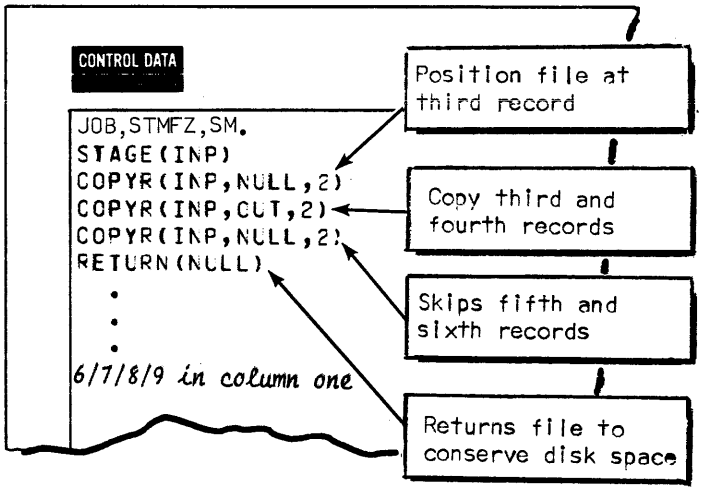
POSITIONING SEQUENTIAL FILES

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.



Example 10-7. Skipping Records Forward

SKIPPING SECTIONS FORWARD

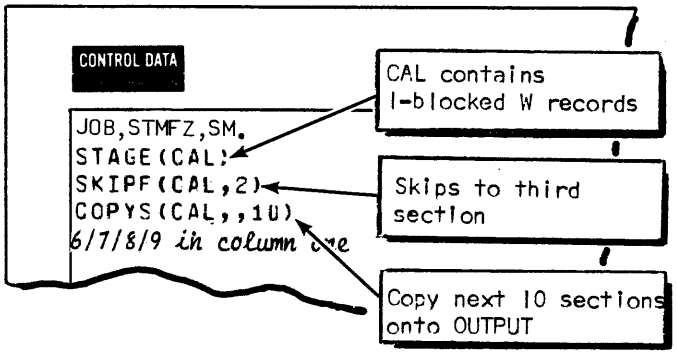
SCOPE 2 provides SKIPF statement that can be used for skipping sections forward on input file types that support sections (Z with C blocking and W) and as a special consideration, can be used for skipping S records. The statement has the following format:

```
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.



Example 10-8. Skipping Sections Forward

SKIPPING PARTITIONS FORWARD

SCOPE 2 provides a parameter on the skip forward statement (SKIPF) for skipping partitions on input 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 the number of records per block is 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.

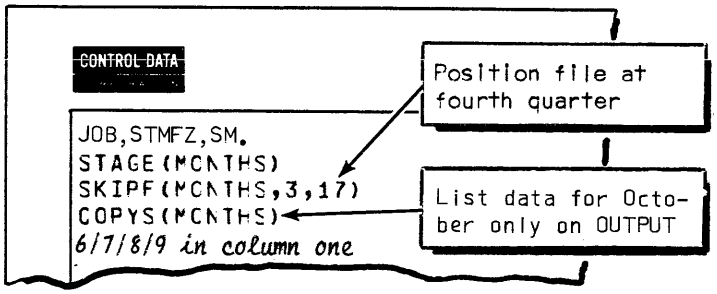
```
SKIPF(lfn, n, 17)
```

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, on 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.



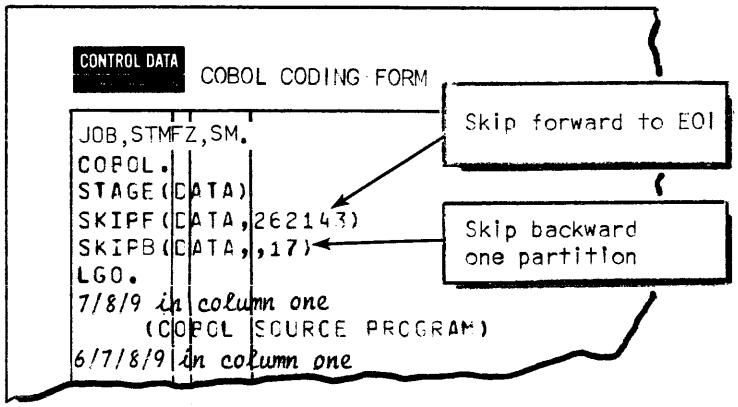
Example 10-9. Skipping Partitions Forward

SKIPPING TO END-OF-INFORMATION

For compatibility with SCOPE 3.4 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.



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:

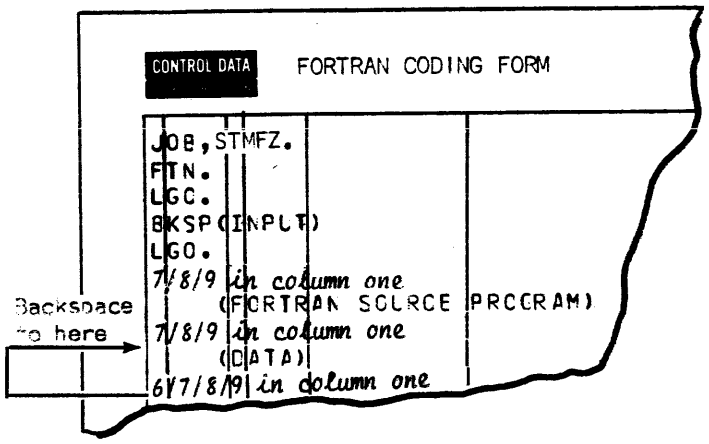
```
SKIPB(lfn,n)
```

```
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. If n is 0, the file is positioned at the beginning of the current section or partition.

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.



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 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 linked mainframe permanent file.

```
REWIND(lfn1, lfn2, ..., lfnn)
```

For compatibility with SCOPE 3.4 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). For all but S records, the buffer must be large enough to accommodate entire records. S records are compared by partial records and 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.)

When both files being compared are Z record files with BT=C, COMPARE treats them as if they were RT=S. This causes no difficulty if the Z record files are terminated with an end-of-section delimiter. In particular, if the Z record file is written by a COPY, COPYP, or COPYS statement, it is terminated properly. However, if COPYR is used and an end-of-section delimiter is not encountered, the file is improperly terminated and a subsequent COMPARE aborts with an invalid block error.

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(lfn1,lfn2)
```

Use of the preceding statement will compare one section. To specify more than one section, follow lfn_2 with a decimal count of the number of sections to be compared.

```
COMPARE(lfn1,lfn2,n)
```

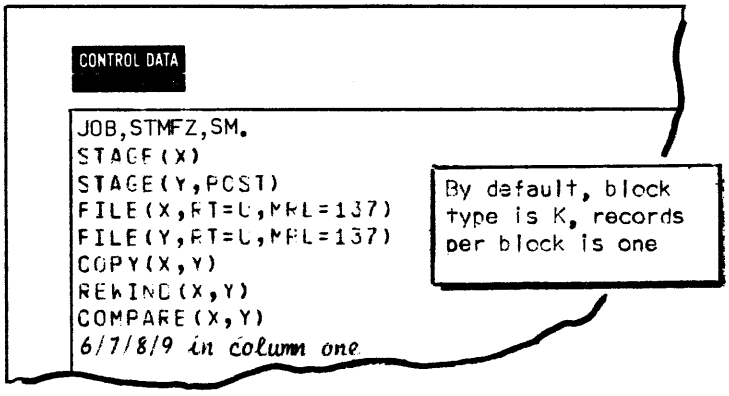
COMPARING PARTITIONS

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(lfn1,lfn2,n,17)
```

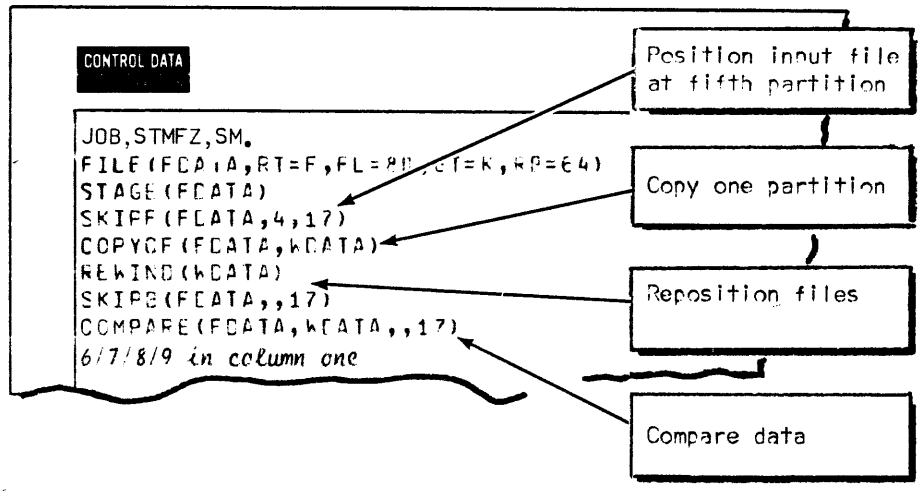
The 17 indicates partitions. If the file type does not support partitions, the comparison continues to EOI.

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.



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.



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 than 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, n S records of level lev, or n partitions have not been encountered.

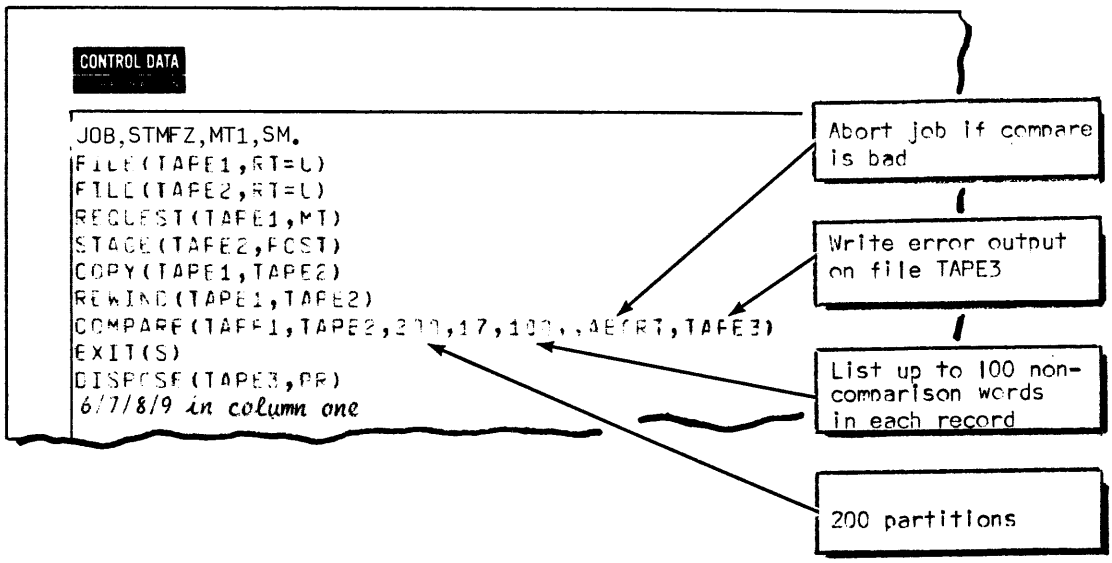
COMPARE(lfn$_1$,lfn$_2$,n,lev,e,r)

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,e,r,a,lfn$_3$)

Example 10-14 illustrates a request for list output generated by a literal comparison of U record input and output.



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:

- 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.
- 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.

$$\text{REQUEST(lfn,}\ \begin{matrix}\text{MT}\\ \text{NT}\end{matrix}\ \text{,...,}\ \begin{matrix}\text{N}\\ \text{E}\end{matrix}\text{)} \qquad \text{or} \qquad \text{STAGE(lfn,}\ \begin{matrix}\text{MT}\\ \text{NT}\end{matrix}\ \text{,...,}\ \begin{matrix}\text{N, POST)}\\ \text{E)}\end{matrix}$$

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.

$$\text{LABEL(lfn,}\ \begin{matrix}\text{R}\\ \text{W}\end{matrix}\text{,...)}$$

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

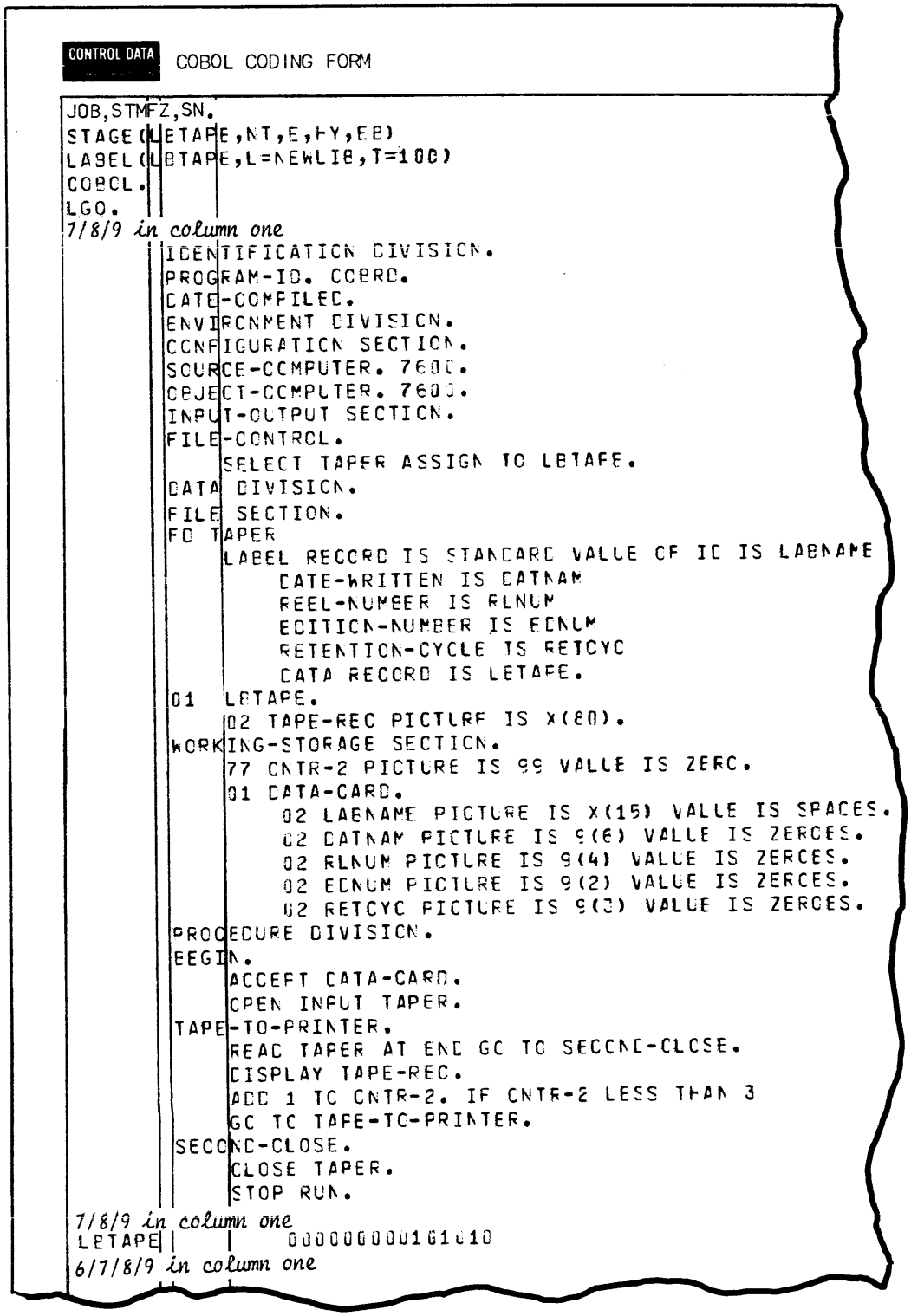
The LABEL statement does not require the existence of a REQUEST or STAGE statement for the file. It can be used to label a blocked sequential mass storage file. However, permanent files at the NOS/BE mainframe cannot be labeled.

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.



CONTROL DATA COBOL CODING FORM

```
JOB,STMFZ,SN.
STAGE(LBTAPE,NT,E,FY,EB)
LABEL(LBTAPE,L=NEWLIB,T=100)
COBCL.
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      0000000001610
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 set density at load point only. For a 7-track input tape, a tape driver attempts to read the label and 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.

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 memory dumps for analysis, and generation of listings of files for determining possible causes of program termination.

This section deals primarily with how to use these features of SCOPE 2. For a comprehensive analysis of a specific program, refer to Analyzing a Sample Program (Appendix G).

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 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 most abnormal conditions.

The type of abnormal condition that occurs dictates the type of EXIT processing performed. For some abnormal conditions, no EXIT processing occurs and the job is terminated immediately. Abnormal conditions can be classified as follows:

1. Job Step Abort - this type of condition terminates the current job step and causes SCOPE to search for any of the four types of EXIT control statements. Most abort conditions are in this classification.
2. Special Abort - this type of condition terminates processing of the current job step and causes EXIT processing to search for an EXIT(S) control statement.
3. Terminal Abort - a terminal abort condition terminates the current job step and also terminates the job immediately. No EXIT processing takes place.

Table 12-1 and Figure 12-1 show the action taken upon the occurrence of these types of abnormal conditions, the action taken when EXIT processing detects an EXIT control statement, and the action taken when the system encounters the end of control statements before encountering an EXIT statement.

Statements following an 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.

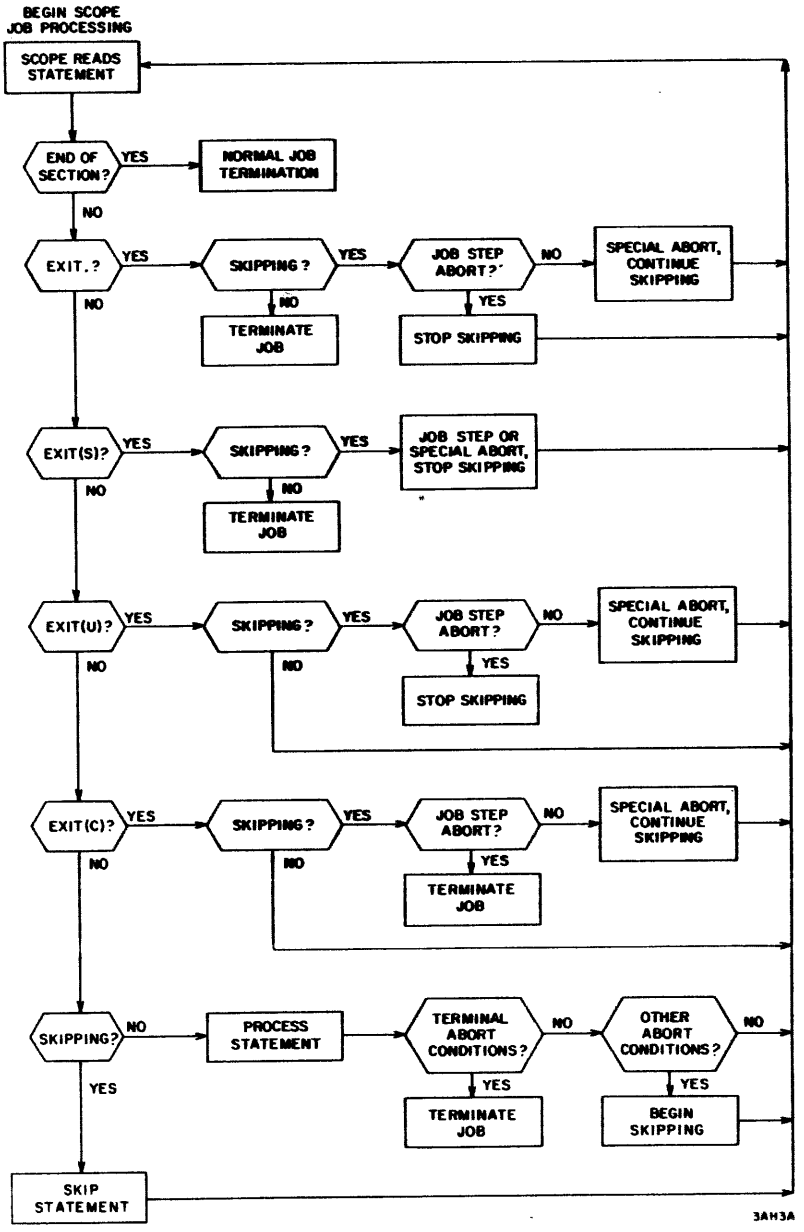


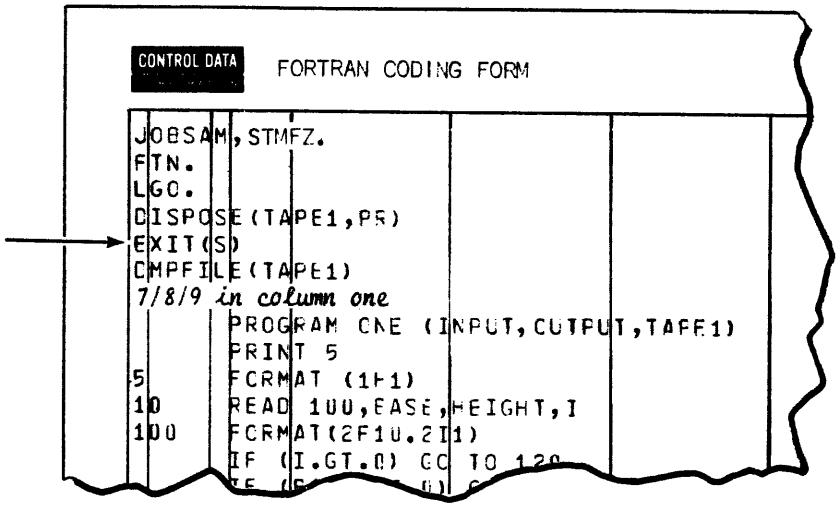
Figure 12-1. Flow Chart of EXIT Processing

TABLE 12-1. EXIT STATEMENT PROCESSING

Dumps				Condition Causing Job Step Termination	Type of Termination and Action taken on Occurrence	Action Taken When EXIT Encountered			
Standard User	JS LCM	JS SCM	Reprievable			EXIT.	EXIT(S)	EXIT(U)	EXIT(C)
			y	. Successful completion (no errors or only nonfatal errors)	Normal job step advance; advance to next control statement and process it; terminate job if EOS detected.	Terminate job	Terminate job	Normal job step advance	Normal job step advance
			y	. ENDRUN macro					
x	x		y	. User PSD error not selected by MODE control statement	Job step abort; abort job step, take indicated dumps, and skip all control cards until an EXIT card is found. Terminate job if no EXIT found before EOS.	Resume normal job step advance at control statement after EXIT.	Resume normal job step advance at control statement after EXIT(S)	Resume normal job step advance at control statement after EXIT(U)	Terminate job
x	x	x	y	. Job supervisor PSD error					
x			y	. Operator DROP					
				. Exceeded limit					
x			y	1. Time (first time only)					
x			y	2. Mass storage					
			y	3. LCM					
x	x			4. Maximum number tape units					
x			y	5. Dayfile messages					
x	x		y	. ABORT macro-no parameters					
x	x		y	. ABORT lfn macro					
			y	. ABORT ,nodump macro					
x	x		y	. Irrecoverable mass storage errors					
				. Control statement syntax error					
x	x		y	. Record Manager error					
x	x		y	. User-accessible LCM/SCM parity error on nonrerunnable job	Special abort; abort job step, take applicable dumps, and skip all control statements until an EXIT(S). (Terminate job if no EXIT(S) found before EOS.)	Continue skipping	Resume normal job step advance at control statement after (EXIT(S)	Continue skipping	Continue skipping
				. Loading binary program with compilation assembly errors					
			y	. ABORT ,nodump,S					
x	x		y	. ABORT ,,S					
				. Job card error	Terminal abort; abort job step and terminate job.	Not applicable	Not applicable	Not applicable	Not applicable
				. Job pre-abort					
				. Nonrerunnable job recovered by deadstart					
				. Account card error					
				. Operator KILL					
			y	. Operator RERUN					
x				. Time limit exceeded (second time)					
				. Dayfile message limit (second time)					

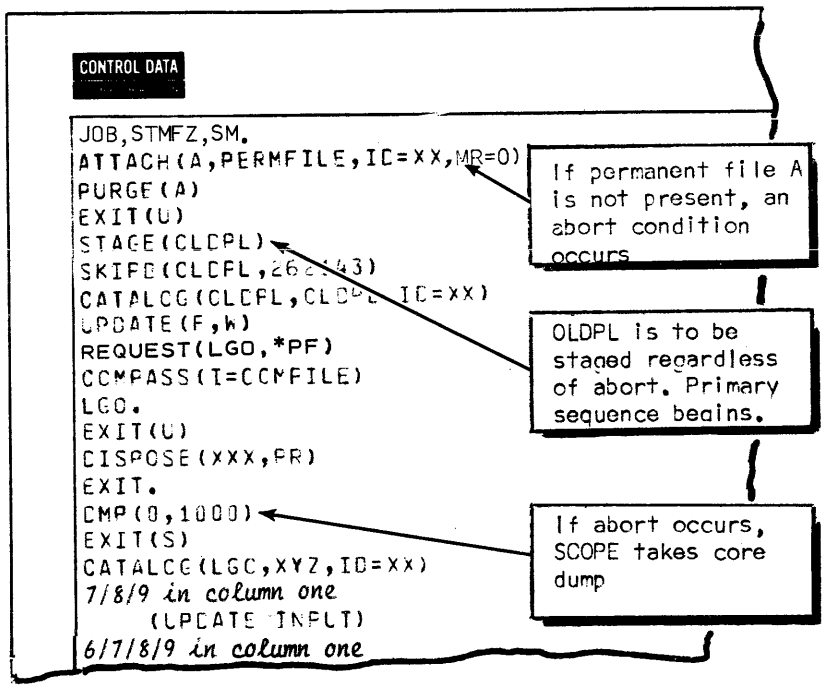
In summary, the EXIT. statement terminates processing if encountered with no error conditions, EXIT(C) resumes processing if encountered with no error conditions, and EXIT(U) resumes processing whether error conditions have occurred or not.

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.



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.



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 zero instruction.

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 (refer to 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

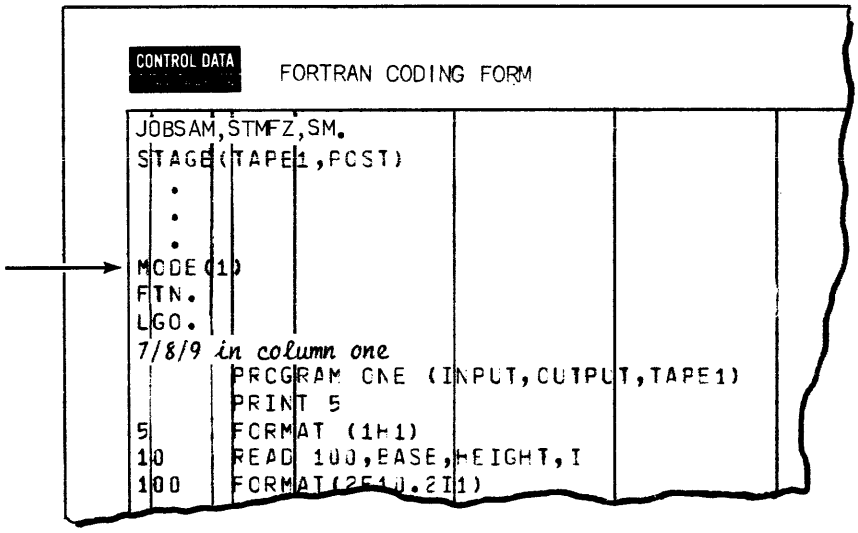
The format of the MODE statement is as follows:

```
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.



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)
```

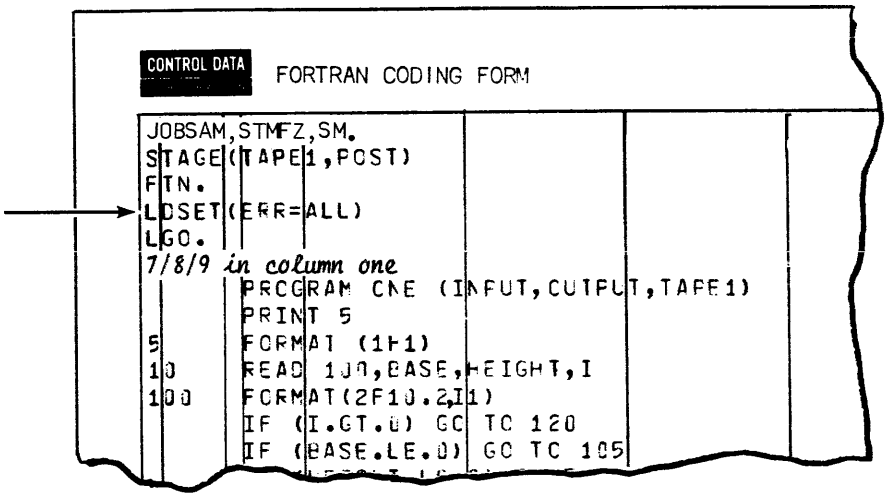
The following option causes the loader to terminate the job even for errors that are normally nonfatal.

```
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.



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, 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 SCM addresses referenced in the A, B, or X registers and the contents of LCM addresses referenced in the X registers.
- Contents of the first 100_8 locations in the SCM field (RAS+0 through RAS+77_8). This is the job communication area used for communication between your job and SCOPE.
- 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, if within the user's field length.

Figure 12-2 illustrates a sample standard dump. For an analysis of a sample dump, refer to appendix G.

DUMP OF EXCHANGE PACKAGE

DUMP OF JOB COMMUNICATION AREA

SCM MEMORY DUMP

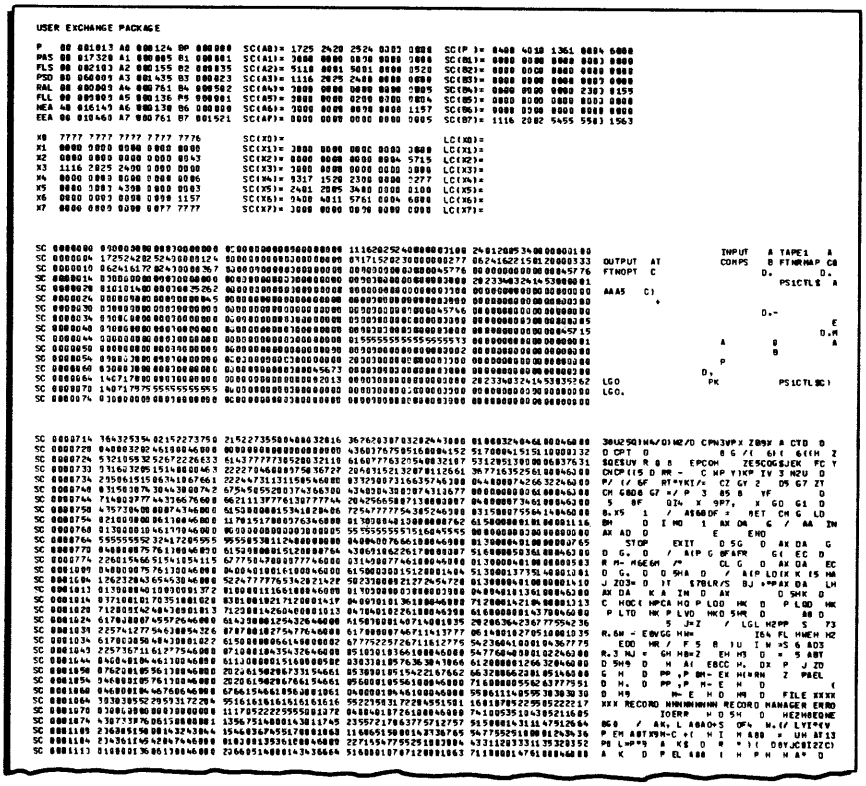


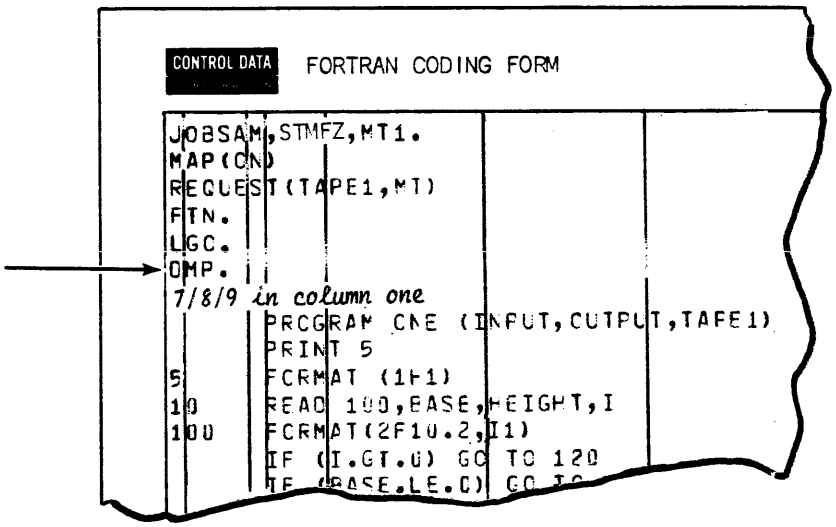
Figure 12-2. Standard Dump

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.



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)
```

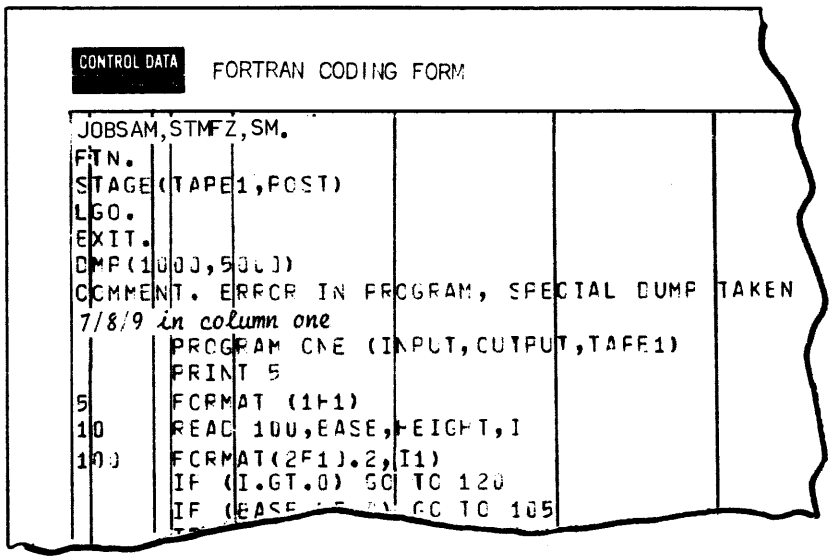
Thus, if you wished a dump of locations 200_8 through 500_8, you would use

```
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. If fwa exceeds the range of SCM, the memory dump is not taken.

Unlike SCOPE 3.4, SCOPE 2 does not provide a means of obtaining a dump of SCM using absolute rather than relative addresses.



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. Because the DMPL/DMPECS program executes in the user field length, request any dump of SCM before requesting the LCM dump.

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 (four 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 four words per line in four groups of 20 octal digits accompanied by character interpretation
2	List two words per line in four five-digit groups accompanied by character interpretation
3	List two words per line in five four-digit groups accompanied by character interpretation
4	List two words per line in two 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 as illustrated in Figure 12-3.

(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) Block name within the program or overlay, where // means blank common

(11) Absolute beginning address of block, where L following address indicates an LCM address

(12) Number of words in the block in octal

(13) Name of the file from which the block was loaded. If the name is a user library, it is prefixed by the letters UL; if the name is a system library, it is prefixed by the letters SL.

(14) List of entry points occurring in the program or overlay

(15) List of subroutines that reference these entry points

(16) Absolute address within the subroutine where the entry point is referenced

(17) List of unsatisfied externals

(18) List of subroutines in which the unsatisfied externals occur

(19) 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 (refer to Table 12-3 for MAP parameters).

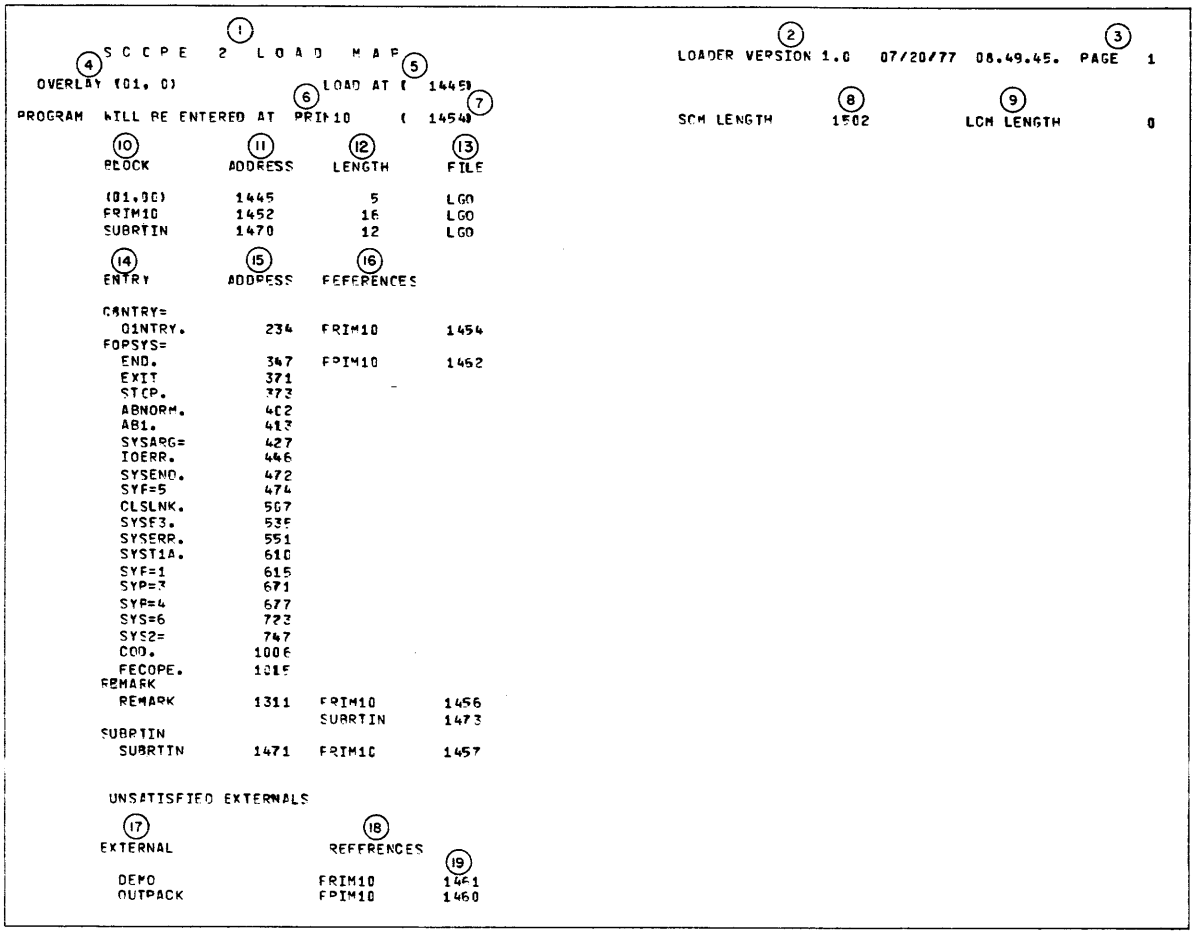


Figure 12-3. SCOPE 2 Load Map

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 (refer to Table 12-3 for options).

TABLE 12-3. MAP OPTIONS

MAP Contents	MAP Statement Parameter	LDSET Option
No map	OFF	MAP=0 or MAP=O
Partial map (items 1-9)	---	MAP=S
Partial map (items 13-15 17, 18 omitted)	PART	MAP=B
Partial map (items 14, 15 17, 18 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. Also refer to appendix G.

OBTAINING FILE DUMPS

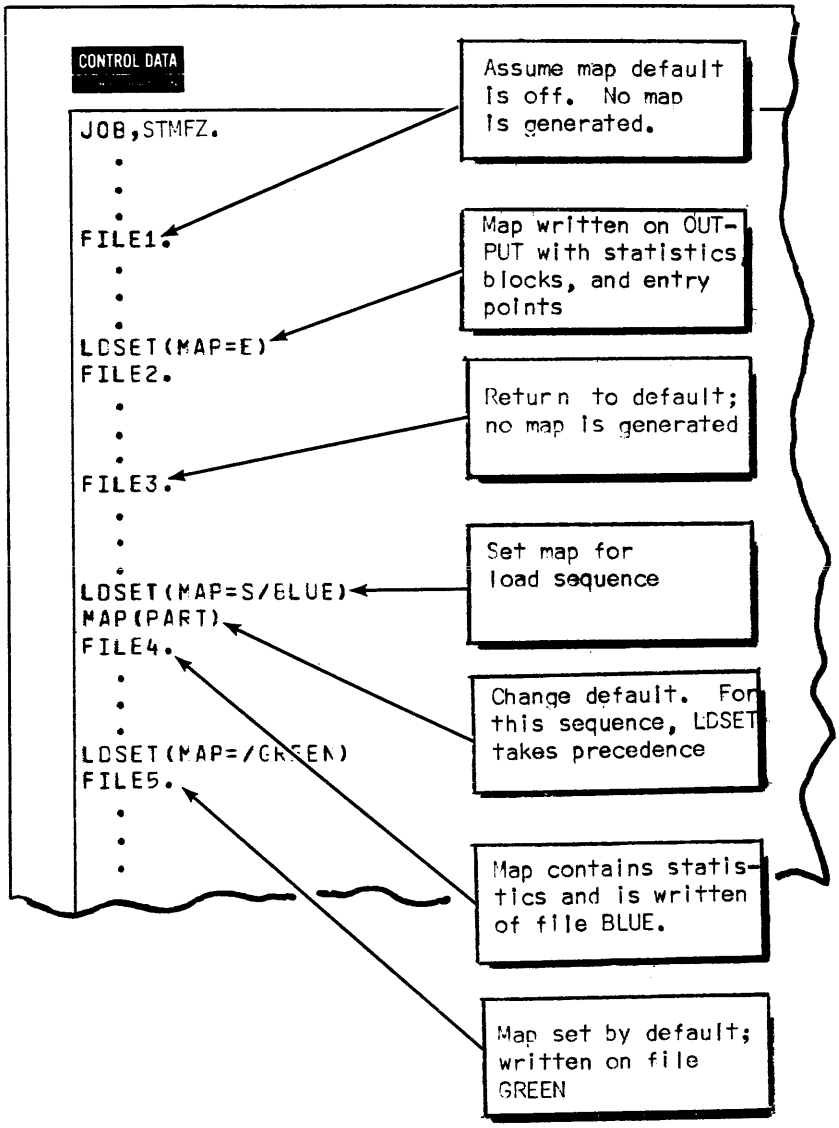
As an analytical aid, SCOPE 2 provides a utility routine that lists portions of a file. This routine is called with the DMPFILE statement and includes a number of options.

The dump format (Figure 12-4) 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)
```



Example 12-7. User Control of Load Map

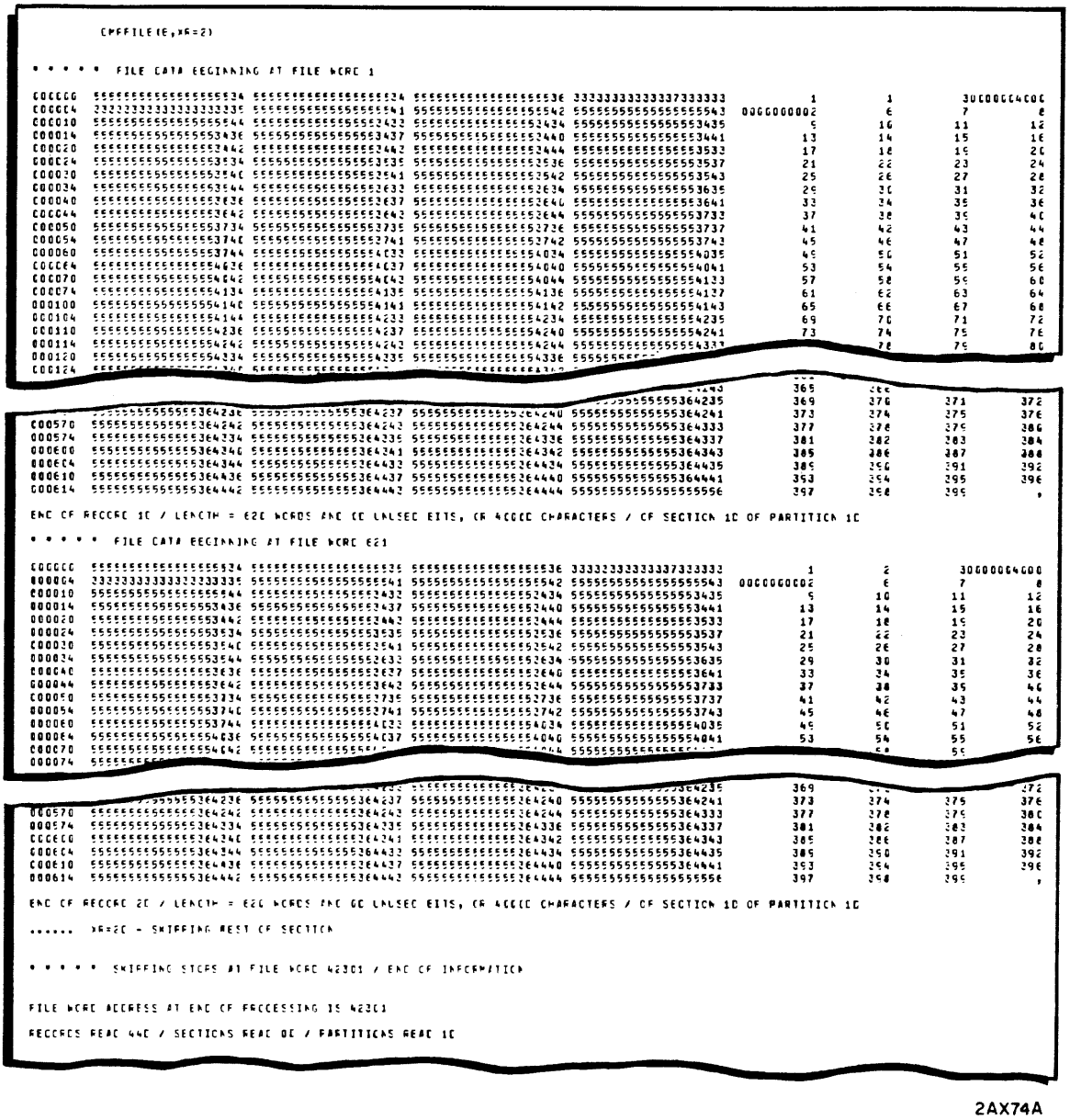


Figure 12-4. Sample of DMPFILE Output

SPECIFYING A LIST FILE

If you want your DMPFILE output placed on a file other than OUTPUT, specify a list file as $L=lfn_{out}$.

$$DMPFILE(lfn_{in}, L=lfn_{out})$$

Remember that if you specify a list file other than OUTPUT, you are responsible for saving the file contents.

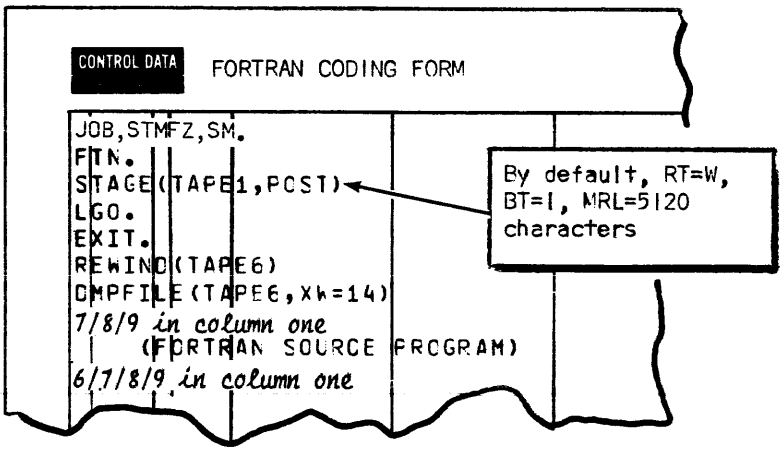
SPECIFYING DUMP LIMITS

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. Default is until end-of-record.
XR=n	n is a decimal count of the number of records in each section to be dumped. Default is until end-of-section.
XS=n	n is a decimal count of the number of sections in each partition to be dumped. Default is until end-of-partition.
XP=n	n is a decimal count of the number of partitions in the file to be dumped. Default is until end-of-information.

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.



Example 12-8. Requesting a File Dump

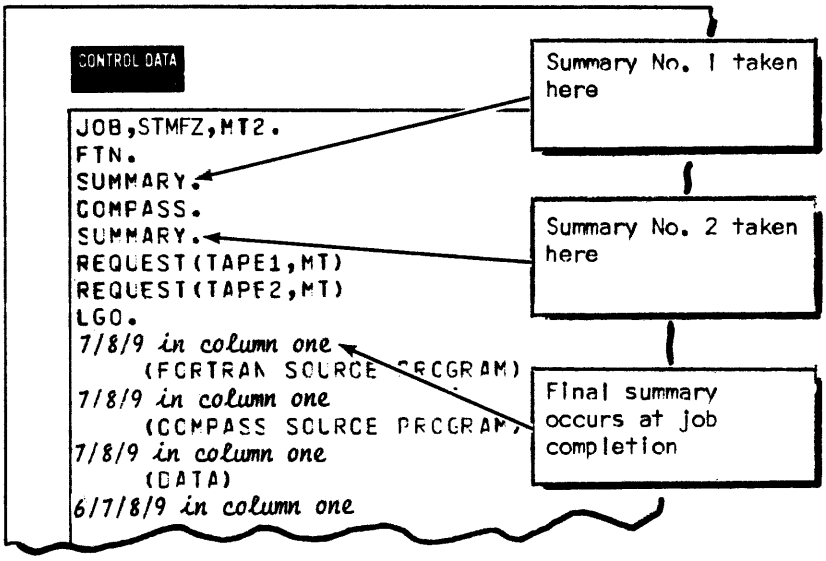
OBTAINING DAYFILE SUMMARIES

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.

- 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.
- 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.
- Relinquish a resource as soon as you no longer need it. This means that you should RETURN any mass storage file or on-line tape unit when you are through using it without waiting for job termination. This includes post-staged tapes and disposed files. RETURNing an on-line tape unit reduces the amount of system resources required by your job and may allow it to complete processing sooner. If, however, the job will need the tape unit again later in the job, the on-line tape unit should be UNLOADed rather than RETURNed; otherwise, the job will abort if the device count is decremented to zero.



Example 12-9. Intermediate Accounting Information

CDC CYBER CONTROL LANGUAGE 13

This section explains how a user can manipulate control statements by using the CDC CYBER Control Language (CCL). The following paragraphs describe CCL expressions. This is followed by an overview of the CCL control verbs, functions, and procedures. The remainder of the section explains how to use CCL statements and procedures.

CCL consists of control statements that the user can insert in the control statement section of a job to initiate tests, substitutions, transfers, and loops within these statements. CCL can also assign values to symbolic names, print results in the job's dayfile, and determine the status of a file. CCL can process control statements in a file other than the original job file.

SYNTAX

A CCL statement consists of a CCL verb followed by one or more separators between parameters and a terminator. The separator following the verb in a CCL statement must be a comma or a left parenthesis. The separator between parameters must be a comma. A CCL statement terminator must be a period or a right parenthesis. Literals ($-delimited character strings) are valid within all CCL statements, but are not evaluated during substitution within a procedure (described later in this section). CCL ignores all blanks except blanks within a literal and within a verb. Blanks cannot be used within a verb. Any CCL statement can continue over more than one card or line if the last character of a continued card or line is a valid CCL separator. Some CCL statements allow only a subset of the generally acceptable separators and terminators. Such restrictions are noted in the descriptions of the applicable statements.

EXPRESSIONS

A CCL expression can form part of many CCL statements. An expression follows the separator after the CCL verb. A CCL expression is similar to the expressions used with higher-level languages. An expression consists of operators and operands and can include other expressions that are enclosed in parentheses. For example, the expression 6 * (R1 - 2) contains the expression R1 - 2.

OPERATORS

There are three types of CCL operators: arithmetic, relational, and logical.

Arithmetic Operators

The following arithmetic operators perform the various arithmetic operations.

+	Addition
-	Subtraction, negation
*	Multiplication
/	Division
**	Exponentiation

Relational Operators

The following are CCL relational operators.

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

Logical Operators

The following are CCL logical operators.

.AND.	AND (both A and B)
.EQV.	Equivalence
.NOT.	Complement
.OR.	Inclusive OR (either A or B or both)
.XOR.	Exclusive OR (either A or B, but not both)

Order of Evaluation

1. Exponentiation
2. Multiplication, division
3. Addition, subtraction, negation
4. Relations
5. Complement

6. AND

7. Inclusive OR

8. Equivalence, exclusive OR

If there is more than one operation of equal order, evaluation proceeds from left to right.

OPERANDS

An operand can be of any of the following:

- Expression — A CCL expression enclosed between separators; the expression is evaluated, and the result is an operand.
- Integer constant — A string of 1 to 10 numeric or literal characters.
- Symbolic name — A string of 1 to 10 alphanumeric characters that is recognized by CCL (described later in this section).
- CCL function — A CCL-defined operand which determines the attributes of a file or symbolic name. An expression can consist entirely of a CCL function. (CCL functions are described in Functions, later in this section.)

Expression

An expression must have a terminator within the first 50 operands. Expressions can be used with the CCL statements IFE, WHILE, DISPLAY, and SET, and the FILE function. The separator preceding an expression is not part of the expression. The user may find that using a comma for such separators, rather than a left parenthesis, improves readability. The separator following the expression must be a comma. Computations are accurate to 10 decimal digits (20 octal digits), and overflow is ignored.

Any character string beginning with a numeric character is treated by CCL as numeric. Therefore, such a string cannot contain any nonnumeric characters, except for an optional post radix of B or D indicating octal or decimal, respectively. Any alphanumeric string must begin with an alphabetic character.

Integer Constant

An integer constant is usually numeric, although it can also be a literal. It must be 10 characters or less, including the post radix if it is specified. If the post radix is omitted, the constant is assumed to be decimal. A literal (a $-delimited character string) must be 10 characters or less, excluding the $ delimiters.

Symbolic Name

A symbolic name is an alphanumeric character string that identifies a numeric value. Table 13-1 describes commonly used symbolic names which are valid in any CCL expression. Less commonly used symbolic names are described in the SCOPE 2 Reference Manual.

TABLE 13-1. COMMONLY USED SYMBOLIC NAMES

Name	Description	User's Range of Values
R1	Value in control register 1	0 to 131071D or 377777B
R2	Value in control register 2	0 to 131071D or 377777B
R3	Value in control register 3	0 to 131071D or 377777B
R1G	Value in global control register 1	0 to 131071D or 377777B
DSC	Dayfile skipped control statement flag	0 and 1
EF	Error flag	0 to 62D or 76B
EFG	Global error flag	0 to 62D or 76B

The symbolic names R1, R2, R3, and R1G are control registers that the user can equate to an expression. CCL evaluates the expression as a numeric value. For example, if the user sets R2 equal to three, then the statement SET,R1=R2 + 2. would be evaluated by CCL, and R1 would equal five.

A value set in R1G, the global control register, can be passed from a procedure file to the job control statement section of a job. The values of R1, R2, and R3 are not retained when passing from a procedure file to the job control section of a job. An example illustrating passing of parameters is shown in the subsection on Procedure Return.

DSC is also a variable, but both the user and CCL control its value. Unless the user specifies value of DSC to be one, DSC remains zero and skipped statements are not printed in the dayfile. When DSC is set to one, any skipped statements that follow are printed in the dayfile. Some CCL error processing changes DSC to one, forcing skipped statements to be printed.

Both the user and CCL can set the values of EF and EFG. If CCL encounters an error, it changes the value of EF to the numeric value of the error type. After processing a procedure (described later in this section), CCL sets the value of EFG to the value of EF in the procedure, unless the value of EFG is already nonzero.

The symbolic names with true or false values are:

TRUE or T = 1

FALSE or F = 0

CCL STATEMENT OVERVIEW

The following CCL verbs assign and print values associated with symbolic names.

- SET — Assigns values of variable symbolic names to special CCL software registers, error flags, or the dayfile skipped control statement flag.
- DISPLAY — Evaluates a CCL expression and prints the result in the job's dayfile.

The following functions are used within an expression.

- FILE — Determines the attributes of a file.
- DT — Determines the type of device on which a file resides.
- NUM — Determines if a parameter has a numeric value.

The following verbs cause CCL to conditionally process or skip control statements.

- IFE — If the expression in the IFE statement is true, CCL processes the following statements. If false, control skips until a terminating statement is found.
- SKIP — Skips until the SKIP-terminating CCL statement ENDIF is found.
- ELSE — Either terminates or initiates skipping, depending upon the other CCL statements used.
- ENDIF — Terminates skipping initiated by IFE, SKIP, and ELSE.

The following CCL verbs identify a group of control statements as a loop that can be repeatedly processed.

- WHILE — Begins the loop. The group of statements is processed as long as the WHILE expression is true. If the expression is false, control skips to the ENDW statement.
- ENDW — Ends the loop.

The following CCL verbs control processing of control statements in a file other than the original job file. This separate group of statements is called a procedure.

- PROC — Identifies the statements that follow as a procedure.
- BEGIN — Initiates processing of a procedure.
- REVERT — Returns processing from a procedure to the control statement that is to be processed next in the control statement section of the job.

SET AND DISPLAY STATEMENTS

SET

The SET statement allows the user to assign the value of a symbolic name to a CCL control register, error flag, or dayfile skipped control statement flag. Only a subset of the symbolic names known to CCL can be used with SET. This subset consists of the control registers (R1, R2, R3, R1G), the error flags (EF, EFG), and the dayfile skipped control statement flag (DSC).

NOTE

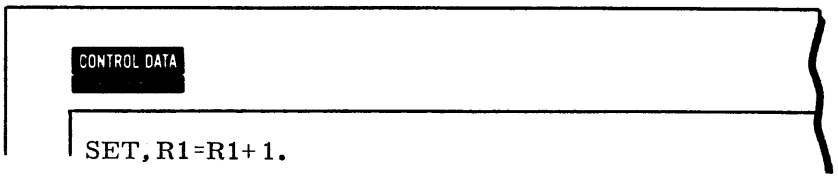
If the value of the expression used in the SET statement is too large (according to the user's range of values for the symbolic names), it is truncated and no error message is issued.

The SET statement has the following format:

```
SET,sym=exp.
```

sym identifies the symbolic name that is to be set. The CCL expression (exp) consists of integer constants, symbolic names, or CCL functions.

As shown in Example 13-1, the SET statement included in a control statement loop increases R1 by one each time the loop is processed.



Example 13-1. SET Statement

Additional uses of the SET statement are given in the examples for the other control statements.

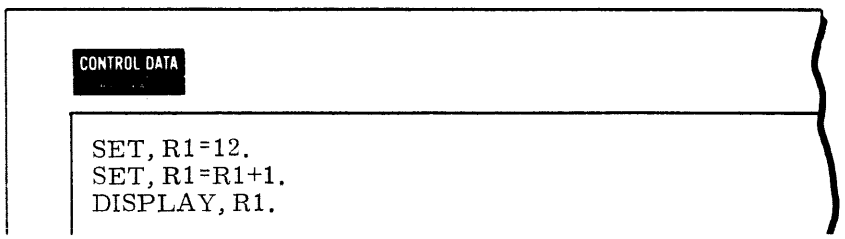
DISPLAY

The user can employ the DISPLAY control statement to evaluate an expression and print the result in the dayfile in both octal and decimal format. The DISPLAY statement has the following format:

```
DISPLAY, exp.
```

exp is a CCL expression consisting of character strings that must be integer constants, symbolic names, or CCL functions. It is the same parameter as exp in the SET statement (described previously).

Using the previous example for SET, where in a control statement loop R1 increases by one each time the loop is processed, the DISPLAY statement (Example 13-2) can be used to display the contents of the register R1.

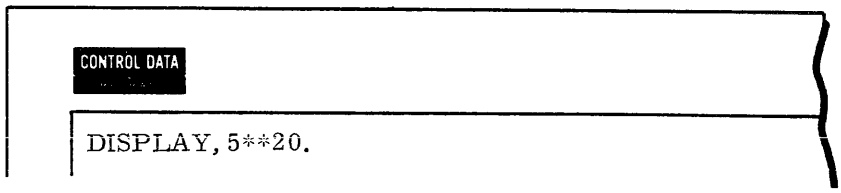


Example 13-2. DISPLAY Statement Register 1

The entry in the dayfile is as follows:

```
DISPLAY,R1
          13          15B
```

In Example 13-3, the number in R2 is larger than 10 digits. The result in the dayfile is GT followed by 9999999999. If the value is negative and larger than 10 digits, LT followed by a minus and 9999999999 is displayed. An octal value as large as 20 digits can be displayed.

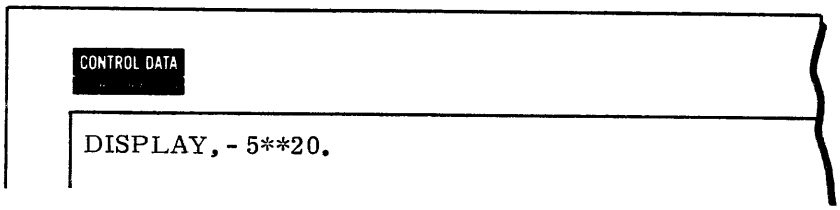


Example 13-3. DISPLAY Statement Register 2

The entry in the dayfile is as follows:

```
GT    9999999999        2553616570553061B
```

In Example 13-4, the value is negative and requires more than 10 digits.

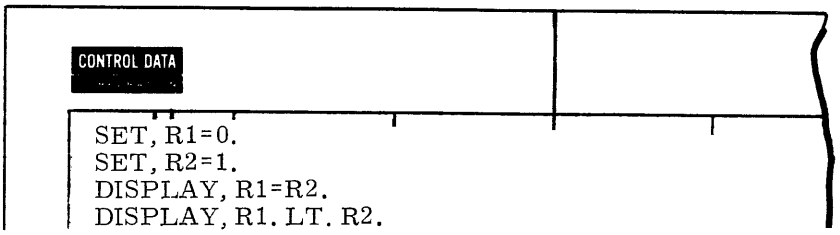


Example 13-4. DISPLAY Statement

The entry in the dayfile is as follows:

```
LT  -9999999999       -2553616570553061B
```

The DISPLAY statement can also evaluate an expression as true or false, displaying a 1 for true and a 0 for false. Example 13-5 shows an evaluation of false when displaying R1 equal to R2, and an evaluation of true when displaying R1 less than R2.



Example 13-5. DISPLAY Statement True or False

The entry in the dayfile is as follows:

```
DISPLAY,R1=R2.
          0           0B

DISPLAY,R1.LT.R2.
          1           1B
```

FUNCTIONS

The CCL functions, FILE and DT, determine attributes of a file. The NUM function determines whether a parameter is numeric. CCL functions can be used as an entire CCL expression or as a part of one. Functions are not statements but rather are part of a CCL statement.

This section does not describe DT and NUM, since they are not so commonly used by the applications programmer as the FILE function. Refer to the SCOPE 2 Reference Manual for a description of DT and NUM.

FILE

The FILE function is used in CCL statements to determine the status of any file assigned to the job. Status includes file type, location, and accessibility. The FILE function has the following format:

FILE(lfn, exp)

lfn is the name of the file for which the status is being determined.

exp is an expression consisting of operators and one or more special FILE symbolic names. The parentheses and comma must be used exactly as shown in the format.

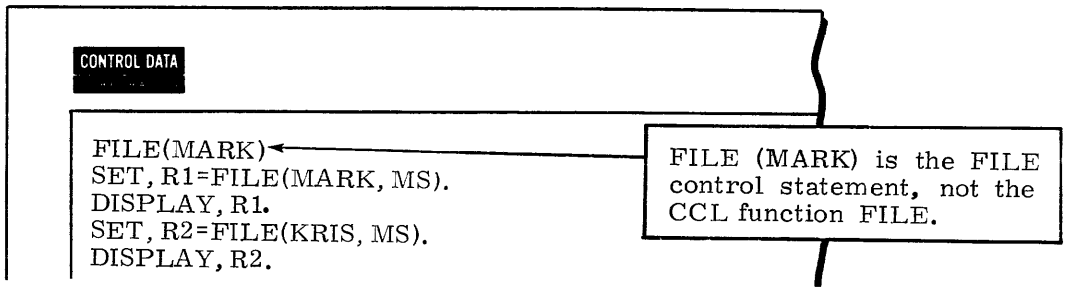
CCL evaluates the expression (exp) and gives it a value of 1 if it is true, and a value of 0 if it is false. The following are symbolic names and their use.

File Type	Use
IN	Input file.
LB	Labeled file.
LO	Local file; that is, the file is a temporary (scratch) file. Attached permanent files are not local.
OP	File that is opened.
PF	Attached permanent file.
PH	Punch file.
PR	Print file.

File Location	Use
AS	File is assigned to the user's job; that is, the file exists and is recognized by SCOPE 2.
BOI	File is positioned at the beginning of information (valid only for mass storage files).
EOF	The last operation moved forward and encountered an EOP; the file is now positioned after that EOP.
EOI	The last operation moved forward and encountered an EOI; the file is now positioned at the EOI.
MS	File is on mass storage.
TP	File is on magnetic tape.
TT	File is connected to a remote terminal.

File Accessibility	Use
EN	File has extend permission.
MD	File has modify permission.
RD	File has read permission.
WR	File has write permission (includes modify and extend permission).

In Example 13-6, the files named MARK and KRIS are checked to see if they are on mass storage. CCL displays a 1 or true value for file MARK and a 0 or false value for file KRIS.



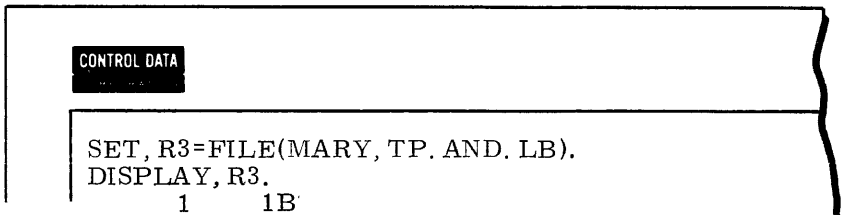
Example 13-6. FILE Function Using Symbolic Name MS

The entry in the dayfile is as follows:

```
DISPLAY, R1.
      1      1B

DISPLAY, R2.
      0      0B
```

In Example 13-7, if file MARY is on magnetic tape and is labeled, the DISPLAY statement evaluates the function as true.



Example 13-7. FILE Function Using TP and LB

Additional examples of the FILE function are shown in the descriptions of the other CCL statements.

CONDITIONAL STATEMENTS (IFE, SKIP, ELSE, ENDIF)

Use the conditional CCL statements to bracket groups of other control statements to be skipped or conditionally processed. Conditional statements begin with a verb and end with a label string consisting of 1 to 10 alphanumeric characters, beginning with an alphabetic character. The label string on the terminating statement must match the label string on the beginning statement.

IFE

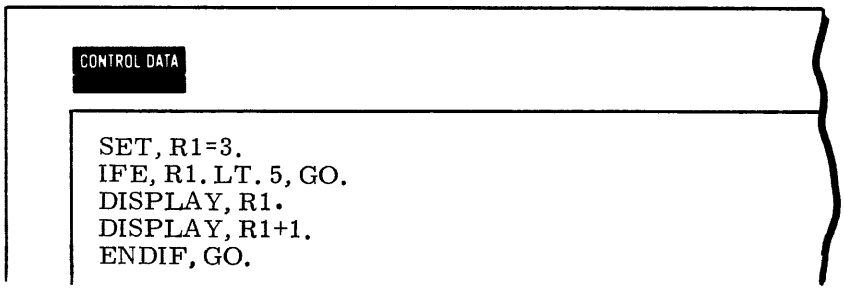
The IFE statement conditionally initiates the skipping of a group of succeeding control statements. If the expression in the IFE statement is true, the statements that follow are processed; if false, the statements are skipped.

The IFE statement has the following format:

```
IFE, exp, ls.
```

exp is a CCL expression and ls is a label string.

In Example 13-8, control register 1 (R1) is set to 3. The IFE statement says that if R1 is less than 5, process all of the following statements. Since the value set in R1 is less than 5, all statements are processed. GO is the label string on the IFE statement. It must match the label string on the terminating statement, ENDIF (described later in this section).

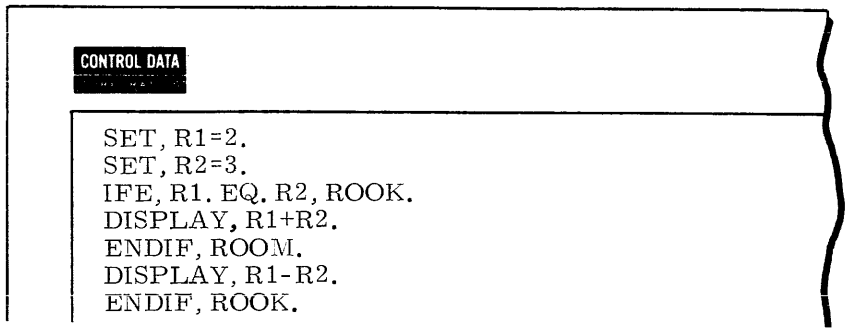


Example 13-8. IFE Statement

The entry in the dayfile is as follows:

```
SET, R1=3.
IFE, R1. LT. 5, GO.
DISPLAY, R1.
      3      3B
DISPLAY, R1+1.
      4      4B
ENDIF, GO.
```

In Example 13-9, the IFE expression is false and control skips until an ENDIF statement with a matching label string is found. The first ENDIF statement label string does not match, and so the second DISPLAY statement is also skipped.

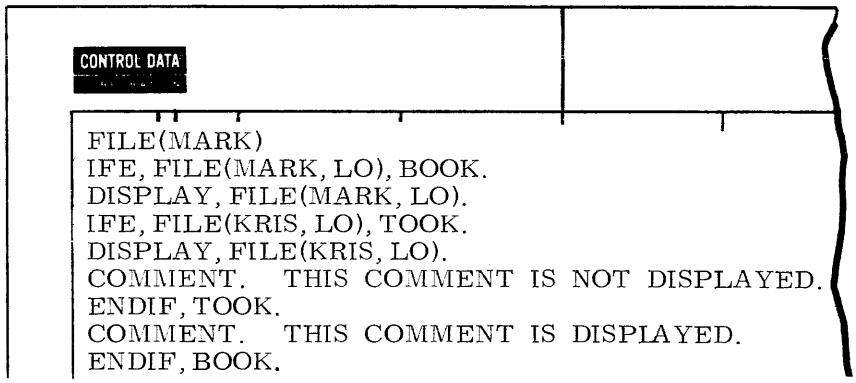


Example 13-9. IFE Expression is False

The entry in the dayfile is as follows:

```
SET, R1=2.
SET, R2=3.
IFE, R1. EQ. R2, ROOK.
ENDIF, ROOK.
```

Example 13-10 shows a conditional IFE statement terminated with an ENDIF statement, with another IFE and ENDIF sequence within the bracketed group of statements. The first IFE expression is true, and the second IFE expression is false.



Example 13-10. Conditional IFE Statement Within Another IFE Statement

The entry in the dayfile is as follows:

```
FILE(MARK)
IFE, FILE(MARK, LO), BOOK.
DISPLAY, FILE(MARK, LO).
      1       1B
IFE, FILE(KRIS, LO), TOOK.
ENDIF, TOOK.
COMMENT. THIS COMMENT IS DISPLAYED.
ENDIF, BOOK.
```

SKIP

Use the skip statement to unconditionally skip control statements that follow it. Terminate skipping with the ENDIF statement (described later in this section).

The SKIP statement has the following format:

```
/SKIP,ls.
```

ls is the label string.

In Example 13-11, SKIP initiates skipping and all following statements are ignored until ENDIF.

CONTROL DATA

```
SKIP,BROOK.
DISPLAY,EF.
COMMENT.  NO COMMENT WILL APPEAR ON THE DAYFILE.
ENDIF,BROOK.
```

Example 13-11. SKIP Statement

The entry in the dayfile is as follows:

```
SKIP,BROOK.
ENDIF,BROOK.
```

ELSE

Use the ELSE statement with the IFE statement to either initiate or terminate a skip. Remember that if the expression in an IFE statement is false, IFE initiates a skip to the first ENDIF or ELSE statement that has a label string matching the label string specified in the IFE statement. In this case the ELSE statement acts as a terminator, the same as ENDIF.

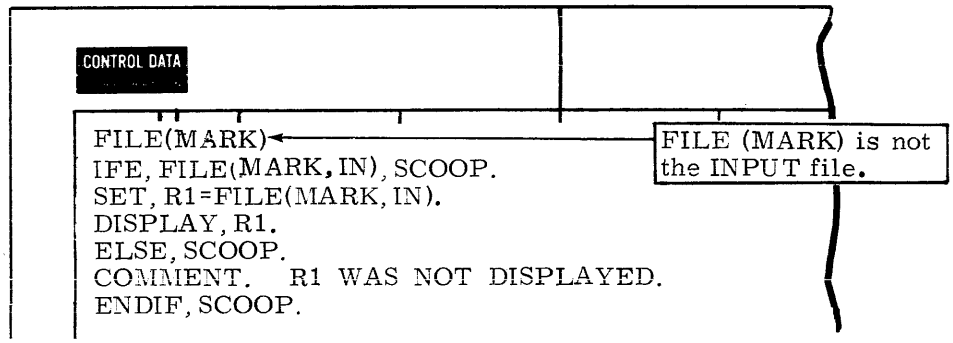
If the expression in an IFE statement is true, all of the statements following it are processed. If an ELSE statement is encountered with a label string matching the label string specified on the IFE statement, the ELSE statement initiates a skip to a terminating ENDIF statement with a matching label string. ELSE does not terminate skipping initiated by a SKIP statement or another ELSE statement.

The ELSE statement has the following format:

```
/ELSE.ls.
```

ls is the label string.

In effect, the ELSE statement makes the IFE statement more versatile by enabling the IFE statement to process one group of control statements if the IFE expression is true, and a different group of control statements if it is false. Example 13-12 illustrates the use of ELSE when the expression in the IFE statement is false.

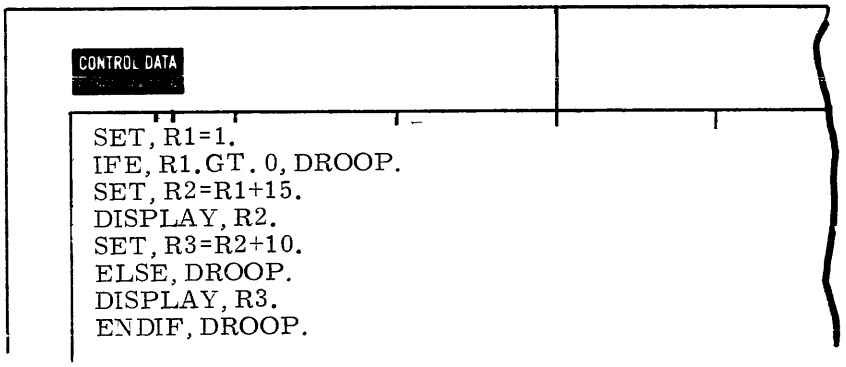


Example 13-12. Use of ELSE When Expression in IFE Statement is False

The entry in the dayfile is as follows:

```
FILE(MARK)
IFE, FILE(MARK, IN), SCOOP.
ELSE, SCOOP.
COMMENT.  R1 WAS NOT DISPLAYED.
ENDIF, SCOOP.
```

In Example 13-13, the IFE expression is true, so all of the statements are processed until the ELSE statement. R3 is not displayed.

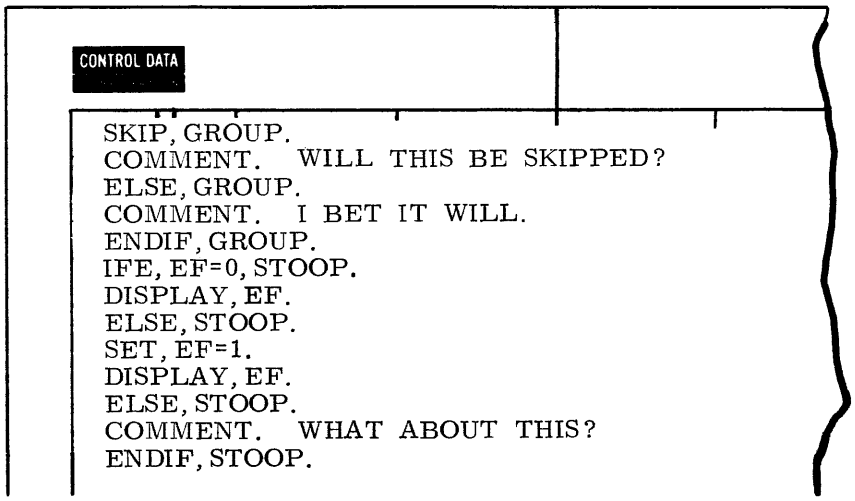


Example 13-13. Use of ELSE When Expression in IFE Statement is True

The entry in the dayfile is as follows:

```
SET, R1=1.
IFE, R1.GT. 0, DROOP.
SET, R2=R1+15.
DISPLAY, R2.
     16        20B
SET, R3=R2+10.
ELSE, DROOP.
ENDIF, DROOP.
```

Example 13-14 illustrates that ELSE does not end a SKIP or another ELSE.



Example 13-14. ELSE Does Not End a SKIP or Another ELSE

The entry in the dayfile as follows:

```
SKIP, GROUP.
ENDIF, GROUP.
IFE, EF=0, STOOP.
DISPLAY, EF.
        0          0B
ELSE, STOOP.
ENDIF, STOOP.
```

ENDIF

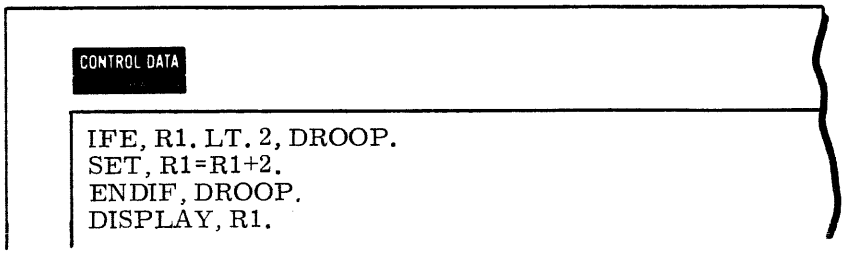
As you have seen in the examples for IFE, SKIP, and ELSE, the ENDIF statement terminates skipping initiated by these statements. If not terminating a skip, ENDIF is ignored but is printed in the dayfile. Again, the label string on ENDIF must match the label string on the statement that initiates skipping or the ENDIF statement is ignored.

The ENDIF statement has the following format:

```
ENDIF, ls.
```

ls is the label string.

In Example 13-15, if R1 is less than 2, it is increased by 2 and the new value is displayed. If R1 is not less than 2, control skips to the ENDIF statement and R1 will be displayed at its original value.



Example 13-15. Skipping Control to the ENDIF Statement

ITERATIVE STATEMENTS (WHILE, ENDW)

The iterative statements WHILE and ENDW bracket a group of control statements into a loop that can be repeatedly processed. WHILE begins the loop and ENDW ends the loop. The loop is repeated as long as the WHILE expression is true. If the expression is never true, control immediately skips to the ENDW statement. When an expression is false and no ENDW with a label string matching the WHILE statement is found, all the remaining statements in the control statement section are skipped.

The WHILE and ENDW statements have the following formats:

```
WHILE,exp,ls.
ENDW,ls.
```

exp is a CCL expression and ls is the label string.

When more than one WHILE/ENDW grouping is used within a job control stream or within a procedure, the label string field should be unique to prevent unpredictable control statement processing.

In Example 13-16, the WHILE expression is true and the loop repeats until R2 is not less than 5.

CONTROL DATA

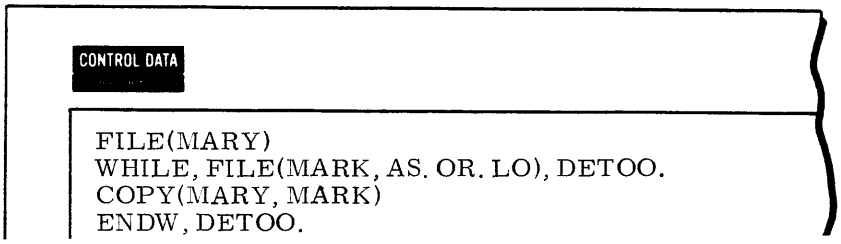
```
SET,R2=3.
WHILE,R2.LT.5,ART00.
DISPLAY,R2.
SET,R2=R2+1.
ENDW,ART00.
```

Example 13-16. WHILE Expression is True

The entry in the dayfile is as follows:

```
SET,R2=3.
WHILE,R2.LT.5,ART00.
DISPLAY,R2.
      3     3B
SET,R2=R2+1.
ENDW,ART00.
WHILE,R2.LT.5,ART00.
DISPLAY,R2.
      4     4B
SET,R2=R2+1.
ENDW,ART00.
WHILE,R2.LT.5,ART00.
ENDW,ART00.
```

In Example 13-17, the expression in the WHILE statement is false. File MARK is not assigned to the job nor is it a local file. Control skips from the WHILE statement to the ENDW statement ignoring the control statement between.



Example 13-17. Expression in WHILE Statement is False

The entry in the dayfile is as follows:

```
FILE(MARY)
WHILE, FILE(MARK, AS. OR. LO), DETOO.
ENDW, DETOO.
```

PROCEDURES

A procedure is a group of control statements that exists on a file separate from the job control statement section. Procedure files afford two advantages:

- The user can identify common sequences of control statements and save them as generalized procedures.
- A complex sequence of control statements can be given a name. Then anyone can call it even if he or she does not know how the procedure is done.

A procedure is stored as a section on a file or as a partition on a library and is called by a CCL statement in a job. This is similar to the way a program calls a subroutine. Several procedures can reside on the same file, and the entire file is searched when a user calls a procedure on that file.

A procedure consists of a procedure header statement and a procedure body. The header statement names and begins the procedure. The body contains all the statements between the header statement and an end of section. The procedure body can contain procedure commands and data as well as control statements.

PROCEDURE HEADER STATEMENT (.PROC)

The procedure header statement, .PROC, starts a procedure, declares the name of the procedure, and specifies any formal keywords and their default values. Unless it contains an error, the header statement is not printed in the dayfile.

The separator following .PROC and between the parameters must be a comma. The terminator must be a period. The header statement can continue over more than one card or line if the last character on each card or line is a separator.

The .PROC statement has the following format:

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

pname is the name of the procedure. It must be 1 to 7 alphanumeric characters and can begin with either a numeric or alphabetic character. pname is a required parameter and cannot be BEGIN (BEGIN is the CCL call statement which is described later in this section).

p_i specifies a keyword and is an optional parameter that can be in one of the following forms:

fk	Formal keyword
fk=default	Formal keyword with default
fk=	Formal keyword with null default

A sample header statement might look like this:

```
. PROC, COLLEEN, A, B, C=X.
```

In this header statement, COLLEEN is the name of the procedure. A and B are formal keywords in the form fk. C=X is a formal keyword with a default value assigned.

The keyword parameter (fk) enables the user to substitute different values for the formal keyword. In other words, formal keyword parameters can be passed from the control statement section of the job to the procedure file. This is similar to the way a FORTRAN program passes formal parameters from the main program to a subroutine. Examples of passing parameters is shown in the section titled Keyword Substitution.

fk is its own default value unless the forms fk=default or fk= are used. Formal keywords must be 1 to 10 alphanumeric characters. The default value can be 1 to 40 alphanumeric characters. The value of the default can be a $-delimited character string. The number of keywords is limited to an installation-defined value. The default is 50.

A formal keyword that is $-delimited must contain only alphanumeric characters. Substitution does not occur if it contains special characters.

fk can also be set equal to two special defaults, ≡FILE and ≡DATA. When the user specifies fk= ≡FILE in the .PROC statement, CCL reads the data in the section after the procedure. If the procedure resides on a library, CCL automatically changes ≡FILE to ≡LIB during substitution. Hence, the user can check if a file is on a library since the dayfile shows the ≡LIB substitution. If the file is not on a library, the file name is substituted for ≡FILE.

The default file can be accessed when the user specifies fk= ≡DATA. When .DATA is used, CCL creates a default temporary file to store the data. The .DATA command (explained later in this section) writes data to this temporary file. Substitution from the BEGIN statement can occur when the user specifies fk= ≡DATA.

Examples of the ≡DATA and ≡FILE defaults are shown with the description of the procedure commands (explained later in this section).

PROCEDURE BODY

The procedure body consists of all statements which follow the procedure header statement. The procedure body can contain control statements, CCL statements, and calls to other procedures.

The procedure body can use any of the formal keywords specified on the procedure header statement. When a procedure is called, CCL scans the statements of the procedure body before processing. The values specified by the parameters in the procedure call statement are substituted for the occurrences of the formal keywords in each statement of the procedure body. If a formal keyword is not indicated on the call statement, the default from the header statement replaces occurrences of the formal keyword. Keyword substitution is explained later in this section.

Procedure Call and Return

Since a procedure file is stored outside the job control statement section, it must be called by the job control statement section and control returned to the job control statement section after it has finished processing. This is accomplished by a call statement (BEGIN) and a REVERT statement. Figure 13-1 illustrates the procedure call and return.

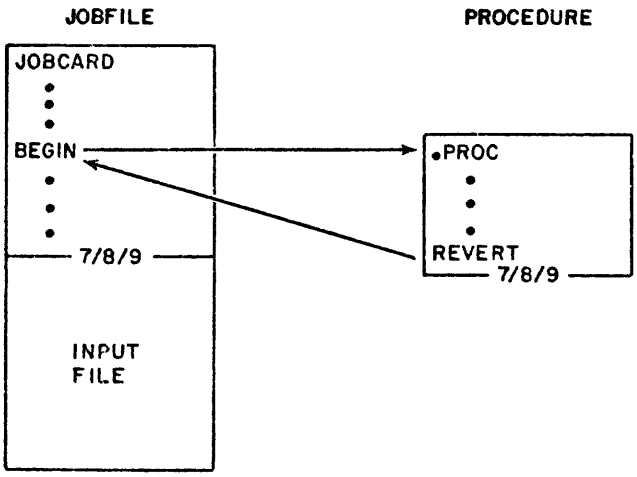


Figure 13-1. Calling a Procedure from a Job

A procedure can be called by another procedure as illustrated in Figure 13-2.

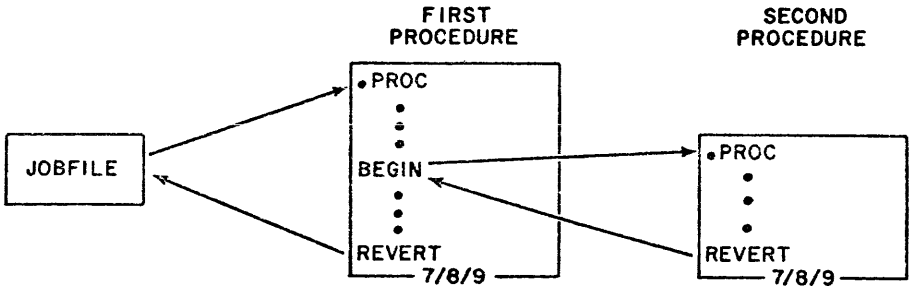


Figure 13-2. Calling a Procedure from Another Procedure

Procedure Call

The procedure call statement has two forms, as follows:

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

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

In both forms, the first separator can be a comma or a left parenthesis; the remaining separators must be commas. The terminator can be a period or a right parenthesis.

The second form is used only to call procedures on a library defined by the installation. If a user has access to library files, the second form (call-by-name form) is a convenient way to call the procedure.

pname is the procedure name. If the user does not specify a procedure name, the default value is the next procedure on file pfile. If the file is at the end of information, CCL rewinds pfile and calls the first procedure.

pfile is the name of the file on which the procedure resides. The default is an installation-defined file name.

p_i is a formal keyword and is an optional parameter that has one of the following forms:

fk	fk is a formal keyword that is the same as a keyword used in the procedure header statement.
fk=	fk= indicates null substitution for the formal keyword in the procedure header statement.
v	v is any alphanumeric or literal value. It is 1 to 40 characters.
fk=v	The value of v is substituted for the formal keyword which appears in the procedure header statement.

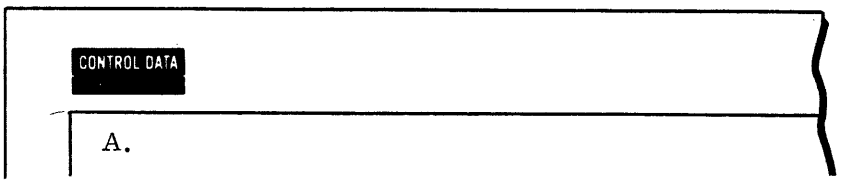
The following BEGIN statement calls procedure A from file MARK.

```
BEGIN,A,MARK.
```

If a second procedure B resides on file MARK after procedure A and is called after A is called, the following BEGIN statement calls procedure B.

```
BEGIN,,MARK.
```

On a library, the call-by-name statement is sufficient, as in Example 13-18 calling procedure A.



Example 13-18. Calling Procedure A From a Library

NOTE

If the procedure file has the same name as a binary file, either can be called as a result of the call-by-name statement.

Procedure Return

CCL provides an implicit REVERT sequence. Usually at the end of a procedure, CCL adds to the dayfile the following message, indicating a normal return.

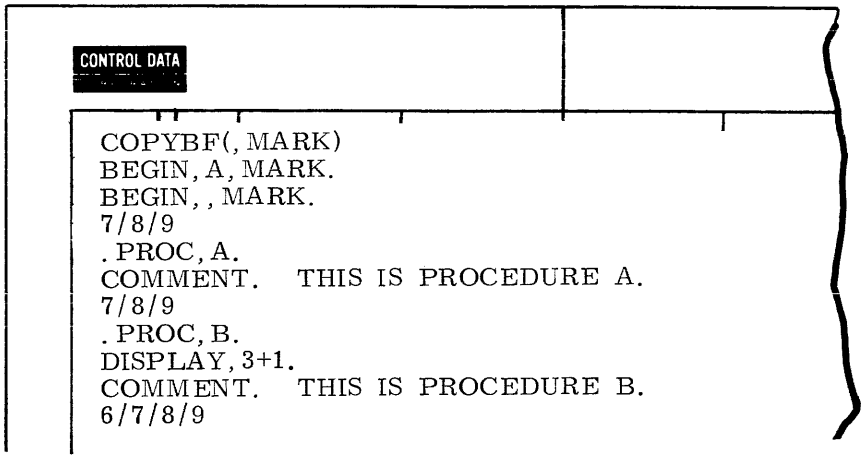
```
REVERT.CCL
```

However, if a fatal error is made in processing, CCL adds the following sequence of messages after the last processed statement of a procedure.

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

After a fatal error occurs in a statement, the statements following it are not processed, and the system searches for an EXIT statement. CCL issues the EXIT,S.CCL statement to stop skipping and then issues the REVERT,ABORT.CCL statement.

In Example 13-19, there is no REVERT statement, but REVERT.CCL is listed on the dayfile messages, signaling the end of the procedures. Procedures A and B both reside on file MARK, with procedure B immediately following procedure A.

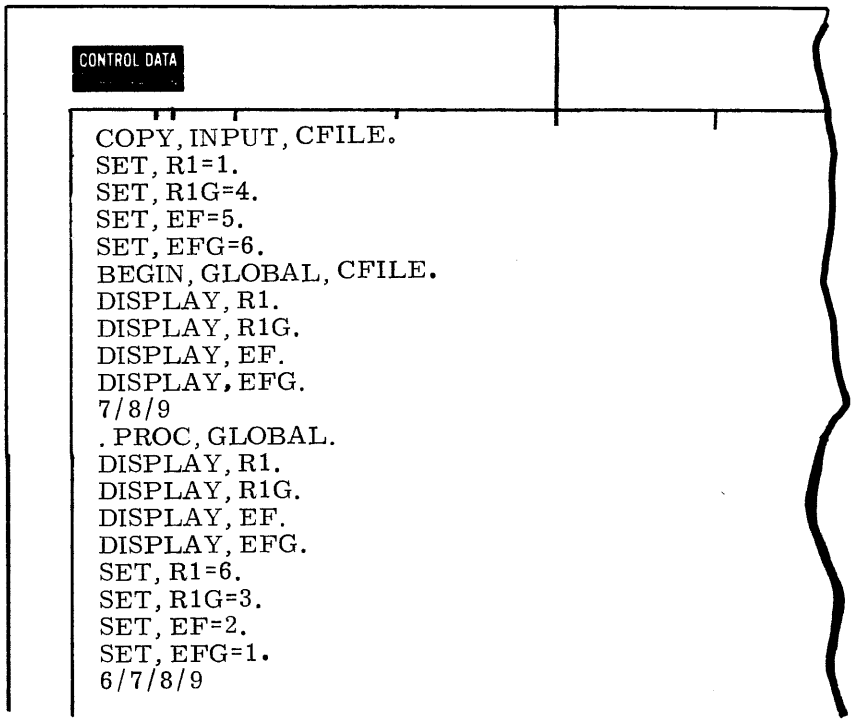


Example 13-19. Implicit REVERT Sequence by CCL

The dayfile entry is as follows:

```
COPYBF(,MARK)
  UT030 - COPY COMPLETE
  UT035 -     EOR - 5     EOS - 2     EOP - 1
BEGIN,A,MARK.
COMMENT. THIS IS PROCEDURE A.
REVERT.CCL
BEGIN,,MARK.
DISPLAY,3+1.
      4     4B
COMMENT. THIS IS PROCEDURE B.
REVERT.CCL
```

Example 13-20 shows that values set in R1G and EFG can be passed from the procedure file GLOBAL to the control statement section of the job. The values set in R1 and EF are passed from the job control statement section to the procedure, but not back from the procedure to the job control statement section.



Example 13-20. Values Passed from Procedure File GLOBAL to Control Statement Section

The dayfile entry is as follows:

```
      :
      :
SET,R1=1.
SET,R1G=4.
SET,EF=5.
SET,EFG=6.
BEGIN,GLOBAL,CFILE.
DISPLAY,R1.
      1      1B
DISPLAY,R1G.
      4      4B
DISPLAY,EF.
      5      5B
DISPLAY,EFG.
      6      6B
SET,R1=6.
SET,R1G=3.
SET,EF=2.
SET,EFG=1.
REVERT.CCL
DISPLAY,R1.
      1      1B
DISPLAY,R1G.
      3      3B
DISPLAY,EF.
      5      5B.
DISPLAY,EFG.
      1      1B
```

The user can also insert a REVERT statement in the procedure. REVERT is most commonly used with a conditional statement to return prematurely to the calling job or procedure.

The REVERT statement has the following formats:

```
REVERT.
```

```
REVERT,ABORT.
```

In both forms, processing returns to the calling job or procedure, except that REVERT, ABORT. tells CCL to abort instead of perform a normal exit.

A user can use an EXIT statement to create his own REVERT sequence. Using an EXIT statement with a REVERT statement creates a condition similar to the IFE statement used with an ELSE statement.

Example 13-21 illustrates the use of the REVERT., EXIT, S., and REVERT, ABORT. statements.

In Example 13-21, Procedure FIRST attempts to get permanent file FIRSTFI and copy it on the OUTPUT file. If successful, control returns to the job control section that called the procedure. If unsuccessful (the file is not located), a fatal error results and control skips to the EXIT, S. statement. The procedure ALTERN is called and it attempts to get permanent file ALTFILE and copy it on the OUTPUT file. If successful, control returns to the end of procedure FIRST. If not, control skips to the EXIT, S. statement and the REVERT, ABORT. statement is processed.

CONTROL DATA

```
COPY,,FILEA.
BEGIN,FIRST,FILEA.
7/8/9
.PROC,FIRST.
GETPF(FIRSTFI,FIRSTFI,ID=FO)
COPYSBF(FIRSTFI,)
REVERT.
EXIT,S.
BEGIN,ALTERN,FILEA.
7/8/9
.PROC,ALTERN.
GETPF(ALTFILE,ALTFILE,ID=CO)
COPYSBF(ALTFILE,)
REVERT.
EXIT,S.
REVERT,ABORT.
6/7/8/9
```

Example 13-21. User's REVERT Sequence

If the file FIRSTFI is not located and the alternate file, ALTFILE, is located, the resulting dayfile is as follows:

```
      :
      :
BEGIN,FIRST,FILEA.
GETPF(FIRSTFI,FIRSTFI,ID=FO)
COPYSBF(FIRSTFI,)
      :
      :
JOB DROPPED.
EXIT,S.
BEGIN,ALTERN,FILEA.
GETPF(ALTFILE,ALTFILE,ID=CO)
COPYSBF(ALTFILE,)
      :
      :
 UT031 - EOI ENCOUNTERED
 UT035 -          EOR - 4       EOS - 1       EOP - 1
REVERT.
REVERT.CCL
```

If the file FIRSTFI is not located and the alternate file ALTFILE, is also not located, the resulting dayfile is as follows:

```
      :
      :
BEGIN,FIRST,FILEA.
GETPF(FIRSTFI,FIRSTFI,ID=FO)
COPYSBF(FIRSTFI,)
      :
      :
JOB DROPPED.
EXIT,S.
BEGIN,ALTERN,FILEA.
GETPF(ALTFILE,ALTFILE,ID=AO)
COPYSBF(ALTFILE,)
      :
      :
JOB DROPPED.
EXIT,S.
REVERT,ABORT.
  SC031 - JOB ABORTED
EXIT,S.CCL
REVERT,ABORT.CCL
```

If file FIRSTFI is located, procedure ALTERN is not called, as shown in the resulting dayfile.

```
      :
      :
BEGIN,FIRST,FILEA.
GETPF(FIRSTFI,FIRSTFI,ID=FO)
COPYSBF(FIRSTFI,)
      :
      :
  UT031 - EOI ENCOUNTERED
  UT035 -          EOR - 4       EOS - 1       EOP - 1
REVERT.
REVERT.CCL
```

KEYWORD SUBSTITUTION

Once a user has created a procedure, he can decide which keyword substitutions to make in the procedure body each time it is called. The substitution values are included on the calling statement, or with a default value from the header statement.

In Example 13-22, procedure COLLEEN is on FILE1 and examples of call statements and the resulting substitutions are shown.

Procedure on File FILE1	Calls and Expansion	Explanation
.PROC,COLLEEN,A,B,C=X. COPYSBF(A,B,C)	BEGIN,COLLEEN,FILE1. yields COPYSBF(A,B,X)	All defaults from the header statement are used.
	BEGIN,COLLEEN,FILE1, R,S. yields COPYSBF(R,S,X)	R replaces A and S replaces B. The default for C is used.
	BEGIN,COLLEEN,FILE1, ,,D. yields COPYSBF(A,B,D)	The null parameters indicate A and B are to be used. D replaces X, the default value.

Example 13-22. Keyword Substitution

There are two modes by which CCL processes control statements: positional and equivalence. Positional mode was illustrated in the previous example where there was a one-to-one correspondence between call statement keywords and keywords on the header statement.

The permissible call statement parameters in positional mode are:

- A keyword identical to a keyword in the procedure header statement (can be $-delimited).
- Any alphanumeric or literal value.
- An omitted entry indicated by double commas or by the call list being shorter than the procedure list.

In equivalence mode, CCL matches each keyword in the call statement with the identical alphanumeric value in the procedure.

Processing always begins in positional mode. The switch from positional to equivalence mode can occur in the following way:

The form fk= or fk=v appears in the call list (BEGIN statement). This initiates equivalence mode for that position in the header statement and all those following in the call list.

Once processing switches from positional to equivalence mode, it remains in equivalence mode until the end of the list. It is impossible to return to positional mode.

In Example 13-23, equivalence mode is illustrated and also the switch from positional to equivalence mode.

Procedure on File FILE1	Calls and Expansion	Explanation
. PROC, COLLEEN, A, B, C=X REWIND(A, B, C)	BEGIN, COLLEEN, FILE1, APPLE, BEAR, CAT yields REWIND(APPLE, BEAR, CAT)	Parameter processing remains in positional mode.
	BEGIN, COLLEEN, FILE1, B=BAT. yields REWIND(A, BAT, X)	Equivalence mode is entered at once with B=BAT. A and C assume their default values.
	BEGIN, COLLEEN, FILE1, ANT, C=CUT, B=BELL yields REWIND(ANT, BELL, CUT)	ANT replaces A in positional mode. Equivalence mode is entered with C=CUT. B=BELL must be specified as an equivalence.

Example 13-23. Equivalence Mode

The equivalence symbol, ≡ (# in ASCII), can be placed immediately before a keyword to inhibit substitution for that keyword.

In Example 13-24, placing the ≡ before the formal keywords CY and PW inhibits substitution of the cycle number 210 and the password CO.

CONTROL DATA

```
COPY, INPUT, GFILE.
BEGIN, GET, GFILE, 210, CO.
7/8/9
. PROC, GET, CY, PW.
ATTACH, LFN, PFN, ≡CY=CY, ≡PW=PW.
```

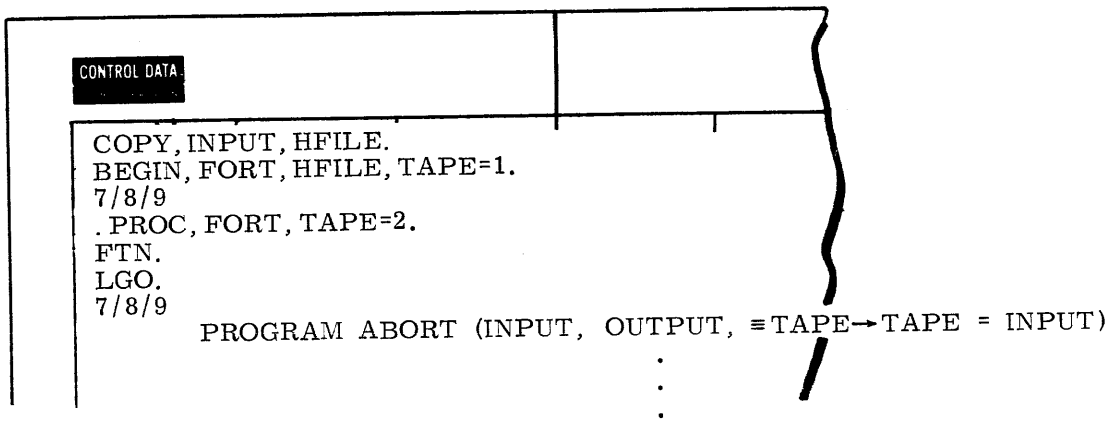
Example 13-24. Inhibit Substitution

The dayfile entry is as follows:

```
BEGIN, GET, 210, CO.
ATTACH, LFN, PFN, CY=210, PW=CO.
```

The right arrow, → (the underline character in ASCII), can be used within the procedure body to make a preliminary separation of two parameters which are combined after substitution takes place. The right arrow is called a linking character.

The linking character allows substitution without delimiters as shown in Example 13-25. TAPE is joined to 1 after substitution.



Example 13-25. Substitution Without Delimiters

The dayfile entry is as follows:

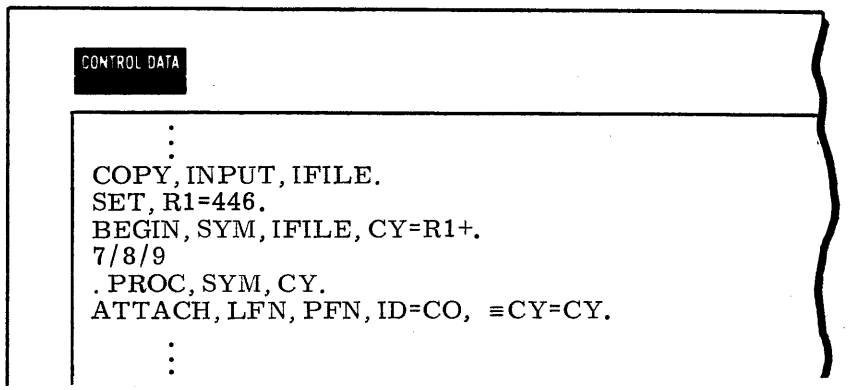
```
    :
BEGIN, FORT, HFILE, TAPE=1.
FTN.
LGO.
      PROGRAM ABORT (INPUT, OUTPUT, TAPE1 = INPUT)
                                    .
                                    .
                                    .
```

The numeric value of a symbolic name can be substituted for a formal keyword in the procedure body by using one of the following forms:

fk=sym+ fk=sym+D	converts the numeric value to decimal
fk=sym+B	converts the numeric value to octal

fk is the formal keyword and sym is a symbolic name.

In Example 13-26 the value of R1 is substituted for the formal keyword, CY.



Example 13-26. Value of R1 Substituted for Formal Keyword CY

The entry in the dayfile is as follows:

```
SET,R1=446.
BEGIN,SYM,IFILE,CY=R1+.
ATTACH,LFN,PFN,ID=CO,CY=446.
```

PROCEDURE COMMANDS

Procedure commands enable the user to include a data file within a procedure and to insert documentary comments within a procedure. The commands must have a period in column 1 and the command name beginning in column 2. There is no terminator. The following procedure commands are described in this section.

- .DATA — Allows data to be stored within a procedure.
- .EOR — Writes an end of section on a data file.
- .EOF — Writes an end of partition on a data file.
- .* — Enables the user to include comments in a procedure that are not printed in the dayfile.

.DATA Command

The .DATA command separates control statements from data statements in a procedure. The .DATA command has the following formats:

```
.DATA
   or
.DATA,lfn
```

The first format writes data on a default file whose name is accessed by equivalencing a formal parameter with ≡DATA. The second format writes data on a logical file name (lfn) specified by the user.

If lfn already exists, CCL returns it and creates a new file. Therefore, the .DATA command cannot add data to an existing file. After data is written on the file (lfn or ≡DATA), CCL rewinds the file.

Examples 13-27 and 13-28 illustrate the use of the ≡DATA default used with the .DATA command. Both the procedure LIST and the data, which consists of a list of names, are on file LAST. All data following the .DATA command is written on the default temporary file ≡DATA. The≡DATA default tells the system to search for input from the temporary file.

In Example 13-27, the COPYSBF statement copies the names from the temporary file to the OUTPUT file.

CONTROL DATA

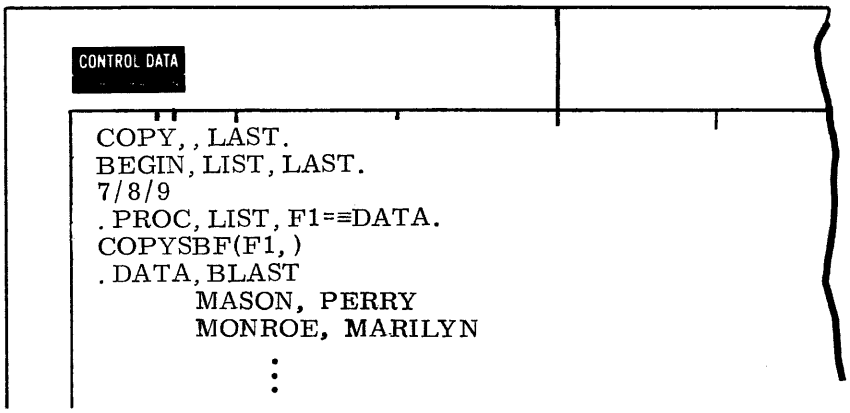
```
COPY,,LAST.
BEGIN,LIST,LAST.
7/8/9
.PROC,LIST,F2=≡DATA.
COPYSBF(F2,)
.DATA
    KING, MARY
    KRONK, DON
        :
```

Example 13-27. COPYSBF Statement Copies Names from Temporary File to OUTPUT File

On the dayfile entry, CCL names the temporary default file ZZCCLAA.

```
        :
BEGIN,LIST,LAST.
COPYSBF(ZZCCLAA,)
   UT031 - EOI ENCOUNTERED
        :
REVERT.CCL
```

In Example 13-28, the default temporary file is assigned the name BLAST.

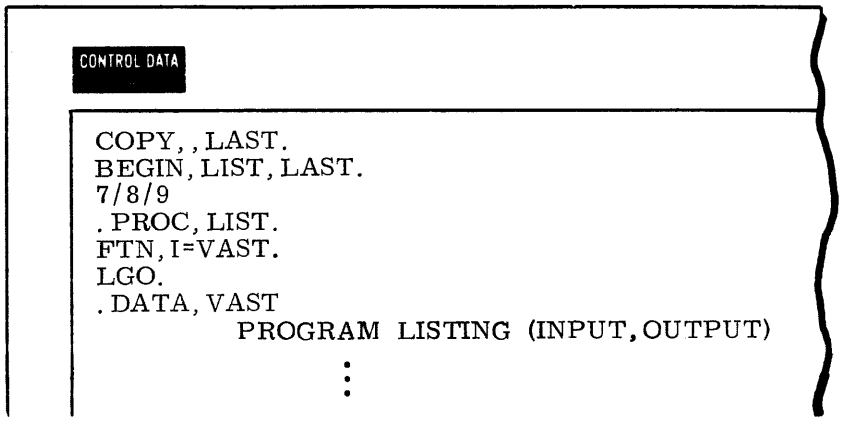


Example 13-28. The Default Temporary File is Assigned a Name

The dayfile entry is as follows:

```
        :
BEGIN,LIST,LAST.
COPYSBF(BLAST,)
   UT031 - EOI ENCOUNTERED
        :
REVERT.CCL
```

In Example 13-29, the input for a FORTRAN is on the procedure file. The file is named VAST.

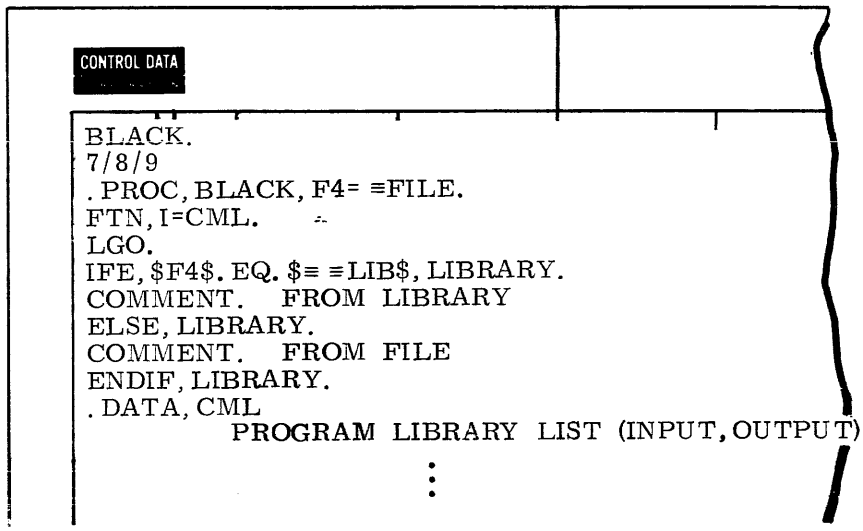


Example 13-29. FORTRAN Program is in the Procedure File Named VAST

The dayfile entry is as follows:

```
BEGIN,LIST,LAST.
FTN,I=VAST.
LGO.
        :
REVERT.CCL
```

In Example 13-30, the procedure BLACK resides on a user-created library PROCLIB. When the procedure is called with the call-by-name statement, the FORTRAN compiler looks for input (source code) from the file CML. ≡LIB is substituted for ≡FILE in the dayfile entry. Therefore, the comment, FROM LIBRARY, is displayed as a result of the IFE statement.



Example 13-30. Call-by-Name Statement to Call Procedure from a Library

The dayfile entry is as follows:

```
BLACK.
FTN.I=CML.
LGO.
IFE,$ ≡LIB$.EQ.$ ≡ LIB$,LIBRARY.
COMMENT. FROM LIBRARY
ELSE,LIBRARY.
ENDIF,LIBRARY.
REVERT.CCL
```

.EOR Command

The .EOR command writes an end of section on the data file specified by a .DATA command. After a .EOR command, statements are written onto a new section. EOR is used instead of EOS for compatibility with other systems. .EOR can be used in place of .DATA where substitution in the section following the .EOR is not desired.

The .EOR command has the following format:

```
.EOR
```

.EOF Command

The .EOF command writes an end of partition on the data file specified by a .DATA command. After an .EOF command, statements are written on a new partition. EOF is used instead of EOP for compatibility with other systems. .EOF can be used in place of .DATA where substitution in the partition following the .EOF is not desired.

The .EOF command has the following format:

```
.EOF
```

.* Command

A .* command enables the user to document a procedure with internal comments. These comments do not appear in the dayfile. After the .*, the comment can contain any combination of characters. The number of characters including the .* cannot exceed 80 after substitution.

The .* command has the following format:

```
.* comment
```

Example 13-31 illustrates all of the procedure commands. On a list of customers to be billed, the billing date has to be changed each month. Using a procedure file with a keyword substitution for the date, this is done very easily. The procedure and data are on file AFILE. The date March 31, 1978 is substituted for the original date on the procedure header statement.

CONTROL DATA

```
COPY(, AFILE)
BEGIN, BILLING, AFILE, D2=$MARCH 31, 1978$.
7/8/9
.PROC, BILLING, D1= ≡DATA, D2=$JANUARY 31, 1978$.
COPY(D1, )
.*THIS COMMENT CAN DOCUMENT ANY PROCEDURE INTERNALLY.
.DATA
    KING, MARY
    7700 TAYLOR STREET
    MPLS, MN  55432
      756-3721
    BILLING DATE = D2
.EOR
    KRONK, DON
    5630 CRESTVIEW DRIVE
    ST. ANTHONY, CT  06421
      343-6691
    BILLING DATE = D2
.EOF
```

Example 13-31. All Procedure Commands

The entry in the dayfile is as follows. ≡DATA is given the file name ZZCCLAA.

```
        :
        .
BEGIN, BILLING, AFILE, D2=$MARCH 31, 1978$.
COPY (ZZCCLAA, )
  UT031 - EOI ENCOUNTERED
  UT035 -     EOR - 10     EOS - 2     EOP - 1
REVERT. CCL
```

The output is as follows:

```
KING, MARY
7700 TAYLOR STREET
MPLS, MN  55432
  756-3721
BILLING DATE = MARCH 31, 1978
KRONK, DON
5630 CRESTVIEW DRIVE
ST. ANTHONY, CT 06421
  343-6691
BILLING DATE = MARCH 31, 1978
```

Example 13-32 illustrates the use of → in rewinding a data file. CCL creates the file named FILE22 by concatenating the register value and FILE. The statements following .DATA are the procedure REW. The file containing procedure REW is created by the initial BEGIN, ONE, PFILE. The inhibit substitution character (≡) is removed. When the BEGIN, REW, F, FN=R1+. call is made in procedure ONE, the current value of R1 is passed and used when the filename is created by the concatenation process. Without the ≡ before the →, substitution for FN does not occur.

CONTROL DATA

```
COPY, INPUT, PFILE.
BEGIN, ONE, PFILE.
7/8/9
. PROC, ONE, F= ≡DATA.
SET, R1=22.
BEGIN, REW, F, FN=R1+.
REVERT.
. DATA
. PROC, REW, FN.
REWIND, FILE ≡ →FN.
6/7/8/9
```

Example 13-32. Use of → in Rewinding a Data File

The dayfile entry is as follows:

```
BEGIN, ONE, PFILE.
SET, R1=22.
BEGIN, REW, ZZCCLAA, FN=R1+.
REWIND, FILE22.
REVERT. CCL
REVERT.
```

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 11-8-2†††	52	12-8-7 or 11-0†††	21
V	V	26	0-5	25	0-5	56							
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 12-8-2†††	72	12-8-4 or 12-0†††	3C
0	0	33	0	12	0	30							
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 SCOPE 3.4 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

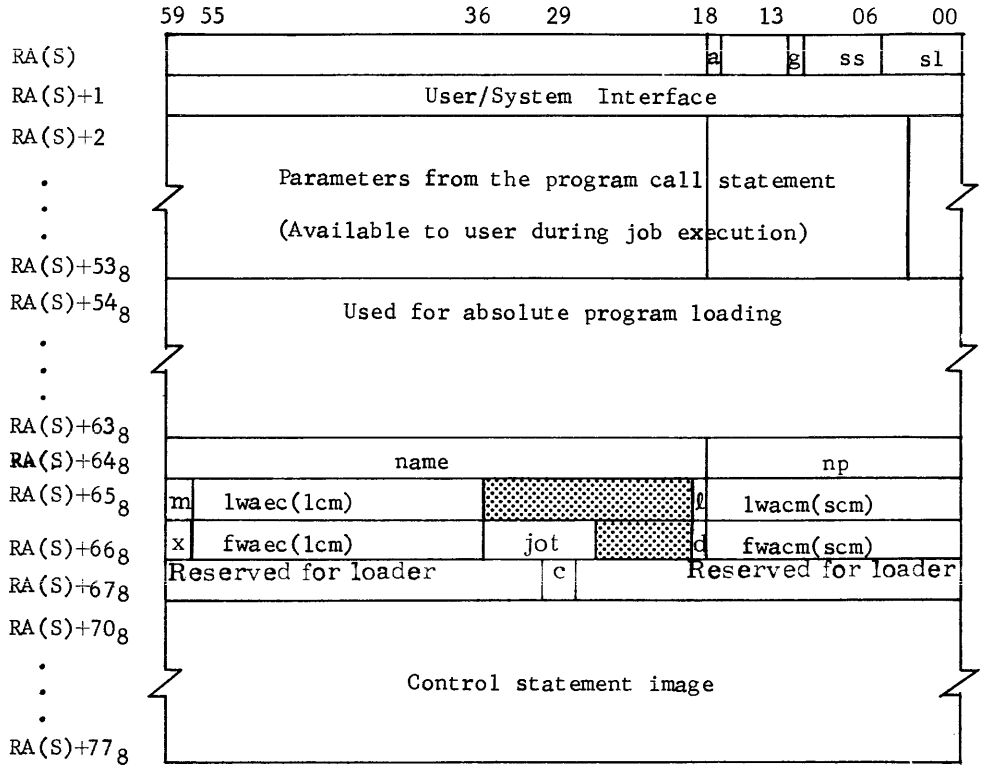
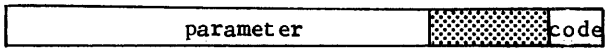


Figure B-1. Job Communication Area

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)+53_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	Reserved	
13	Reserved	
14	Reserved	
15	Reserved	
16	Other	
17	Termination	. or)

RA(S)+54_8 through RA(S)+63_8	59-00		Used for absolute program loading
RA(S)+64_8	59-18	name	Name of file from which program was loaded
	17-00	np	Number of locations used for parameters

Word	Bits	Field	Significance
RA(S)+65$_8$	59	m	Compare move unit† 0 No compare move unit 1 Compare move unit available
	58-36	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 sequential 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.
	35-24	jot	Job origin type 0 System or spot job 1 Local batch job 2 Remote batch job 3 Terminal job (connected I/O)
	23-19	none	Unused; zero
	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	control statement	Control statement

†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.

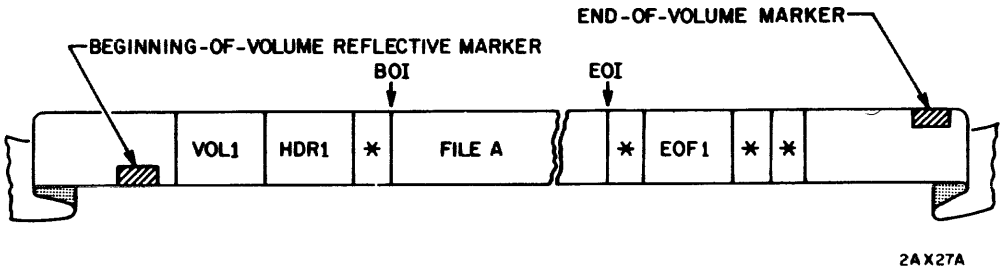


Figure C-1. Single Volume File

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.

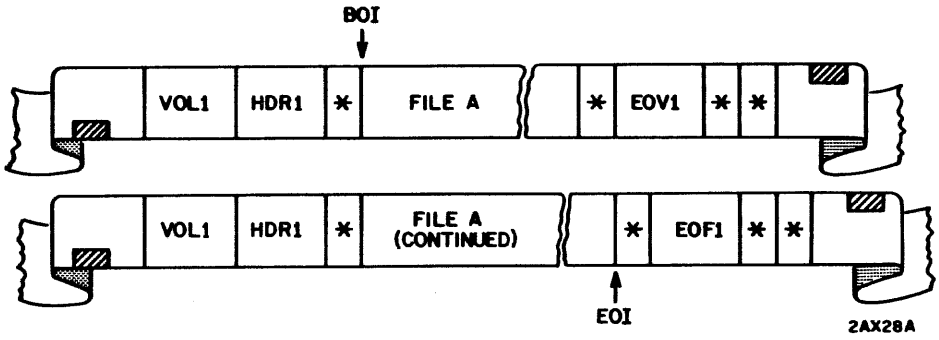


Figure C-2. Multivolume File

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 supported by SCOPE 2 for on-line tapes only.

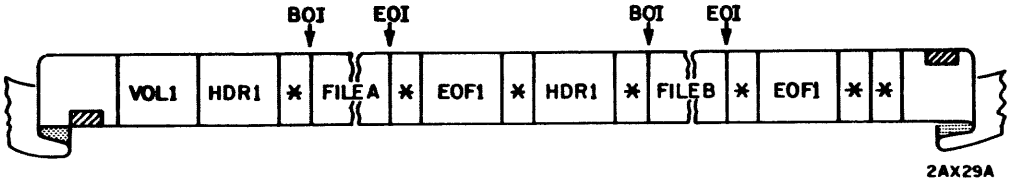


Figure C-3. Multifile Volume

Let us also consider the multivolume multifile set composed of several labeled files occupying several volumes (Figure C-4). This combination is allowed for on-line tapes under SCOPE 2.

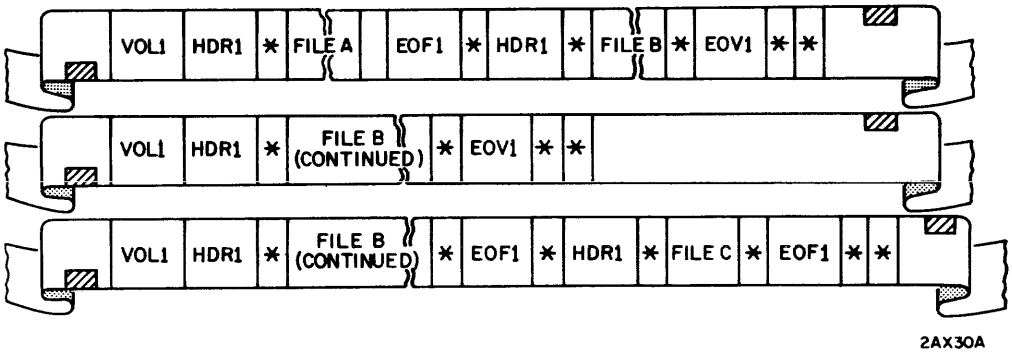


Figure C-4. Multifile Multivolume

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.

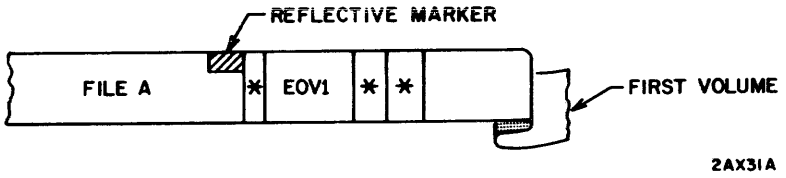


Figure C-5. Simultaneous EOI and EOV Marker

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:

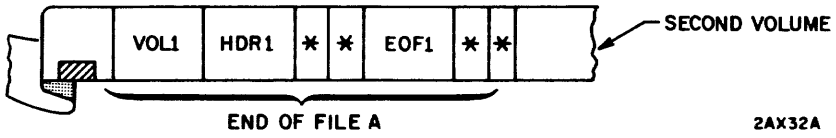


Figure C-6. Continuation of EOF Labeling

The double tapemark between the volume/header group and the end-of-file trailer group indicates that no further data appears within file A. This condition constitutes a multivolume 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 date 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 format 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	
22-27	Set identification	6	
28-31	File section number	4	
32-35	File sequence number	4	
36-39	Generation number (optional)	4	Same as the corresponding fields in the first file header label for this file
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.

A feature of W-records is that they may be constructed in portions. Each portion is prefaced with its own W-control word along with a flag (WCR) in the control word to indicate the portion is not a complete record but a partial record. The total record length then becomes the sum of the lengths of all the portions. The advantage is that W-type records can be constructed in portions without the RL being known at the time the first portion is constructed.

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.

S RECORDS

S records are also known as SCOPE 3.4 Standard. This is the only record type recognized by both SCOPE 3.3 and SCOPE 2.

Only one record type, S, is 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.

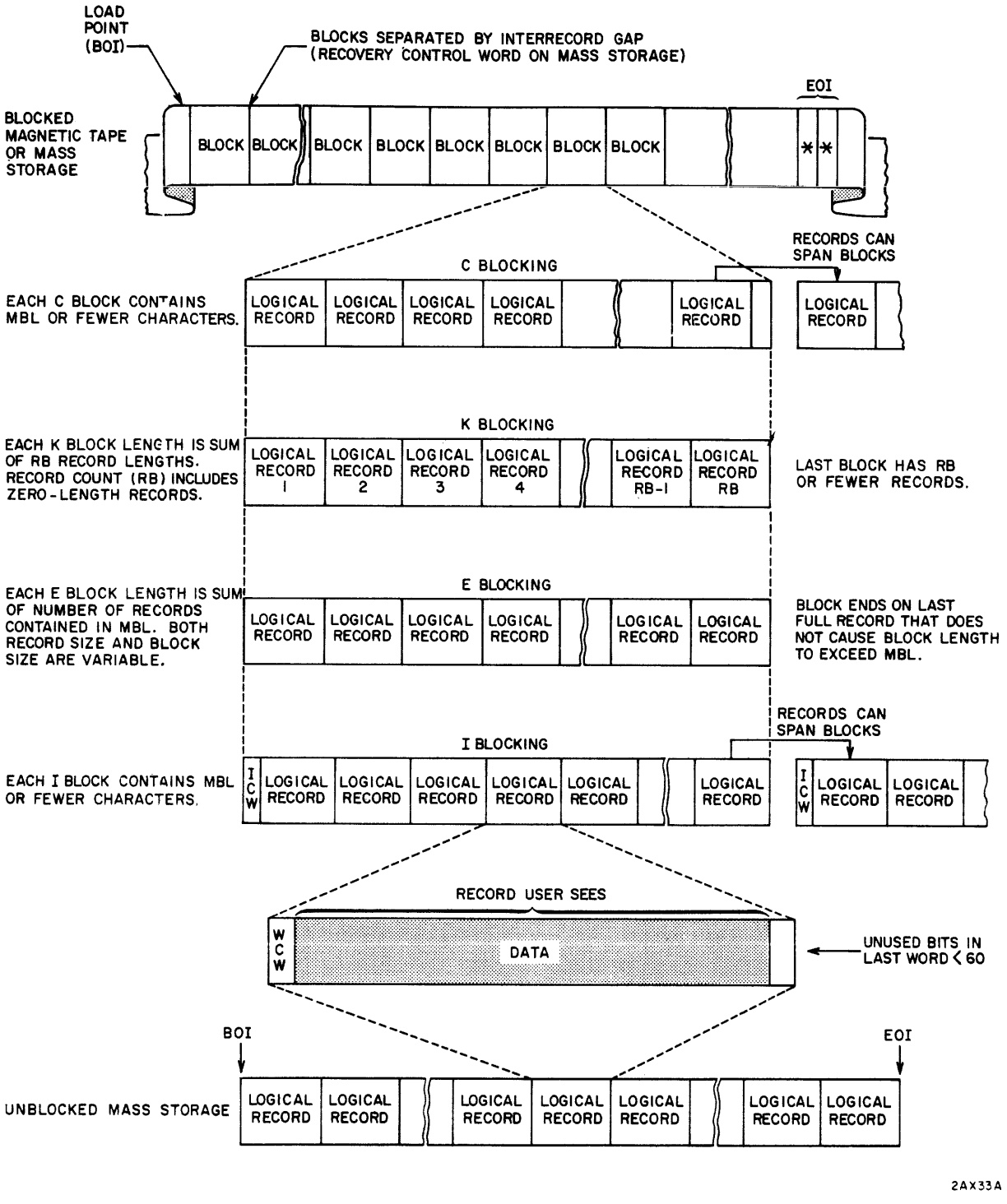


Figure D-1. W Record Format

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.

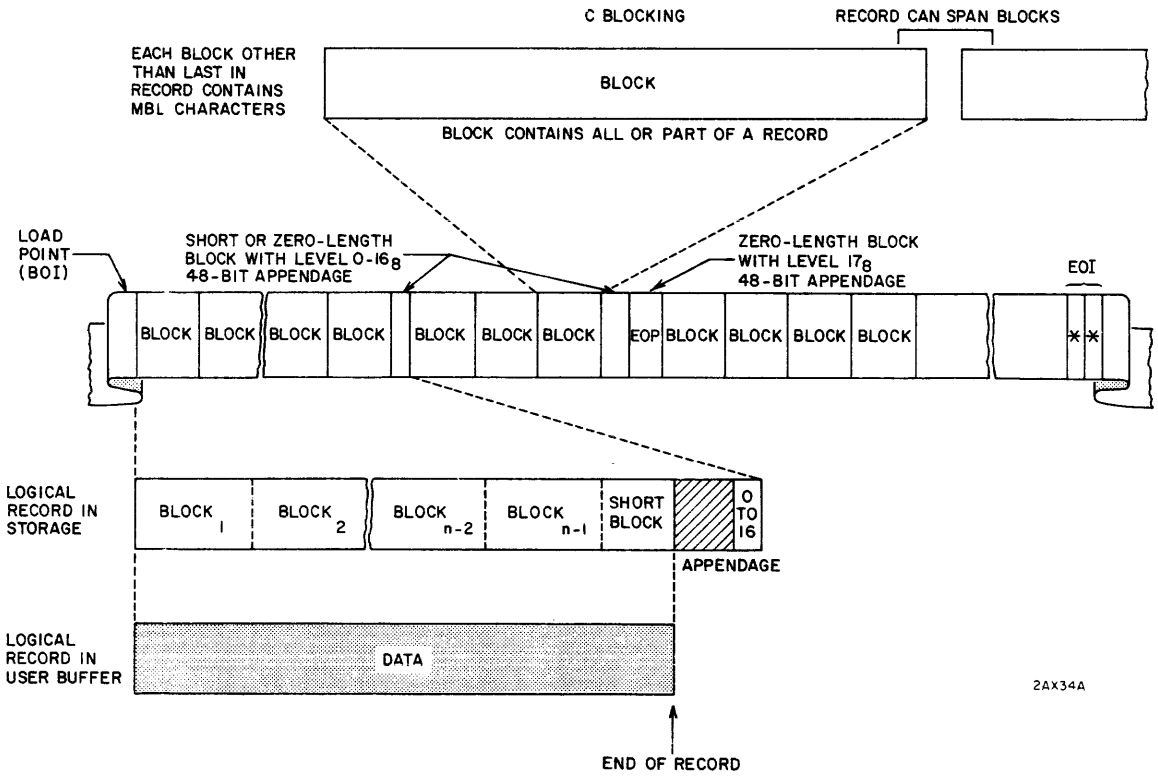


Figure D-2. S SCOPE Logical Record Format

Z RECORDS

The Z (zero byte) record type is commonly used in the SCOPE 3.4 systems 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, which are replaced by blanks when the record is read.

Consider first what happens on an output request such as a FORTRAN WRITE or BUFFER OUT statement. Referring to Figure D-3, 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.

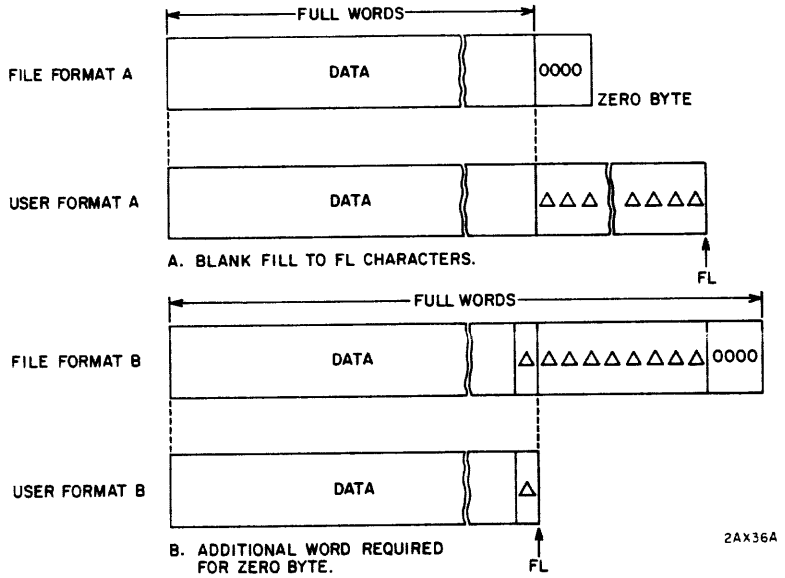


Figure D-3. Padding of Z Records

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 (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 premature 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.

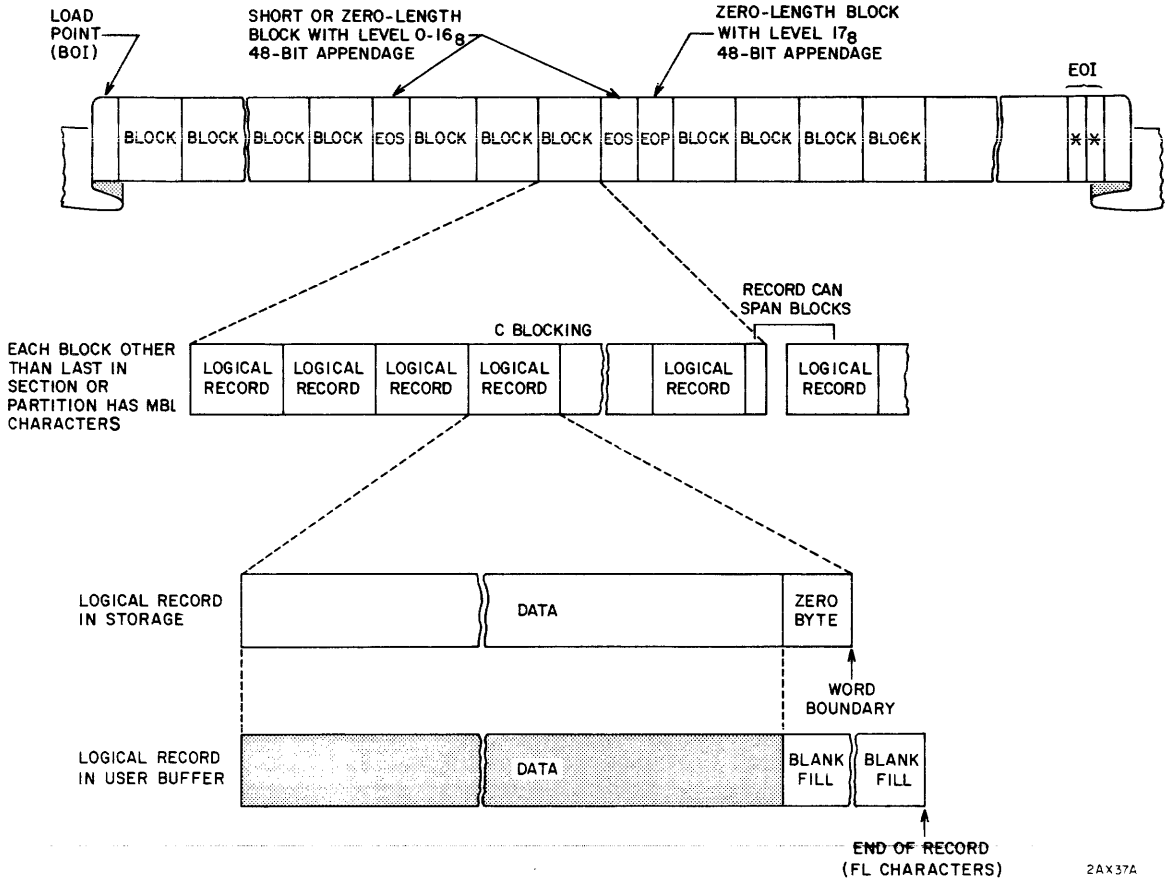


Figure D-4. Z Zero Byte Records with C Blocking

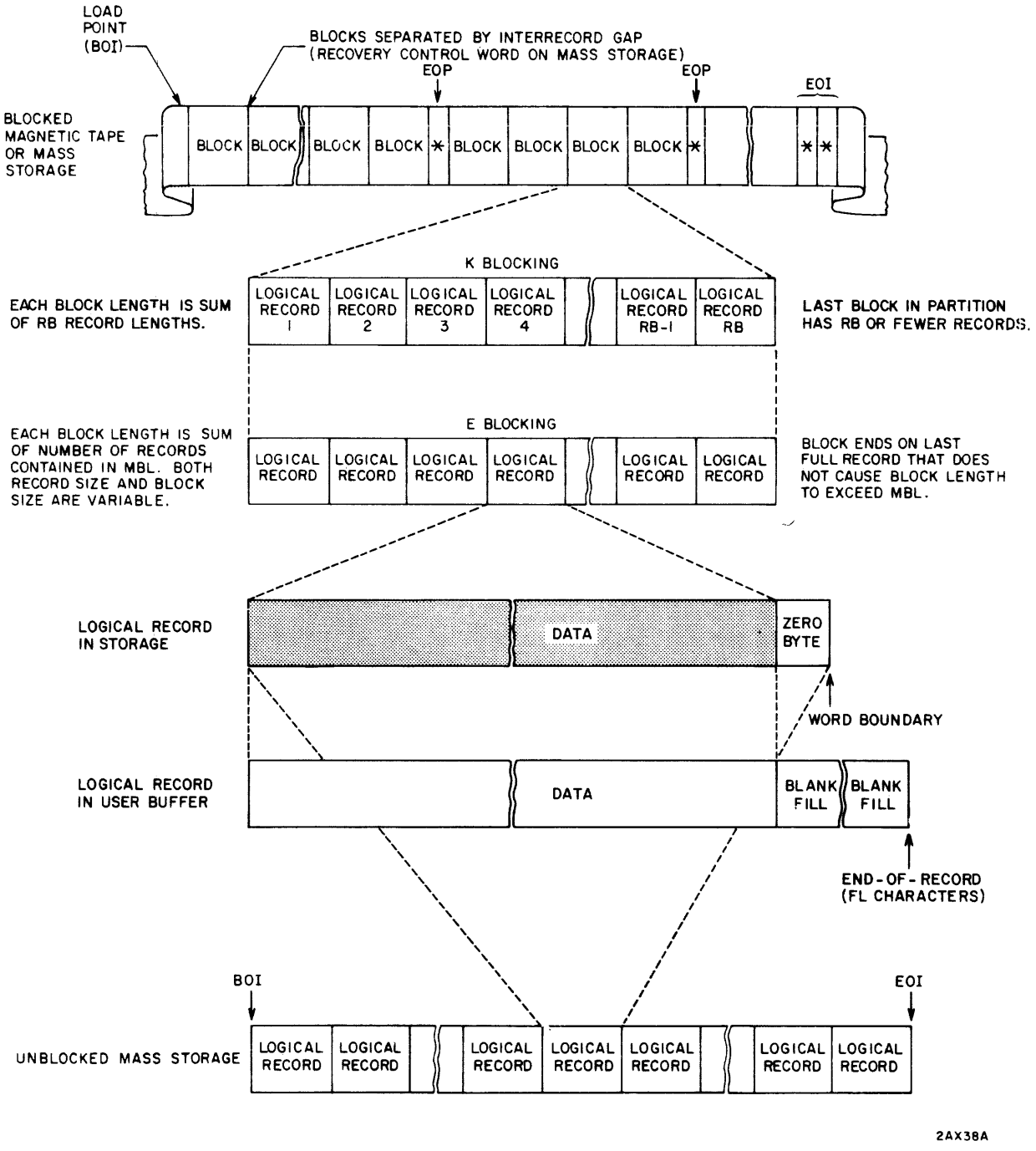


Figure D-5. Z Zero Byte Records Unblocked or with K and E Blocking

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.

On an output request such as a FORTRAN WRITE or BUFFER OUT using F records, 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.

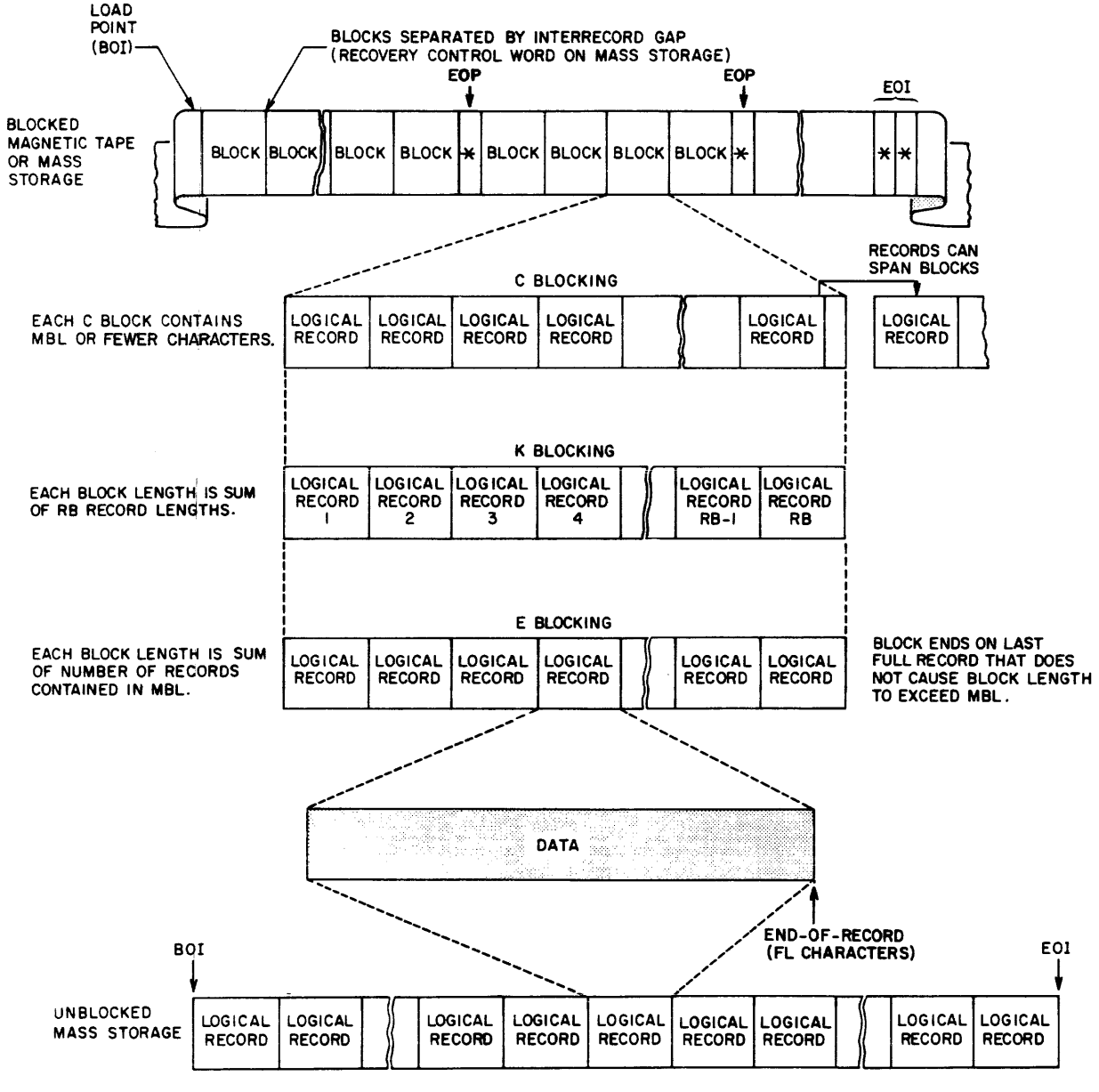


Figure D-6. F Fixed Length Record Format 2AX39A

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-7) 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.

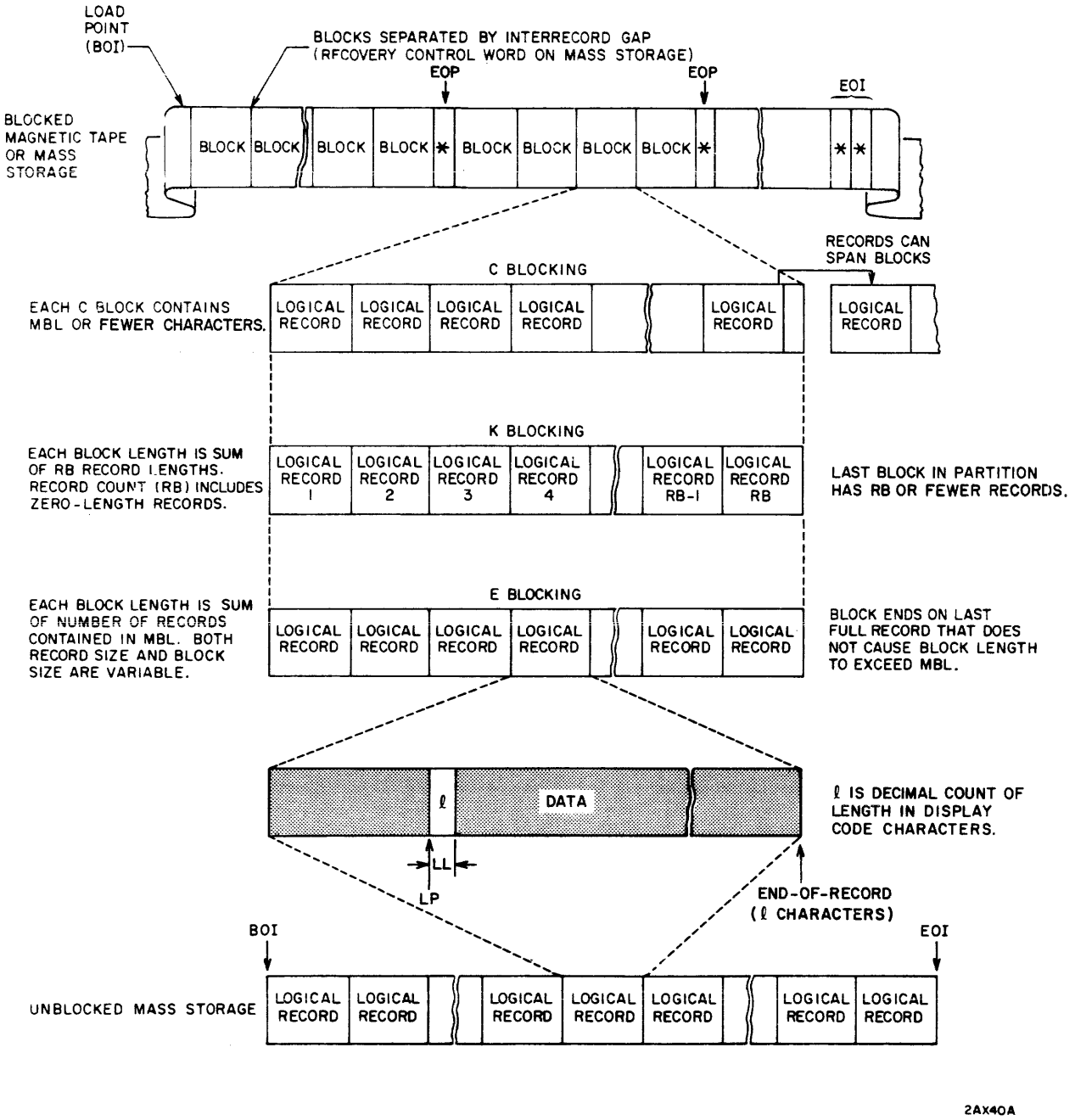


Figure D-7. Decimal Count Record Format

R RECORDS

Record mark (R) records (Figure D-8) 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 default record mark character is] .

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], 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.

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.

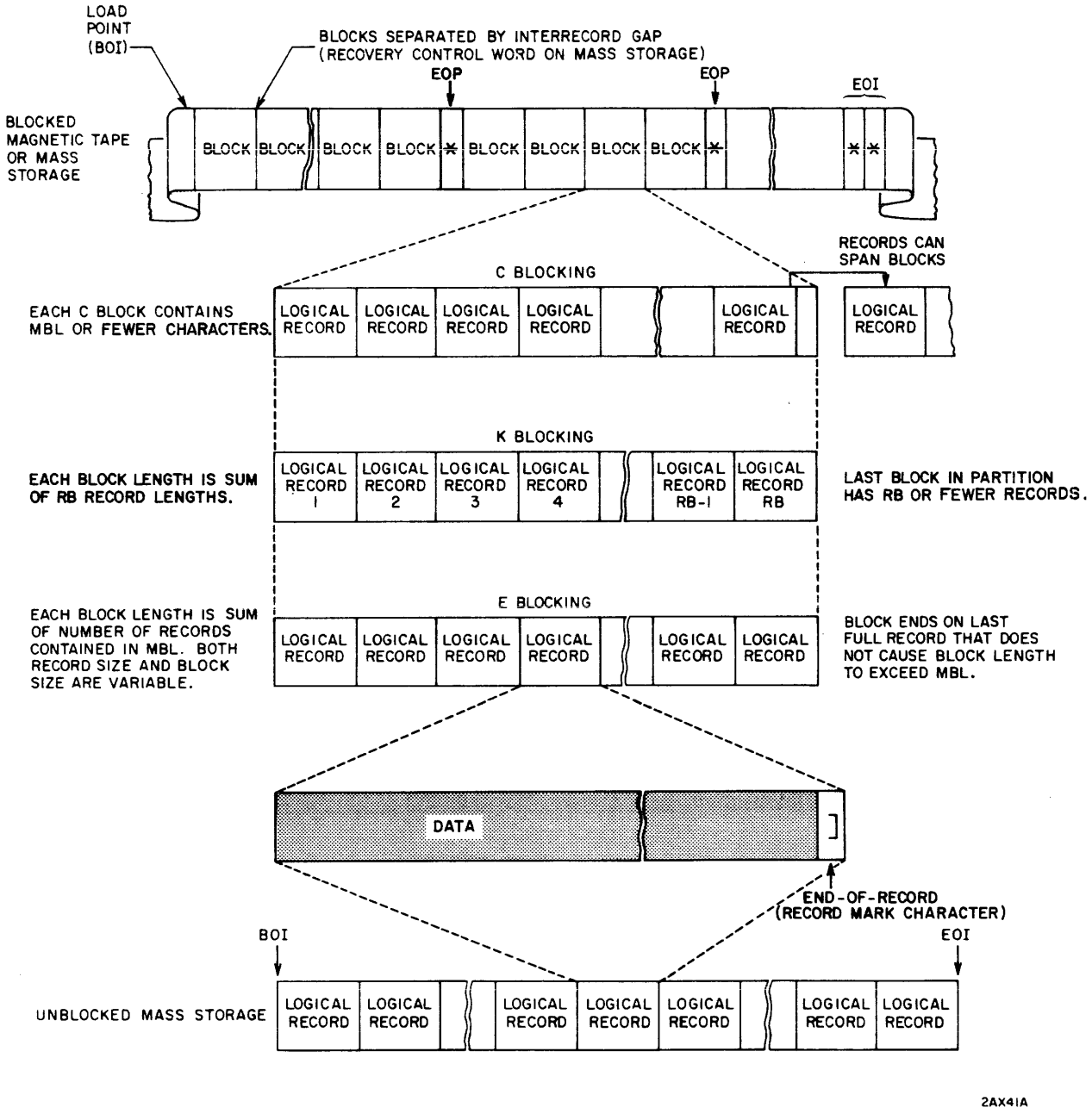


Figure D-8. R Record Mark Character Record Format

T RECORDS

Trailer (T) records (Figure D-9) 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. SCOPE 3.4 record manager allows blank fill; SCOPE 2 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.

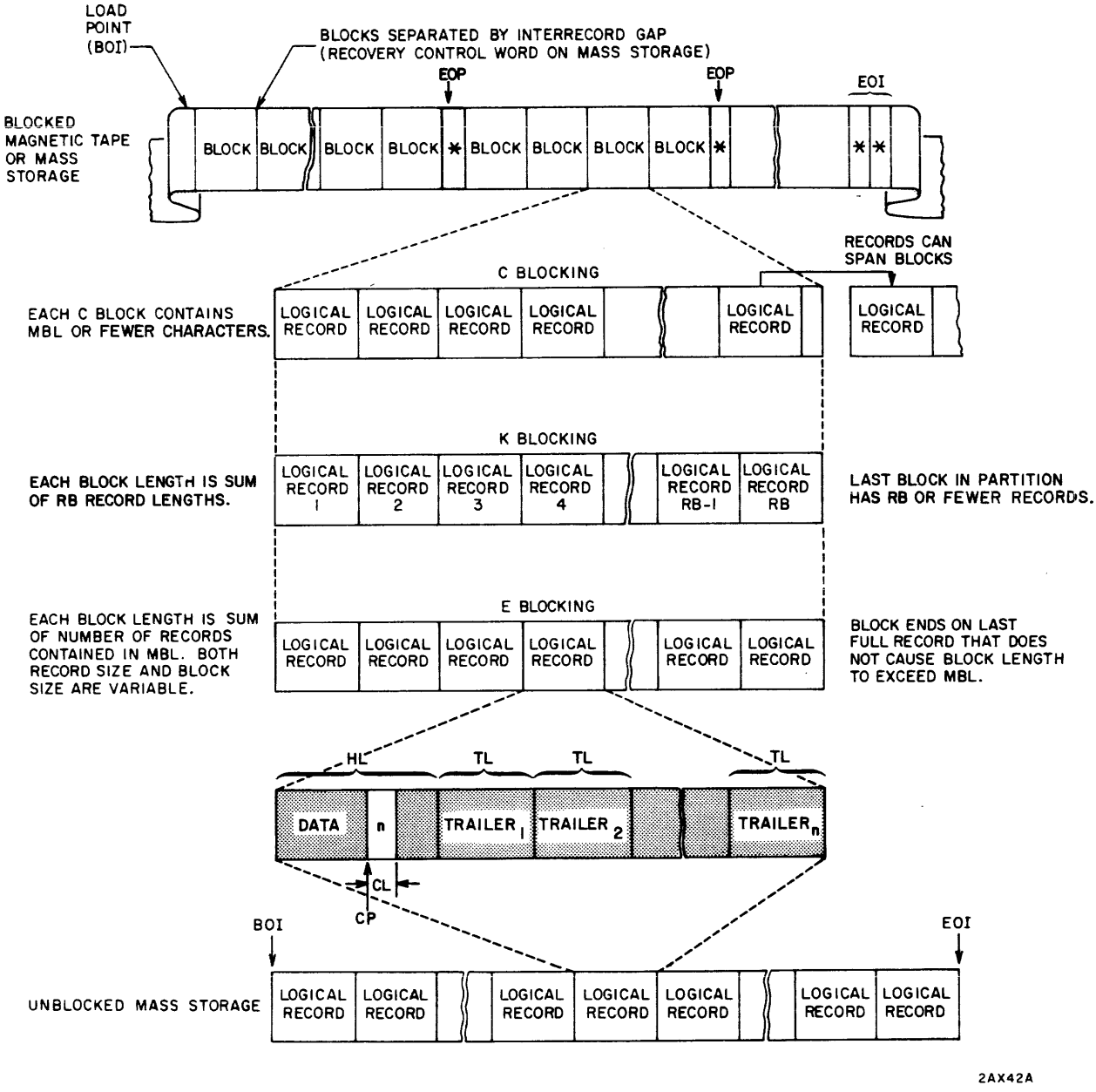


Figure D-9. T Trailer Count Record Format

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-10), 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 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.

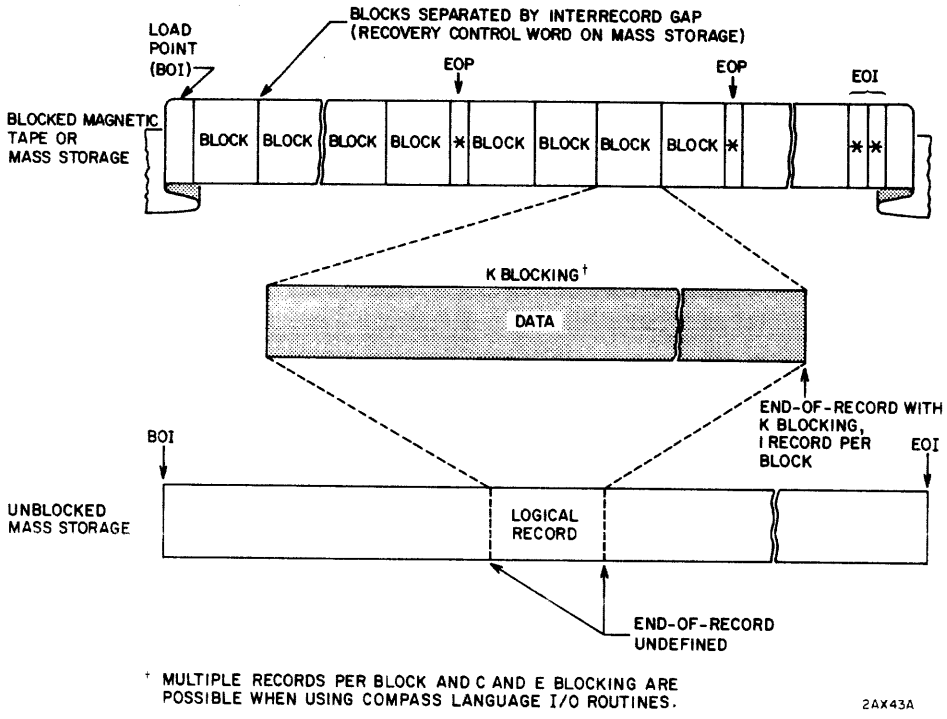


Figure D-10. U Undefined Record Format

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 memory management 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 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 input record type is 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.

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 only one file specifies MRL or FL, that value sets the size of the buffer for both files.

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)	1,5,10,11
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). The compare will fail if file delimiters do not match.

Usually, W records cannot be converted to S records because S records must be full words of data.

If input file contains a 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.

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. To specify unblocked, simply specify BT on the FILE statement.

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 necessary when 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 SCOPE 3.4.

10. On magnetic tape, W records can be recorded in binary mode only.

11. W continuation records are not supported for records with E-type blocking; MRL must be specified when a record of this type exceeds 5120 characters.

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 SCOPE 3.4 standard 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 SCOPE 3.4 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.

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 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 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 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,BT=E)	1,2,3
WA	Unblocked	FILE(lfn,FO=WA,RT=R,RMK=char)	1,5

Notes and Rules

1. If RMK is not specified, it defaults to 62_8 which is a display code] .

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 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 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 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

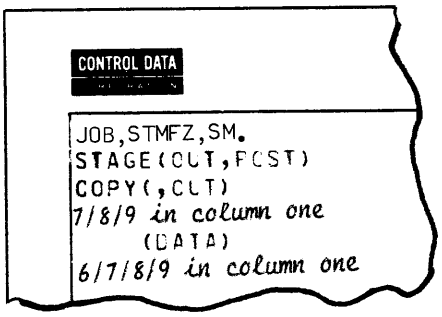
   Output: Two tapemarks

   On blocked mass storage, a tapemark is maintained as a recovery control word.

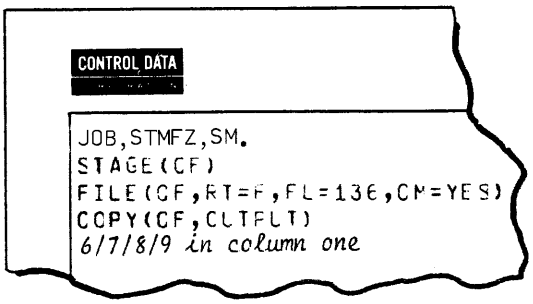
   Section delimiters are lost if they are on the input file. Partition delimiters on the input file are lost if the output file type does not support partitions (not W or not blocked sequential).

CONVERSION EXAMPLES

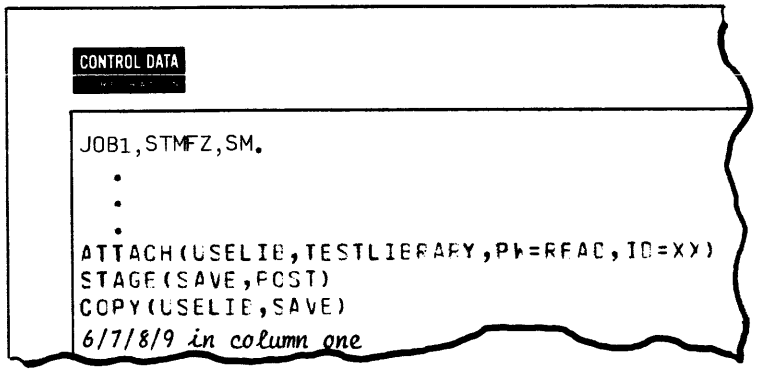
1. A very common application of COPY is to block or deblock W record files. This example illustrates a job that copies data on INPUT to a magnetic tape file (it performs a card-to-tape operation). Neither file requires a FILE statement because defaults apply. The only difference between file descriptions 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.

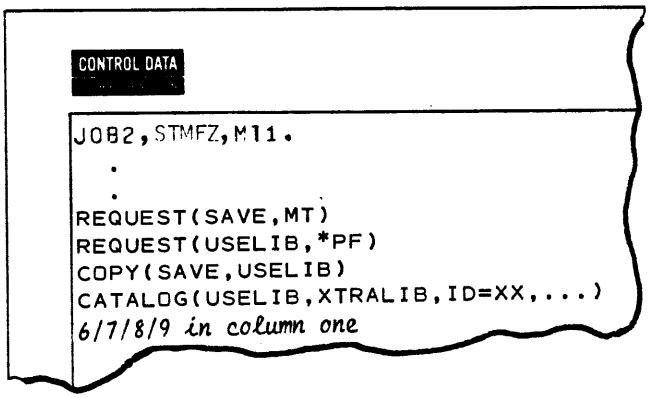


2. Example 2 illustrates how to print from an external-coded, even-parity tape recorded one print line per block (Industry standard unblocked). Each print line is 136 characters; the first character of which is a printer control character. Tape density is 556 bits per inch.

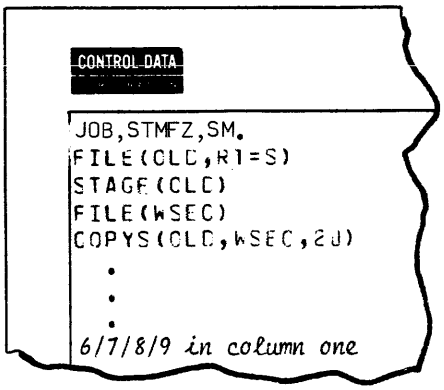


3. In this example, library USELIB is saved on staged magnetic tape and later loaded on-line, perhaps at another site. USELIB is an unblocked W format sequential file. The first job blocks an unblocked file. The process is reversed in the second job; the file must be unblocked before the loader can load from it.

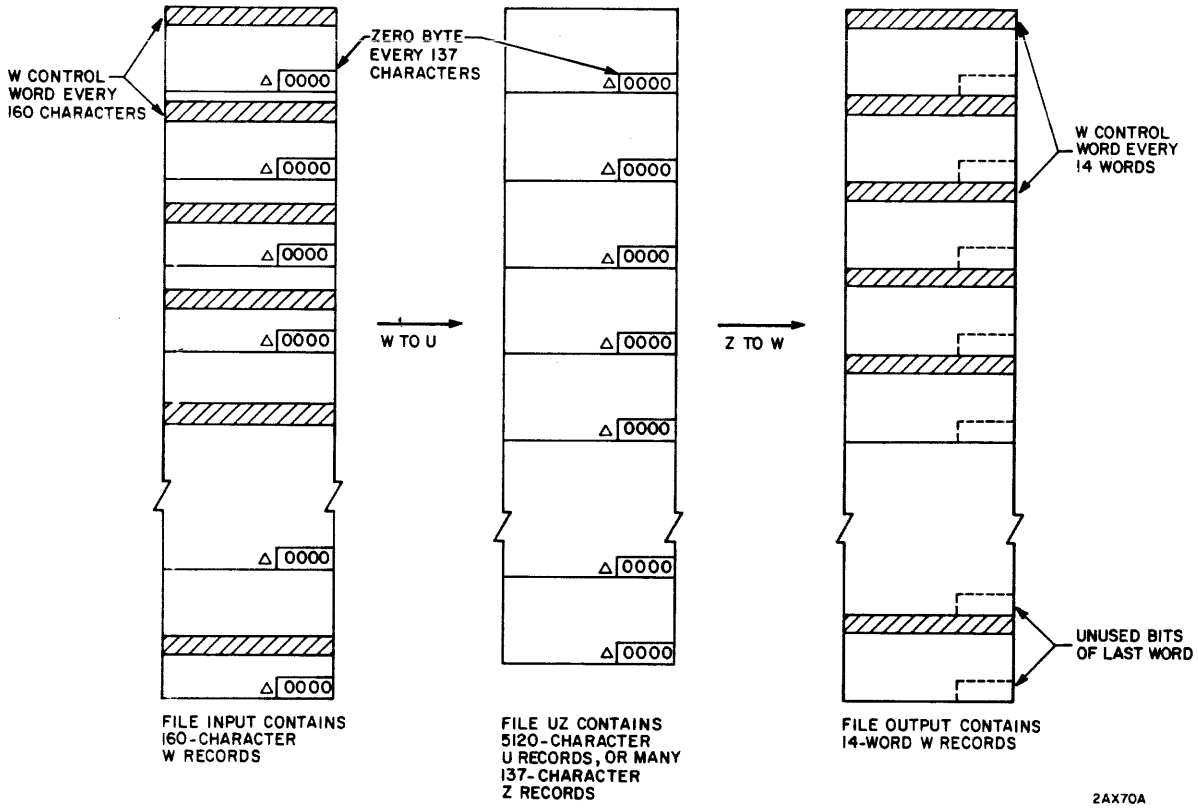




4. This example illustrates an S to W conversion using COPYS. Each S record is a library routine. To be able to be used at all by LIBEDT, each routine must be converted to a section in W format. COMPARE cannot be used following this copy to show equivalence.



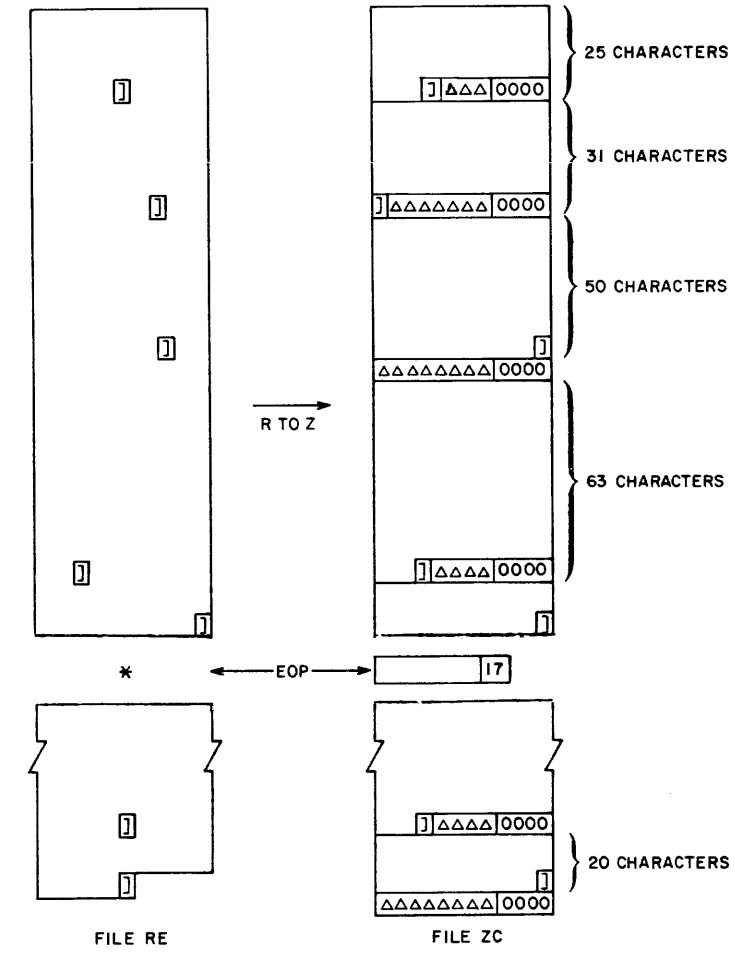
5. In this example, the second section on INPUT is a free-form binary deck that originally consisted of 137-character print lines in the form of Z records. On input, the deck has W control words inserted every 160 characters. The job shows how the W control words are removed through a W to U copy. Then, the U-type file is redefined as Z-type and copied to the OUTPUT file.

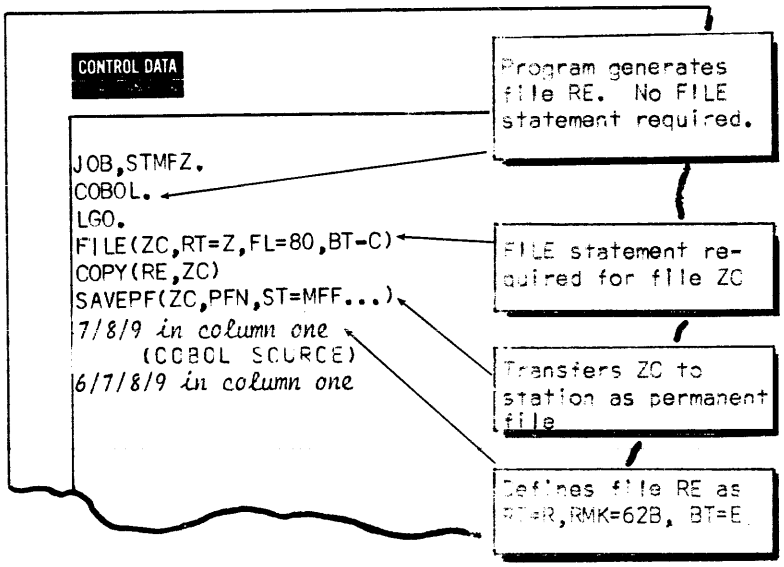


CONTROL DATA

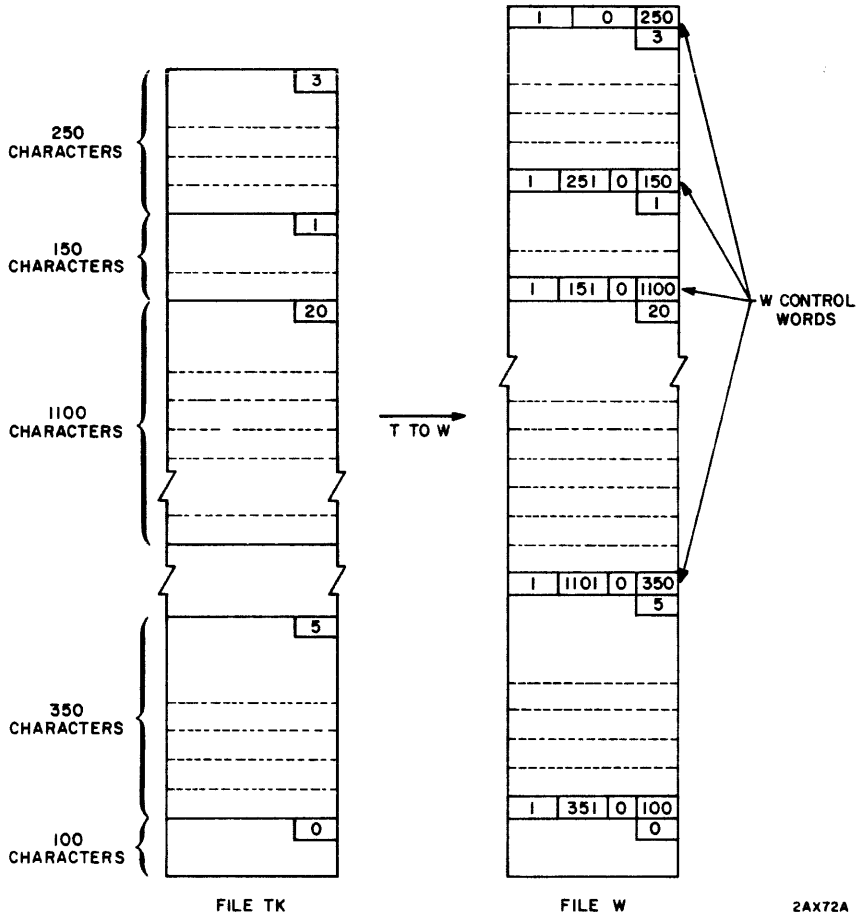
```
JOB,STMFZ.
FILE(UZ,RT=U)
COPY(INPUT,UZ)
FILE(UZ,RT=Z,FL=137)
REWIND(UZ)
COPY(UZ,OUTPUT)
7/8/9 in column one
      (FREE FORM BINARY DECK)
6/7/8/9 in column one
```

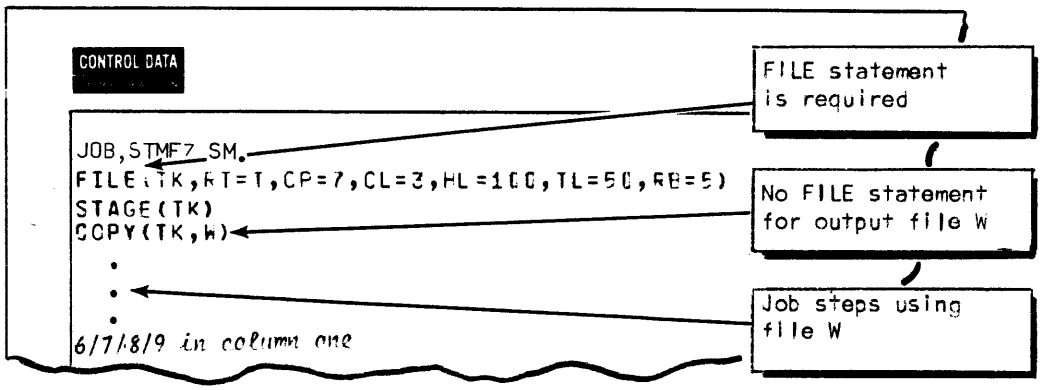
6. The user has a COBOL-generated file containing R records with E blocking. The record mark character is] . He wishes to convert the file to C-blocked Z records for use as a SCOPE standard permanent file at the CDC CYBER station (blocking not shown). Each tapemark on input becomes a level 17_8 48-bit appendage (EOP).





7. 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.





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
Program image module	---	W	unbl	No
Utilities				
COPY, COPYS, COPYP, etc.				
Input file	INPUT	W	unbl	Yes; refer to appendix E
Output file	OUTPUT	W	unbl	Yes; refer to appendix E
COPYSP				
Input file	INPUT	W	unbl	Yes
Output file	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
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, S, 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 must be Z; 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 loaded 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
Source output	SOURCE	W	unbl	Yes; can be blocked; to be reloaded under SCOPE 2, record type must be W or Z
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
Sort/Merge				
Source input	INPUT	W	unbl	Yes; can be RT=Z or F if FL≤100
List output	OUTPUT	W	unbl	Cannot be printed under SCOPE 2 if redefined
Binary owncode routines	LGO	W	unbl	Cannot be loaded under SCOPE 2 if redefined

†Word addressable file organization

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

ANALYZING ERRORS IN A SAMPLE FORTRAN PROGRAM G

This appendix describes the sample program that has been used repeatedly in examples in this manual. It shows how to analyze the output from the job to detect programming errors.

Figure G-1 illustrates the sample program. This FORTRAN Extended program is very simple, serving only to illustrate certain techniques. The program reads a section of data from the INPUT file. The final data card contains a number in column 21. The data consists of the base and height of a triangle which the program uses for computing the area of the triangle. The base, height, and area are written on OUTPUT; the area is written on a magnetic tape file.

The program performs these steps:

1. Reads the base, height, and contents of column 21 from a card in the data section.
2. Checks to see if column 21 contains a number.
   a. If column 21 does not contain a number, the program continues with step 3.
   b. If column 21 contains a number, the program stops. The presence of the number signals that the last card in the data section has been read.
3. Checks to see if the base or height is less than or equal to zero.
   a. If either value is zero or less, the triangle cannot exist. The program takes an exit to a subroutine that prints the message FOLLOWING INPUT DATA NEGATIVE OR ZERO and then returns to the main program.
   b. If neither value is zero or less, execution in the main program continues.
4. Calculates the area of the triangle using the base and height just read.
5. Prints the base, height, and area of the triangle.
6. Writes the area on a magnetic tape file.
7. Returns to step 1.

Figure G-2 presents a flow diagram of the program.

JOB PREPARATION

Suppose you are running the sample job for the first time and suspect that faulty logic, coding, or input might create errors. To help resolve any errors should they occur, you want to ensure that certain analytical aids are available with the output. This requires that you select the necessary options in the control statement section.

Since diagnostic messages are generated automatically by SCOPE and appear in the dayfile, you need take no special steps to obtain them.

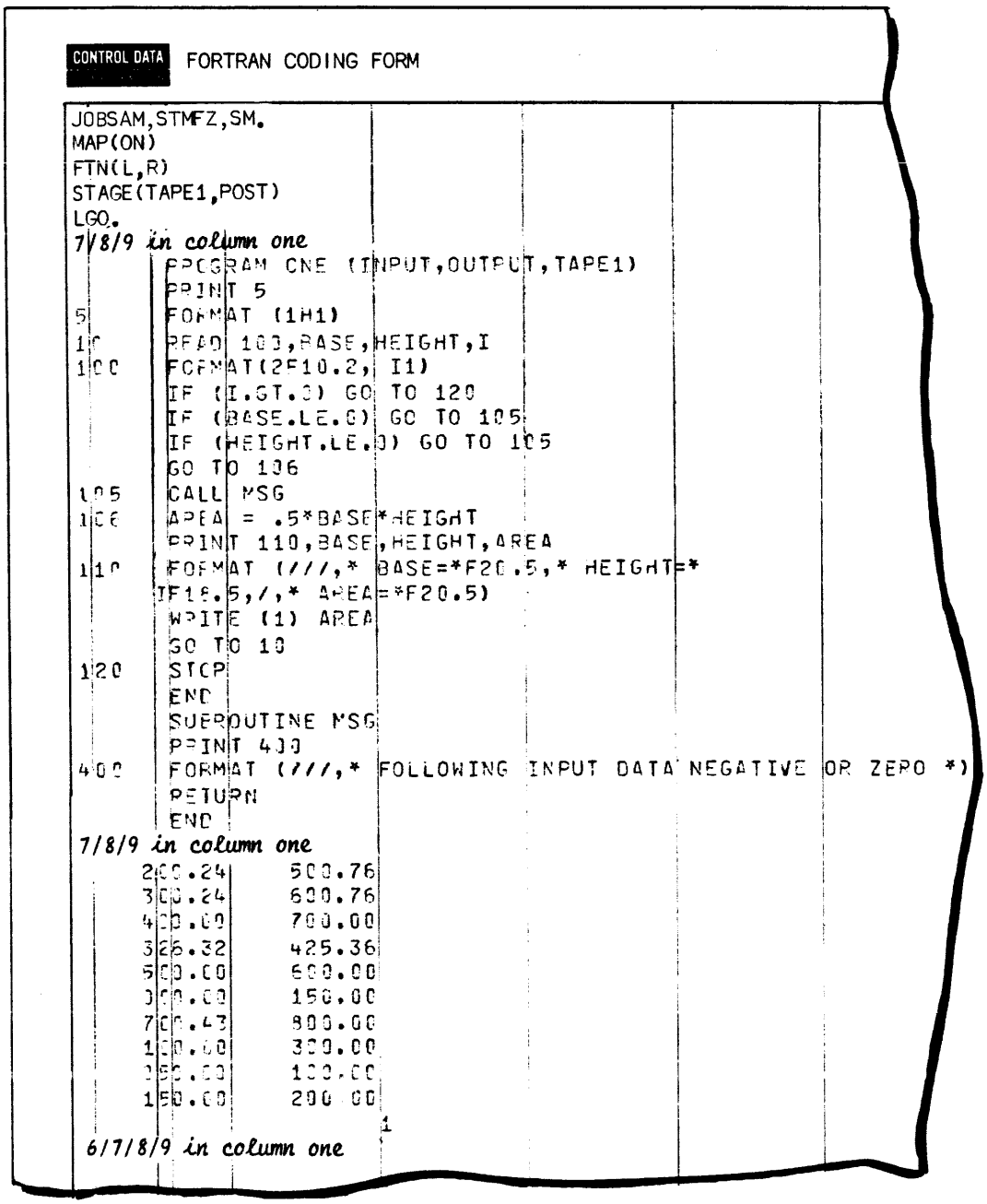


Figure G-1. Sample FORTRAN Extended Job

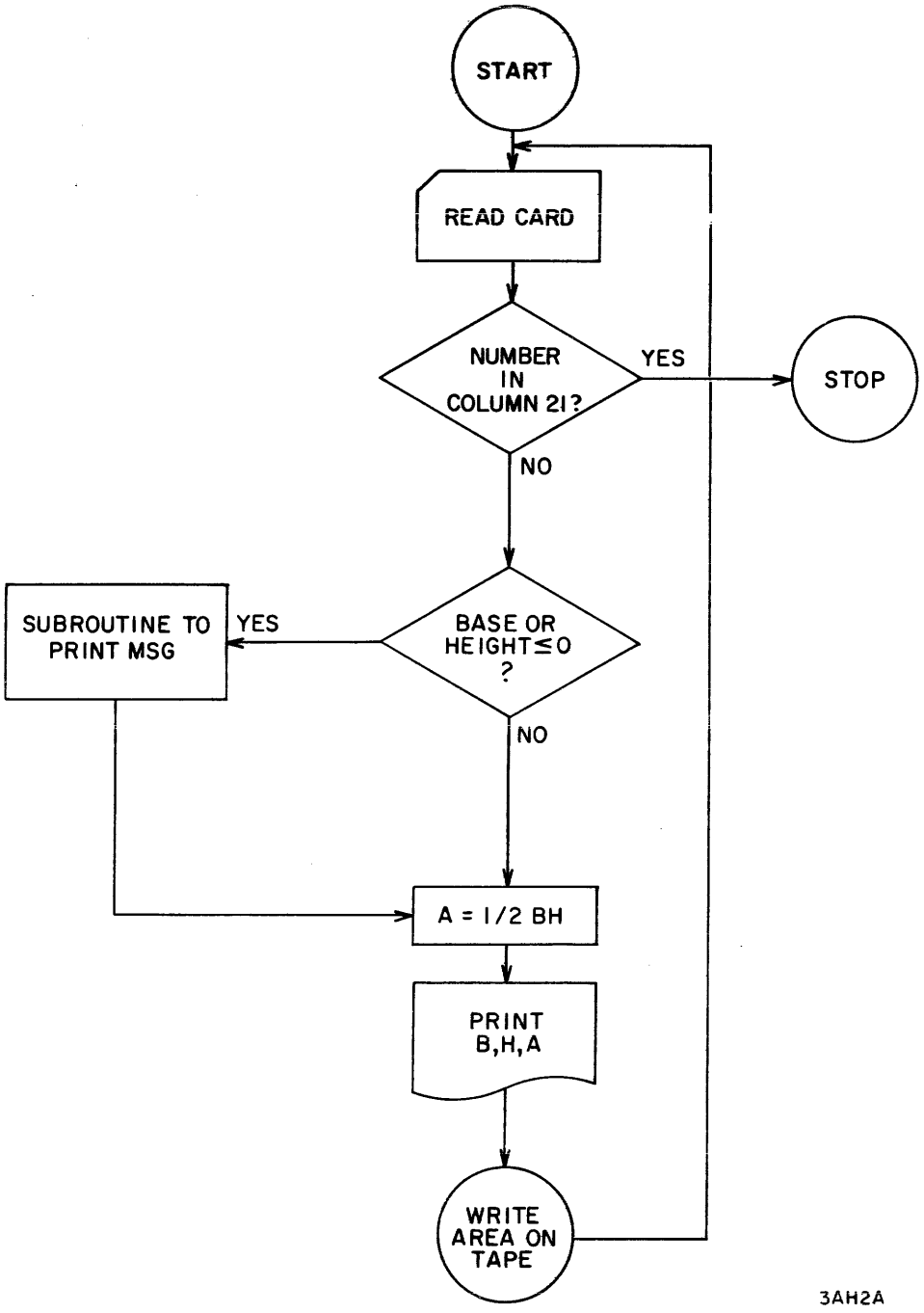


Figure G-2. Flow Diagram of Sample Job

A listing of a program in source language is generally printed automatically. You can make doubly sure that this output is present by placing the letter L in parentheses on the FTN statement. Look at the control statement section in the job in Figure G-1. Notice that parameters L and R have been selected on the FORTRAN statement. The R parameter requests an additional analytical aid, the symbolic reference map. Applications programmers seldom request this table unless they are relatively certain that their program contains errors. Although it is not generally needed, the symbolic reference map can make finding the values of variables easier. Later in this appendix you will find a description of how to interpret the symbolic reference map. The symbolic reference map is furnished by FORTRAN Extended and not by SCOPE. It is not available with all compilers. Check the reference manual for the compiler you are using to determine if it provides this table.

You may also want a program memory map. Map generation may be automatic at your installation. If it is not, you can ensure that a map is produced by including a MAP(ON) statement in the control statement section before calling for loading of the compiled program.

In most cases, if an error occurs that causes the job to abnormally terminate, the operating system produces a standard dump. Usually, a programmer refers to the dump only if the cause of the error cannot be determined from other diagnostics. This dump is described later in this appendix.

ANALYSIS

After running his job, the programmer first determines whether the output contains any obvious errors. If it does, he then proceeds through a set of logical steps to identify, analyze, and correct the errors. As each of these steps is described, the major feature of the aid is noted, whether or not the aid pertains to the detected error.

EXAMINATION OF OUTPUT

When a programmer gets a job back, he glances through his listing to see if he obtained the type of output he expected and to learn if any errors occurred. Figure G-3 identifies the items in the listing and shows the sequence in which they occur for the sample job.

The first item is the banner pages identifying the output for the job. This is described in Section 1 of this manual.

Next is the listing of the source statements in the FORTRAN Extended main program, ONE, followed by its list of diagnostics, if any. There are none for this example. Next is the symbolic reference map for the main program. Following the map is a listing of the source statements in subroutine MSG, its list of diagnostics (none in this example), and its symbolic reference map.

The next item is the load map of the program, identified as such by the title SCOPE 2 LOAD MAP on the upper left-hand corner of the page.

Following the map is the output from the program. A brief glance at this output (Figure G-4) reveals that only five of the anticipated ten sets of answers appear. Thus, an error occurred in the program.

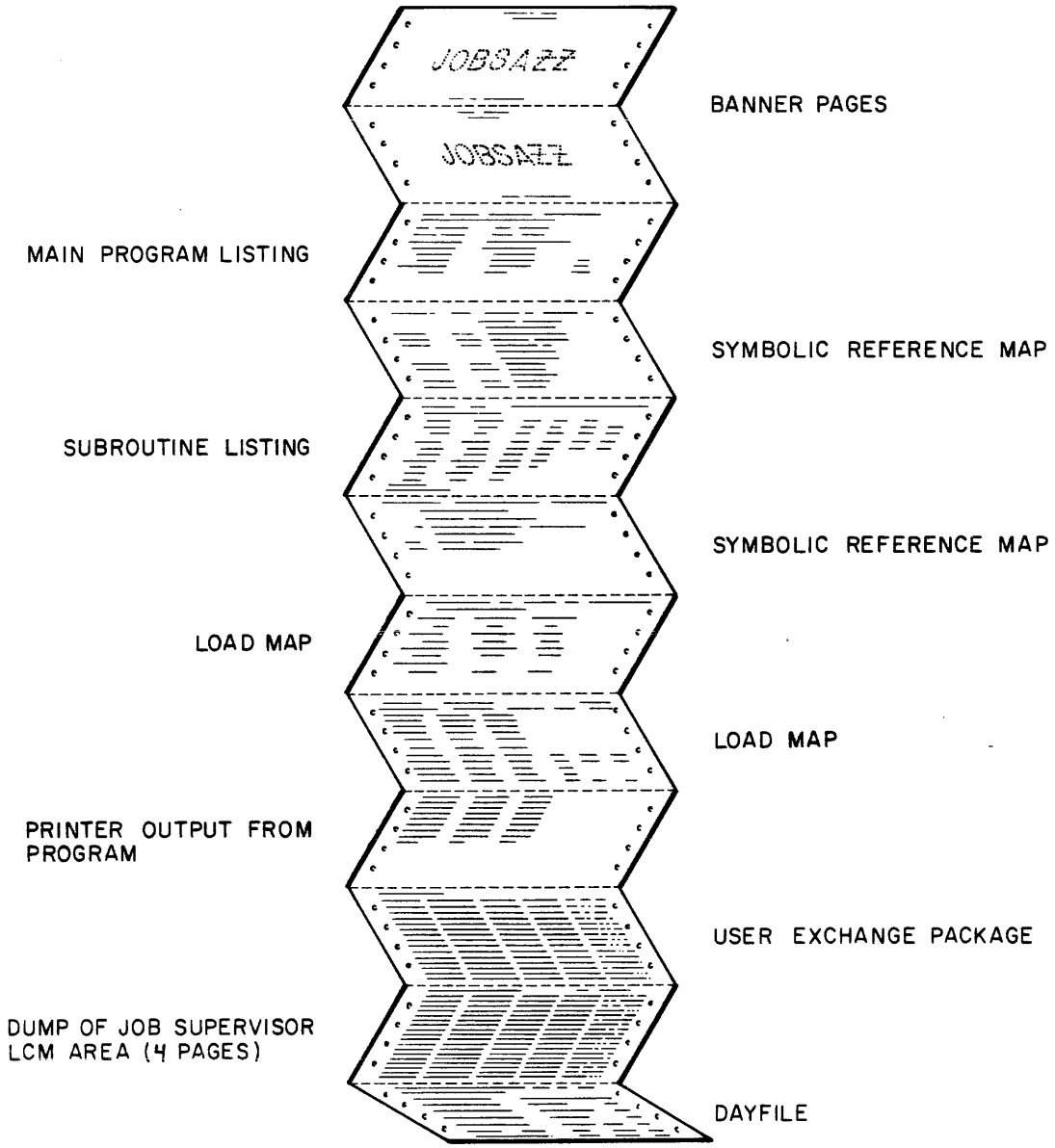


Figure G-3. Output from Sample Job

```
BASE=          200.24000 HEIGHT=          500.76000
AREA=        50136.09120

BASE=          300.24000 HEIGHT=          600.76000
AREA=        90196.09120

BASE=          400.00000 HEIGHT=          700.00000
AREA=       140000.00000

BASE=          326.32000 HEIGHT=          425.36000
AREA=        69401.73760

BASE=          500.00000 HEIGHT=          600.00000
AREA=       150000.00000
```

Figure G-4. Printer Output from the Program

Because this error resulted in job termination, a standard dump was generated automatically. This dump, which consists of a listing of the user exchange package and a dump of the job supervisor paged LCM area, is the next item on the listing. The final item in the listing is the dayfile for the job. (The dayfile for the job may be printed at the beginning of the listing, depending on an installation option.) It is, however, the first item that will be considered in detail.

EXAMINATION OF THE DAYFILE

After glancing through the listing, the next step in detecting and analyzing errors is examination of the dayfile in detail. Errors that occur during loading or execution are noted here without elaboration. More specific descriptions of such errors may be noted in other sections of the listing.

The dayfile for the sample job is reproduced in Figure G-5. Every fifth line is numbered for reference. The numbers do not appear in the dayfile.

The date the program was run and other data related to the versions of SCOPE and the station operating system appear above line 1.

Each line begins with the time that the item on the line occurred or the time that the request on that line was processed in terms of hours, minutes, and seconds. The leftmost column gives the wall clock time; the second column gives the elapsed CPU time, except when the event is a station event occurring independent of the CPU. In this case, the wall clock time ordinarily given in the left column is shifted right to the second column.

Lines 2, 3, 5, and 6 show the control statements that were processed before job termination. Each control statement is listed preceded by a dash in the sequence encountered in the control statement section.

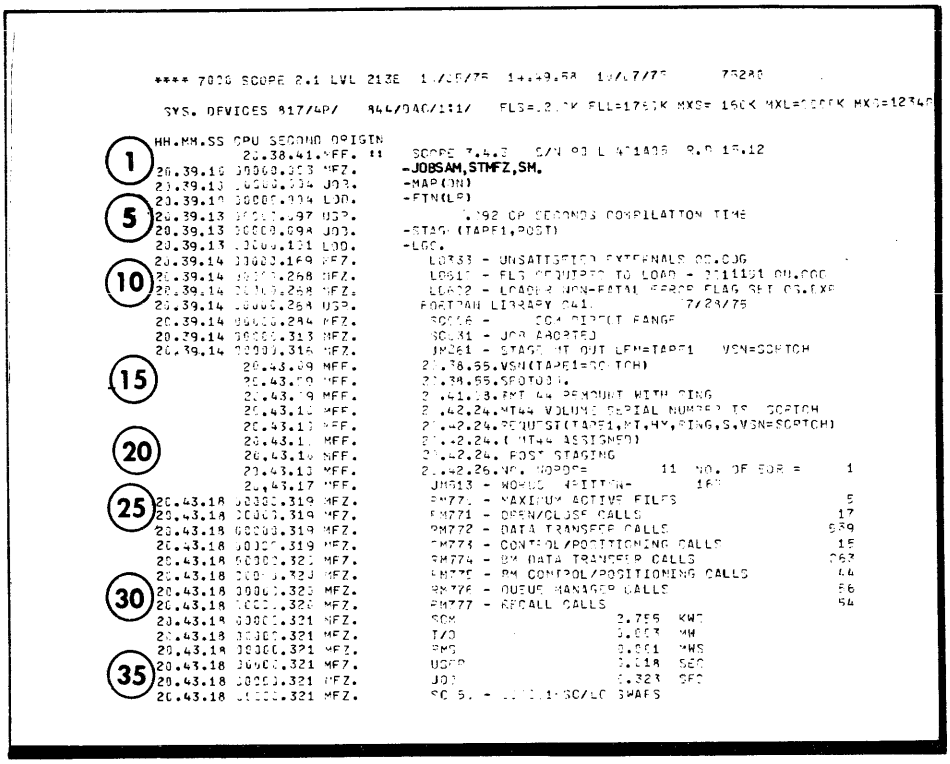


Figure G-5. Job Dayfile

Line 2 contains the MAP statement requesting generation of the load map, as mentioned previously. Note that although the STAGE statement on line 5 identifies file TAPE1 as a post-staged tape, it is not until line 13, following job termination, that the operator is requested to mount the tape. Lines 14 through 22 provide information related to the post-staging of file TAPE1.

Line 3 calls for the loading and execution of the FORTRAN Extended compiler. The next line informs us that compilation of the sample program required 0.092 seconds. Execution of the object program began when the LGO statement (line 6) was processed. Lines 7, 8, and 9 provide loader information. Line 7 is of particular interest because it mentions the presence of an unsatisfied external. Although not a fatal loader error, it may be significant in isolating the cause of the error. Line 8 informs us that the loader required 0011151 words in SCM to perform the load. This figure does not represent the size of the user program. Line 9 notifies us that the message noting the unsatisfied external (LD333 on line 7) is nonfatal and does not inhibit program execution. Line 10 identifies the library and version of the library used for loading. In this case, cycle 410 of the library named FORTRAN was used.

Then, we are informed on lines 11 and 12 that the job was aborted as a result of an SCM direct range error. This means that the error occurred because the program referenced an address that was out of the range of the job's field length in small central memory.

Following job termination, staging of output occurred. The last items on the listing provide the accounting information as described in section 1.

Note that although the job was in the system for approximately 4 minutes wall clock time, it used the CPU for only 0.323 seconds. The time a job is in the system depends on resource availability, job priority, and the number and priorities of other jobs in the system.

EXAMINATION OF SOURCE PROGRAM LISTINGS

After examining the dayfile, return to the listing of the FORTRAN Extended program (Figure G-6) and subroutine (Figure G-7). The first line of each listing provides the following items.

Name of program or subroutine

Name and version number of the compiler used

Date on which program was compiled

Time at which program was compiled

Page number

```
PROGRAM ONE        76/76   OPT=1                    FTN 4.5+410        10/08/75  18.45.13$      PAGE     1

                PROGRAM ONE (INPUT,OUTPUT,TAPE1)
                PRINT 5
          5     FORMAT (1H1)
          10    READ 100,BASE,HEIGHT,I
          100   FORMAT(2F10.2, I1)
                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=*
               1F13.5,/,* AREA=*F20.5)
                WRITE (1) AREA
                GO TO 10
          120   STOP
                END
```

Figure G-6. Program Listing

```
SUBROUTINE MSA        76/76   OPT=1                    FTN 4.5+410        10/08/75  18.45.13$      PAGE     1

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

Figure G-7. Subroutine Listing

In both Figures G-6 and G-7, numbers appearing in the leftmost column are added by the compiler to every fifth line, making reference to a specific line in a listing easier. Notice that for this program, no diagnostics appear here. Nevertheless, careful examination of the listing often reveals certain kinds of errors. One of these, detectable by comparing the listing to the original coding sheet, is the error resulting from a mispunched card. Often, it is easier to determine errors in coding or logic from a listing simply because it is easier to read a listing than a coding sheet or card deck. You may be able to learn all you need to know to correct the error in this program by looking very carefully at the listing.

EXAMINATION OF THE LOAD MAP

Now, examine the load map, first to see what it contains and second to learn more about the error in the program. In this map, reproduced in Figure G-8, each item is numbered for reference.

Keep in mind two facts about this map.

1. All address and address contents are expressed as octal values.
2. All addresses are relative to the first word of the user field in small central memory. They are not absolute physical addresses.

First, examine the top line (item 1) of the load map. This line identifies the printout as being a LOAD MAP, identifies the version of the loader in use, and gives a page number.

If overlays or segments were generated, which is not the case for this program, overlay information would appear between items 1 and 2.

Item 2 gives the symbolic name for the point of entry to the program. In FORTRAN programs, this is the same as the program name; in this case, ONE. Item 3 tells us that the program is entered at SCM relative address 166.

Items 4 and 5 reveal that the program occupies 5351_8 words of SCM and no LCM. Looking at item 6, it becomes apparent why this aid is known as a map. This item tells where the first word of each program and subroutine is loaded and their lengths.

Figure G-9 shows the layout of the program in SCM. Notice that the first word, RAS+000000, is at the bottom of memory representation; the last word is at the top. RAS stands for Reference Address Small. It represents the starting location in memory of the user SCM field. All locations in the map are relative to this address. When using automatic memory management, the length of the user field in SCM is the 100-word increment next higher than the last word address of the program. Thus, the SCM field length for this job step is 5400_8.

As described in Section 4, the first 100 words of the user field in SCM serve for communication between SCOPE and your job. Program ONE begins at RAS+000100; subroutine MSA begins at address RAS+000256.

Returning again to item 6 of the Load Map, examine the data listed under the headings BLOCK, ADDRESS, and LENGTH. These columns contain the names beginning addresses, and lengths of the program and subroutines needed to process the job. The names starting with Q8NTRY= are object-time FORTRAN routines obtained by the loader from the system library named FORTRAN. Names which are indented and enclosed by slashes are common blocks.

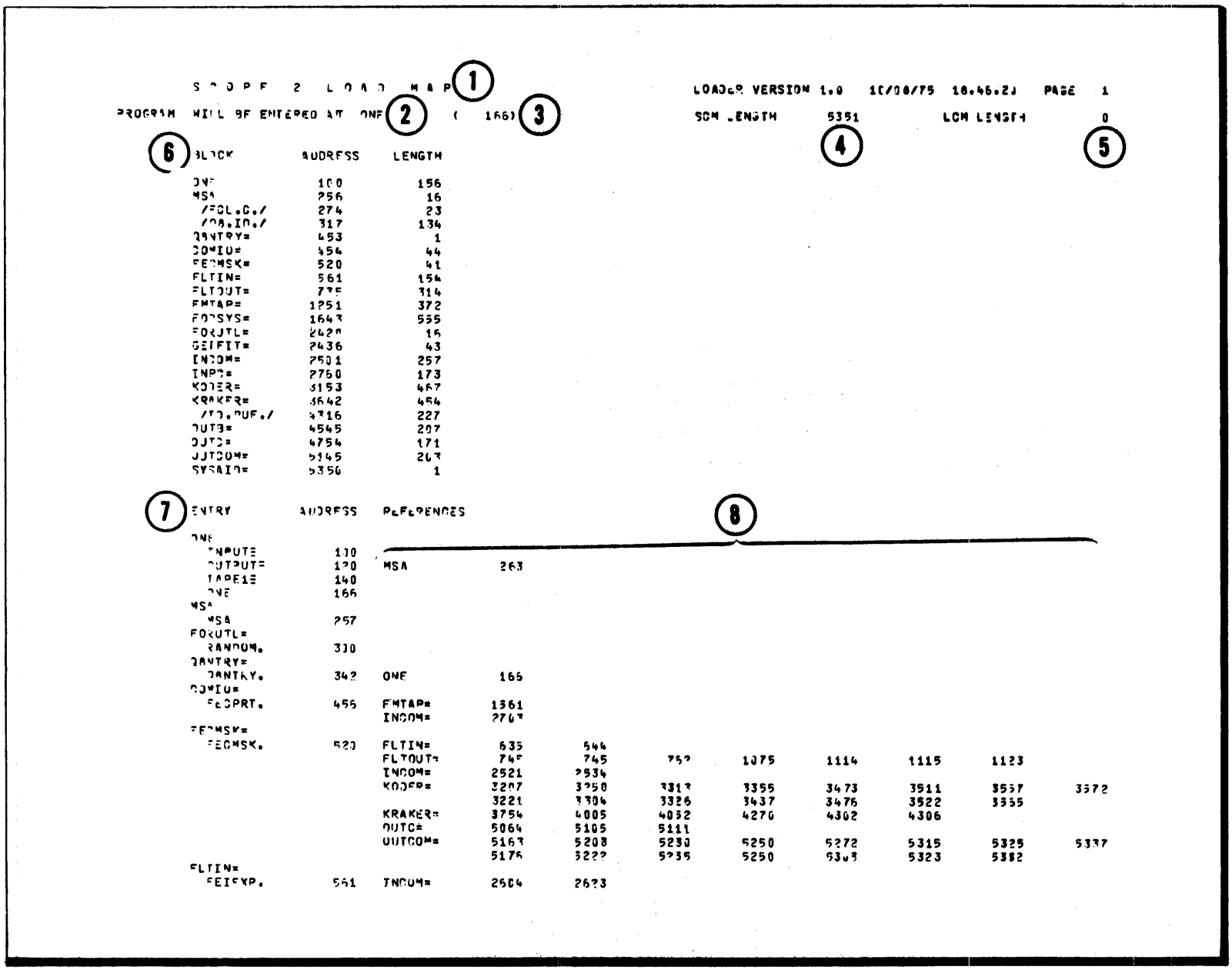
Data given in item 7 under the column headings ENTRY, ADDRESS, and REFERENCES give information related to the entry points in each of the subroutines. Each program or subroutine is left-justified under the heading ENTRY. Indented under each name is a list of symbolic addresses at which the routine can be entered and the octal address associated with each entry point symbol. Many entry point names and addresses are listed; in fact, the full list for this program occupies over two pages. The adjacent columns are designated as item 8. Each word in the first column under REFERENCES is the name of a program or subroutine that contains references to the entry point. Each entry in the next columns is the address of an instruction that refers to that entry point. For example, the entry point OUTPUT= in program ONE is referred to by an instruction at location 000263 in subroutine MSA.

The entry point Q8NTRY=Q8NTRY. is referred to by an instruction in the subroutine MSA. This instruction is at location 166. Once you can visualize the name and location of an entry point in a subroutine and can visualize the location of an instruction within another program or subroutine that refers to that entry point, you have a clear idea of how two subroutines are logically linked.

Three more items on the map remain to be considered. Here, we find another clue to the cause of the error in the program. Item 9 on the second page of Figure G-8 has the heading UNSATISFIED EXTERNALS. It contains one entry, MSG. The unsatisfied reference occurred in program ONE at address 200 (items 10 and 11). When the loader was attempting to satisfy externals, it searched fruitlessly for a subroutine or program that contained the entry point MSG. Upon load completion, the reference to MSG was still unsatisfied. The loader issued the dayfile message previously mentioned. The loader then added 400000_8 to the address to force an error condition if the reference is ever encountered during program execution. Thus in the loaded program, the reference to the unsatisfied external contains the address 400200. Now, checking the listing of program ONE in Figure G-7, we can easily find this instruction. It occurs as line 10 of the listing. The FORTRAN statement CALL MSG is obviously a call to the FORTRAN subroutine, but checking this routine in Figure G-7 reveals an interesting fact; although this subroutine was called by the name MSG in program ONE, it is named MSA at the beginning of the subroutine. Obviously, the name of the subroutine has been mispunched. Checking the original coding sheet in Figure G-1 shows that the name should have been punched as MSG in both places.

Now, you can understand why the program terminated prematurely. When the reference to 400200 was encountered in program ONE, the reference was out of range of the SCM field size (5400_8) causing an error condition. SCOPE issued a message stating that an SCM direct range error had occurred (line 11 of Figure G-5). The operating system then printed the standard dump and terminated the job.

As additional evidence of this error, you can see in Figure G-8 that the entry point named MSA is not referred to by any program or subroutine. Thus, there is no way it could have been executed.



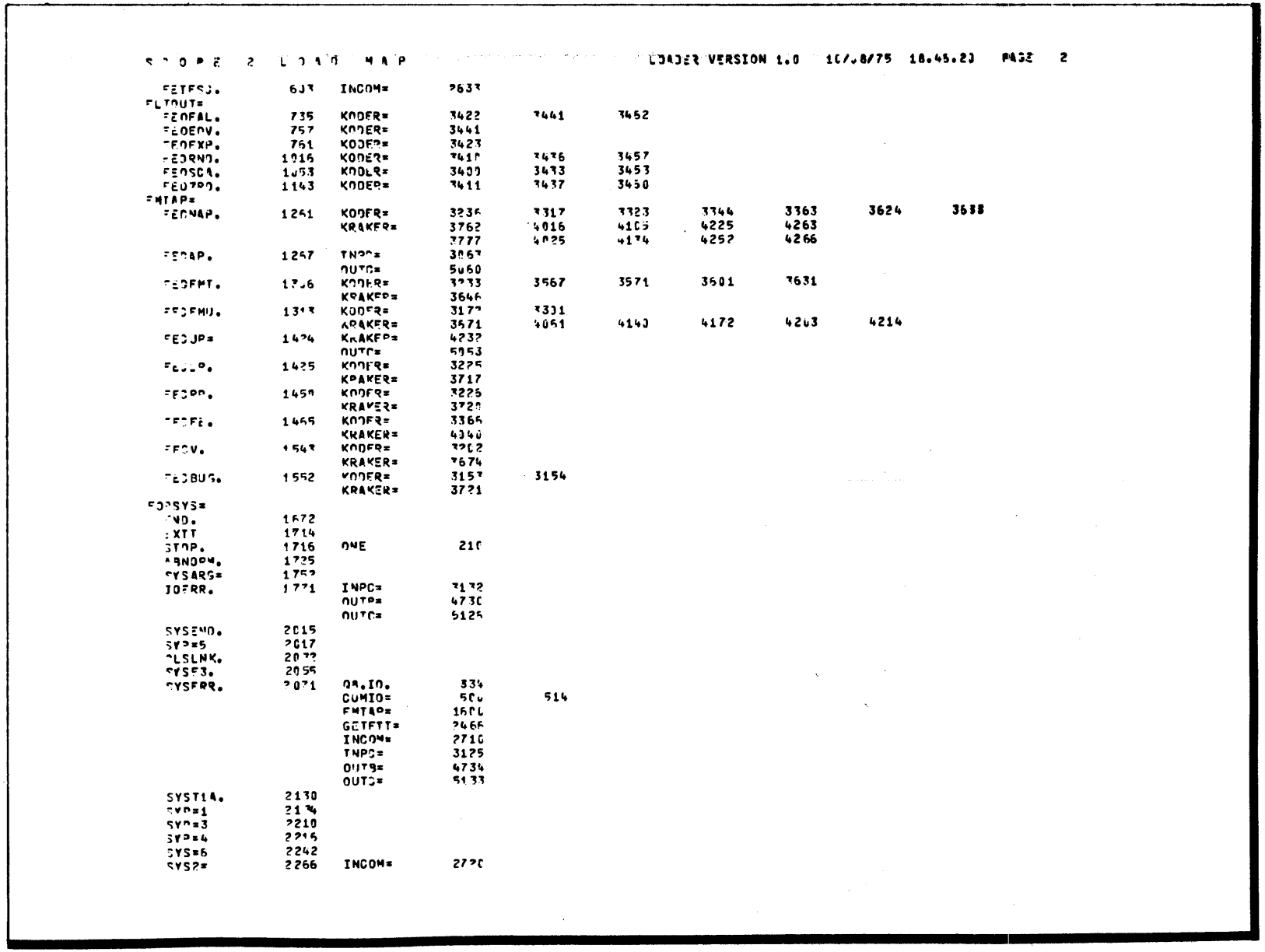
```
S C O P E   2   L O A D   M A P (1)                                    LOADER VERSION 1.0   10/08/75  18.46.20   PAGE   1
PROGRAM  WILL BE ENTERED AT  ONE (2)   (   166) (3)                    SCM LENGTH      5351          LCM LENGTH        0
                                                                                   (4)                                (5)
(6) BLOCK          ADDRESS    LENGTH

    ONE              100        156
    MSA              256         16
     /FCL.C./        274         23
     /DB.ID./        317        134
    QANTRY=          453          1
    COMID=           454         44
    FECMSK=          520         41
    FLTIN=           561        154
    FLTOUT=          735        314
    FMTAP=          1251        372
    FORSYS=         1643        555
    FORUTL=         2420         16
    GEIFIT=         2436         43
    INCOM=          2501        257
    INPC=           2760        173
    KODER=          3153        467
    KRAKER=         3642        454
     /TD.PUF./      4316        227
    OUTB=           4545        207
    OUTC=           4754        171
    OUTCOM=         5145        203
    SYSAID=         5350          1

(7) ENTRY          ADDRESS    REFERENCES                                  (8)

    ONE
      INPUTE          100
      OUTPUTE         120    MSA          263
      TAPE1E          140
      ONE             166
    MSA
      MSA             257
    FORUTL=
      RANDOM.         300
    QANTRY=
      QANTRY.         342    ONE          166
    COMID=
      FECPRT.         456    FMTAP=      1561
                             INCOM=      2703
    FECMSK=
      FECMSK.         520    FLTIN=       635      644
                             FLTOUT=      745      745      752     1075     1114     1115     1123
                             INCOM=      2521     2534
                             KODER=      3207     3250     3313     3355     3473     3511     3557     3572
                                         3221     3304     3326     3437     3476     3522     3555
                             KRAKER=     3754     4005     4032     4270     4302     4306
                             OUTC=       5064     5105     5111
                             OUTCOM=     5163     5208     5230     5250     5272     5315     5325     5337
                                         5176     5222     5235     5250     5303     5323     5382
    FLTIN=
      FEIEXP.         561    INCOM=      2604     2623
```

Figure G-8. Load Map



```
S C O P E   2   L O A D   M A P                                    LOADER VERSION 1.0   10/08/75  18.45.20   PAGE   2

  FETFSC.        603   INCOM=        2633
FLTOUT=
  FEOFAL.        735   KODER=        3422      3441      3452
  FEOEOV.        757   KODER=        3441
  FEOFXP.        761   KODER=        3423
  FEORND.       1016   KODER=        3410      3436      3457
  FEOSCA.       1053   KODER=        3400      3433      3453
  FEOTRO.       1143   KODER=        3411      3437      3450
FMTAP=
  FECNAP.       1261   KODER=        3236      3317      3323      3344      3363      3624      3638
                       KRAKER=       3762      4016      4105      4225      4263
                                     3777      4025      4134      4252      4266
  FECAP.        1267   INPC=         3063
                       OUTC=         5060
  FECFMT.       1306   KODER=        3233      3567      3571      3601      3631
                       KRAKER=       3646
  FECFMU.       1313   KODER=        3172      3301
                       KRAKER=       3571      4061      4140      4172      4203      4214
  FECUP=        1424   KRAKER=       4232
                       OUTC=         5053
  FECLP.        1425   KODER=        3225
                       KRAKER=       3717
  FECRP.        1450   KODER=        3225
                       KRAKER=       3720
  FECFE.        1465   KODER=        3365
                       KRAKER=       4040
  FECV.         1543   KODER=        3202
                       KRAKER=       3674
  FECBUG.       1552   KODER=        3153      3154
                       KRAKER=       3721
FORSYS=
  END.          1672
  EXIT          1714
  STOP.         1716   ONE            210
  ABNORM.       1725
  SYSARG=       1752
  IOERR.        1771   INPC=         3132
                       OUTP=         4730
                       OUTC=         5125
  SYSEND.       2015
  SYP=5         2017
  CLSLNK.       2032
  SYSE3.        2055
  SYSERR.       2071   QA.IO.         334
                       COMIO=         500       514
                       FMTAP=        1600
                       GETFTT=       2466
                       INCOM=        2710
                       INPC=         3125
                       OUTB=         4734
                       OUTC=         5133
  SYST1A.       2130
  SYP=1         2134
  SYP=3         2210
  SYP=4         2215
  SYS=6         2242
  SYS2=         2266   INCOM=        2720
```

Figure G-8. Load Map (Cont'd)

```
S C O P E   2   L O A D   M A P                                    LOADER VERSION 1.0   10/08/75  18.45.20   PAGE   3

  FECOPE.        2334   INPC=        3044
                        OUTC=        5033
FORUTL=
  CBD.           2423   FORSYS=      1665      2002      2075      2173
                        INCOM=       2575
  DEN.           2430   FORSYS=      1773      2146
GETFIT=
  GETFIT.        2441   INPC=        3023
                        OUTB=        4567
                        OUTC=        4777
  NAME.          2477
INCOM=
  FEIGNO.        2503   FLTIN=        553       567       572       573       574
                        KRAKER=      3705      3707      3713      4067      4112      4216
  FEISBL.        2520   KRAKER=      3660      3663      3702      3726      4106
                                     3661      3664      3703      3727      4107
  FEINUM.        2541   KRAKER=      3731      4064
  FEIFSG.        2552
  FEIBLK=        2615   KRAKER=      4235
  FEIBLK.        2625   KRAKER=      4232
  FEIFST.        2641   FLTIN=        652       654       661       662       665       676
                        KRAKER=      4240
  ERRSET         2644
  FEIERR.        2660   FLTIN=        570       671
                        KRAKER=      4070      4137
INPC=
  INPCI.         3002   ONE           172
  INPCR.         3066
KODER=
  KOUPT.         3154   OUTC=        4767
  KODHRT=        3625   OUTC=        5046
  KOPFP.         3632   OUTC=        4770
KRAKER=
  KRTNIT.        4226   INPC=        3060
OUTB=
  OUTBI.         4561   ONE           206
  OUTBR.         4721
OUTC=
  OUTCI.         4771   ONE           170       204
                        MSA           261
  OUTCR.         5067
OUTCOM=
  FEOL.          5145   KODER=       3171      3234
  FEOI.          5152   KODER=       3165      3167
  FEOXFL.        5222   FLTOUT=       760      1154
                        KODER=       3464      3534
  FEOAFM.        5230   FLTOUT=      1004      1007      1011      1015      1152
  FEOBLS.        5235   FLTOUT=       740       742       744      1140      1141      1145      1167
                        KODER=       3415      3544
  FEOCNV.        5250   FLTOUT=       756
  FEOCHR.        5314   KODER=       3231      3322
  FEOR1F.        5325   FLTOUT=      1136
  FEOR1D.        5331
  FEONTL.        5336
SYSATD=
  SYSAID=        5350   08.IO.        341

UNSATISFIED EXTERNALS
```

Figure G-8. Load Map (Cont'd)

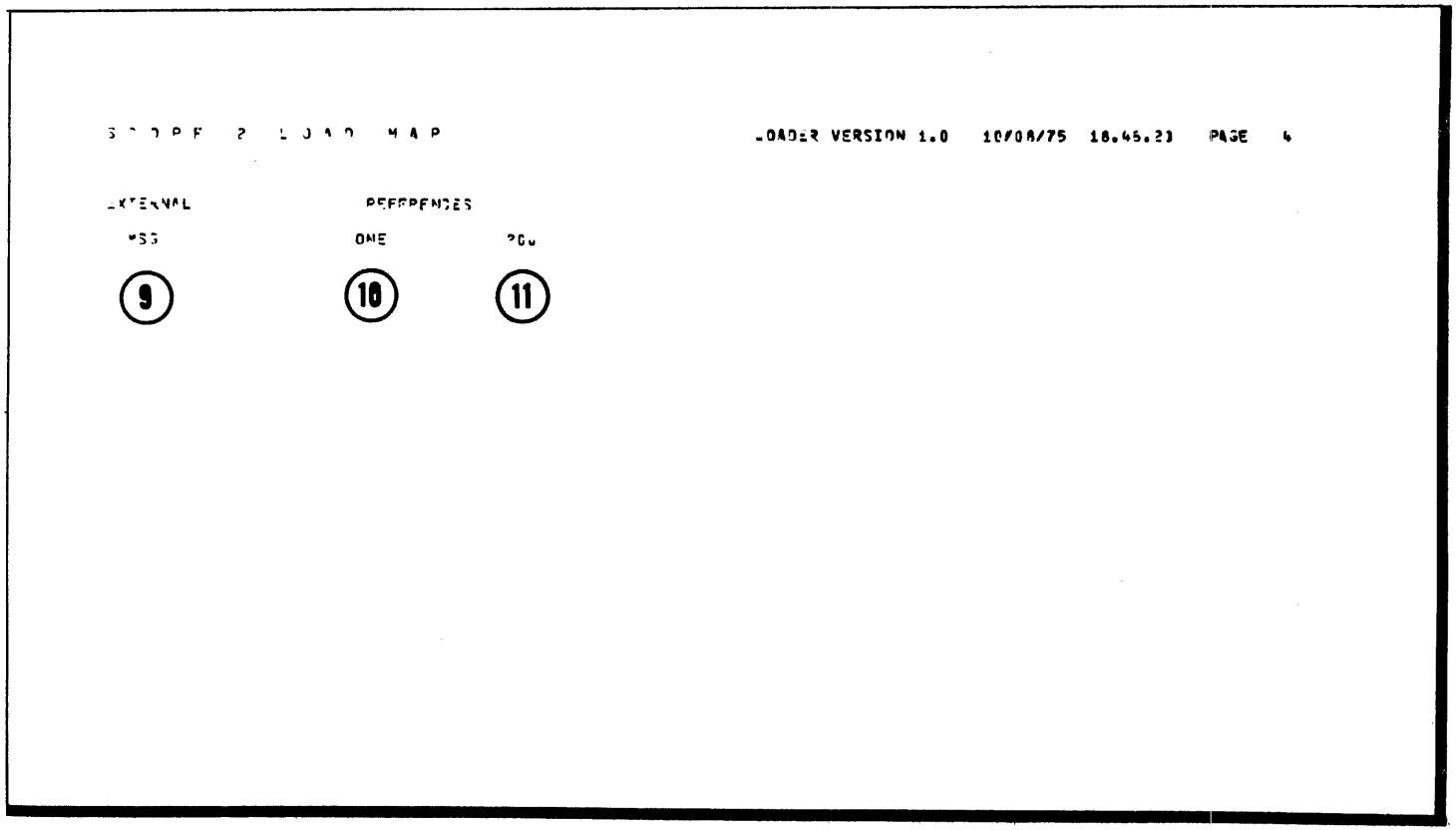


Figure G-8. Load Map (Cont'd)

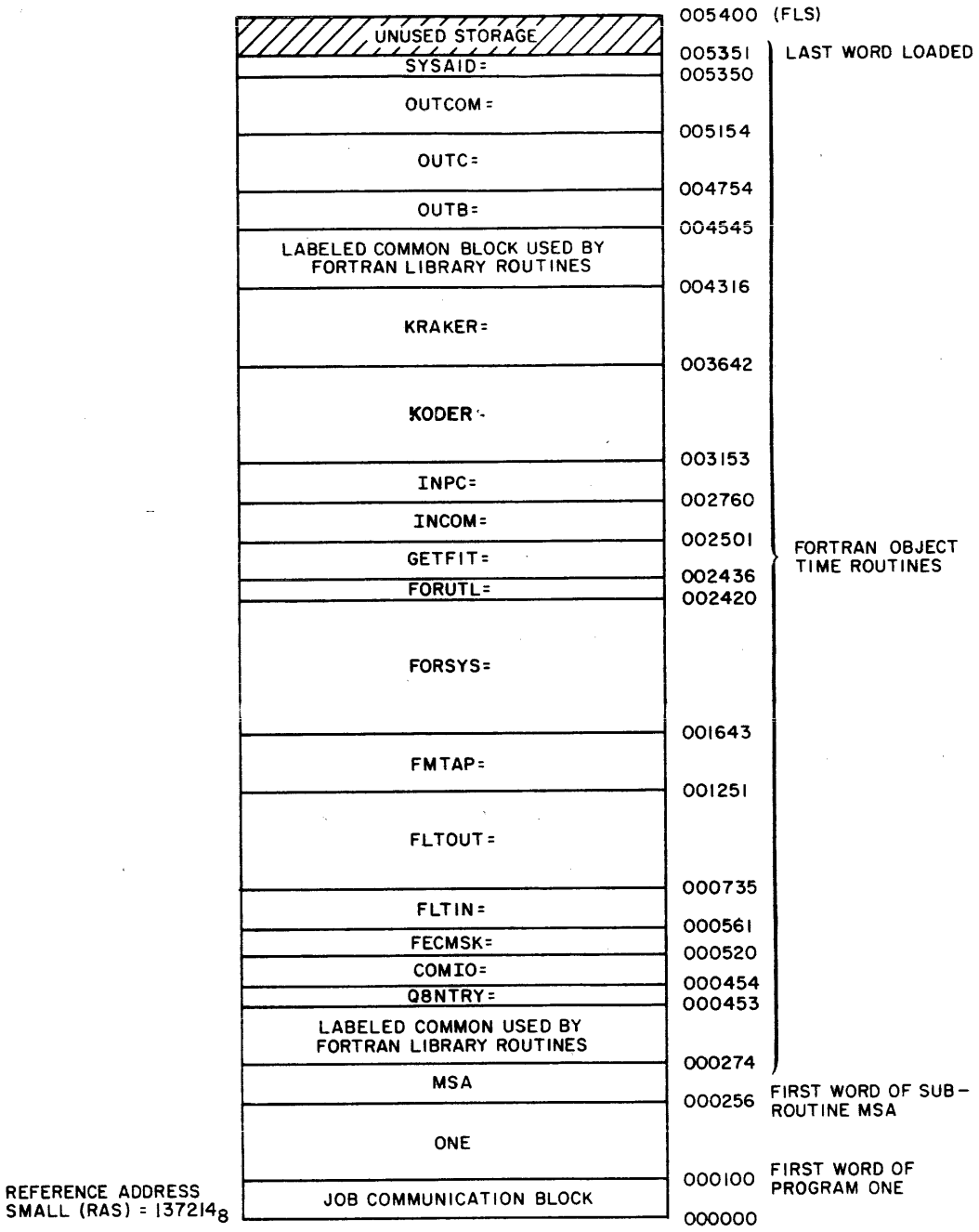


Figure G-9. Small Central Memory Layout

EXAMINATION OF THE STANDARD DUMP

Next, we will look at the standard dump, first to determine what information it provides and then to see how the error is reflected in the dump. This dump consists of the User Exchange Package and the Dump of Job Supervisor Paged LCM Area.

Many programmers maintain that the only way to fully understand the contents of a dump is through experience and practice. In time, one develops an understanding of how a dump should be used. Our purpose here is to give you a basis from which to practice using dumps. We shall introduce the main items in a dump and encourage you to use this information to practice reading dumps.

USER EXCHANGE PACKAGE

The User Exchange Package (Figure G-10) includes the contents of the User Exchange Package, the contents of the first 100 words in SCM (the job communication area), and when applicable, the contents of the 100 words preceding and the 100 words following the address at which the job terminated. In this example, the memory dump of the area that includes the termination address is not provided, since the instruction referenced was out of range of the SCM field.

On a standard dump, the item used most frequently in tracing errors is the User Exchange Package.

Referring to item 1 in Figure G-10, P, the program address counter, contains the relative address of the instruction in the program that would have been executed next if the job had not terminated. Notice that P contains 400201, one more than the unsatisfied reference. Here, then, is another clue to the error. The leading 4 in the address again indicates the presence of the unsatisfied external. Usually, P indicates an address one more than the last instruction executed. In some instances, however, P can contain an address many instructions beyond the instruction at which the error occurred. In particular, when a floating point divide generates an illegal operand, the contents of P can be several instruction words beyond the erroneous divide. Execution may even have progressed to another subprogram or subroutine.

When P indicates a nonsensical value, check to see that you have not violated the integrity of the program by loading data into the program area. This is a possibility when I/O statements use indexed variables for loading data into arrays.

Item 2, RAS, gives the reference address, that is, the absolute address of the first word of the field in SCM. In this case, the field begins at address 137214. This address is determined solely by the operating system and changes each time the program is loaded in SCM.

Item 3, FLS, denotes the SCM field length for this step in the job. As previously noted under automatic field assignment (Section 4), the size of this field is set by the loader to the next increment of 100. Thus, this job step requires 5400_8 words.

Item 4, the PSD register, gives the cause of the job termination.

The program status designator (PSD) register is a collection of 18 programs status flags. Six of these flags are mode designators and 12 are condition designators. The arrangements of these flags in the register is illustrated in Figure G-11.

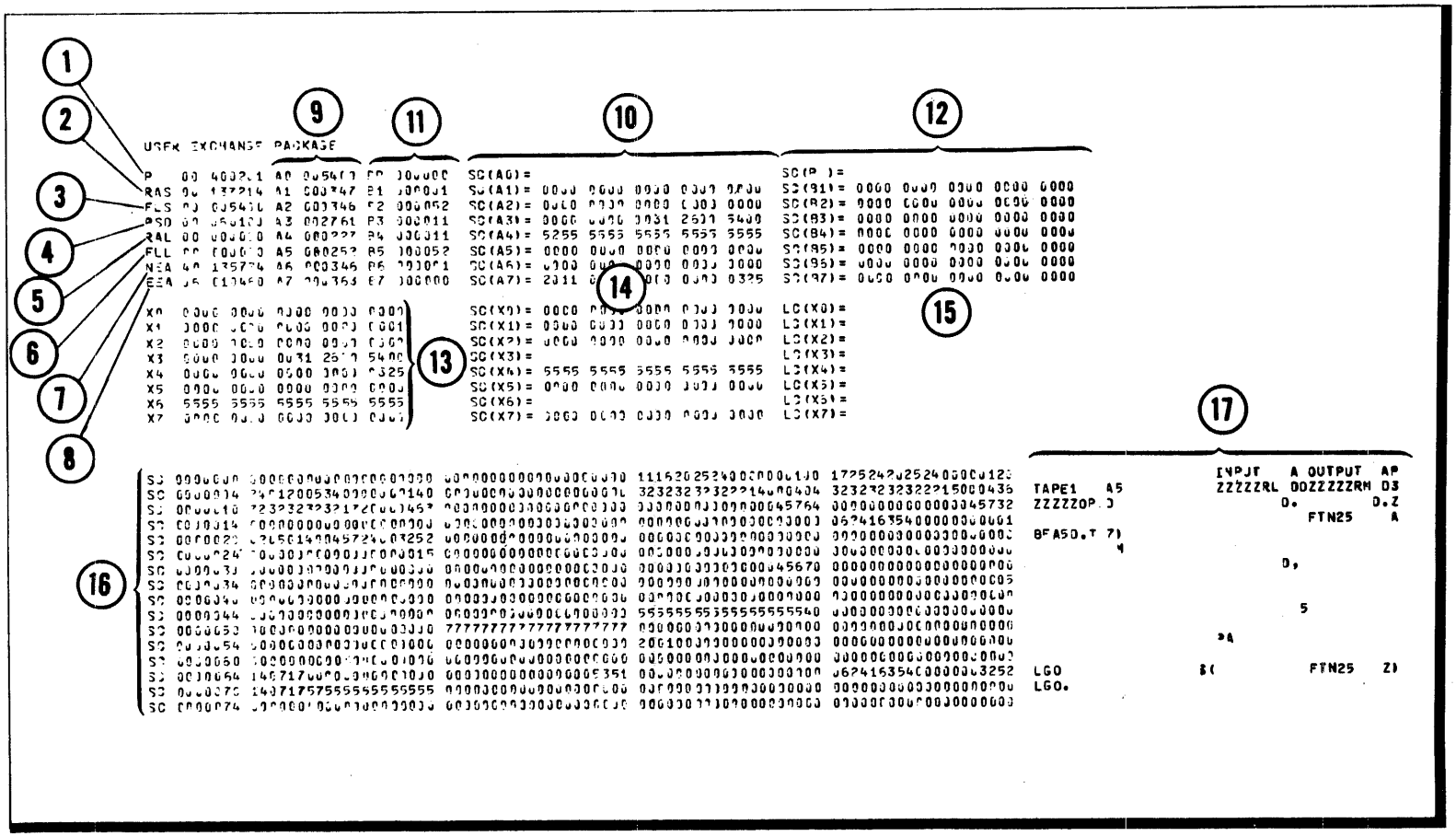


Figure G-10. User Exchange Package and Job Communication Area

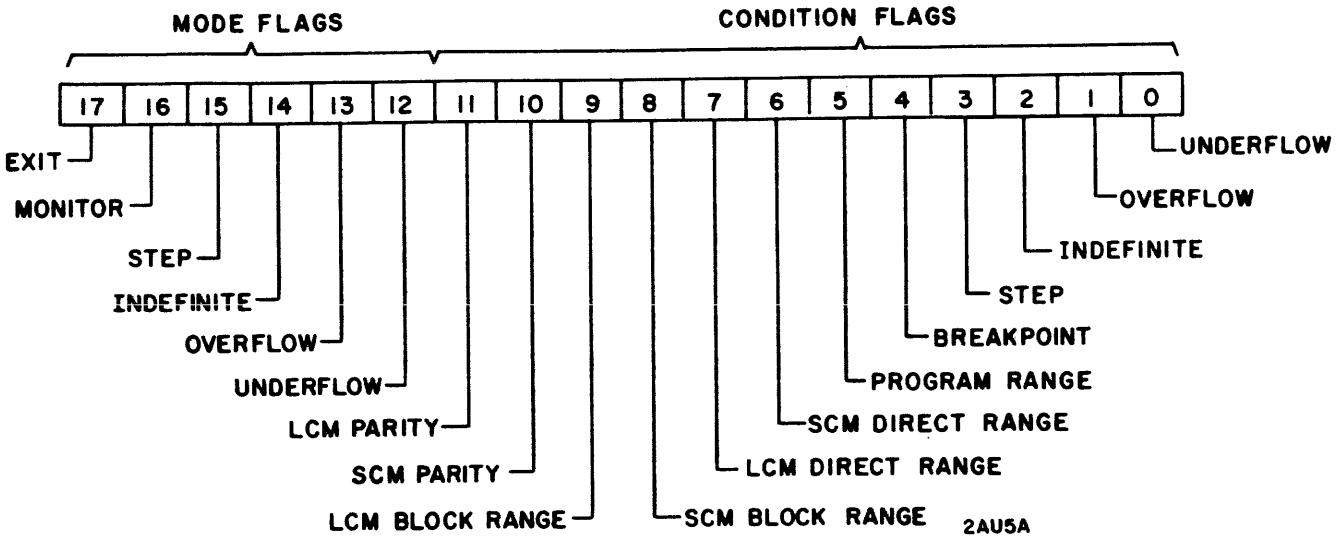


Figure G-11. PSD Register

The PSD register is loaded from the exchange package during an exchange jump sequence. All 18 bits are entered in the register at this time. The six mode designators remain unaltered throughout the execution interval for the exchange package. The 12 condition designators may be set by conditions that occur during the execution interval. All flags are stored in the SCM exchange package at the end of the execution interval.

Comparing this figure with the value 060100 in the field, we find that the indefinite and overflow bits are set and that the SCM direct range error bit is set. The mode flags are set through the MODE statement or by system default. In this case, 6 indicates that the system default is in use. Items 5 and 6 pertain to the use of large central memory. RAL indicates the beginning absolute location (reference address) of the user field in LCM; FLL indicates the size of the field. RAL and FLL reflect that no LCM was requested for this job step.

Items 7 and 8 in Figure G-10 give the normal exit address and the error exit address. The NEA gives the absolute location of the user exchange package. The EEA indicates the absolute address taken when the error was detected.

As noted in section 1, a CPU contains 24 operating registers with which the computer system performs arithmetic and logical operations. These 24 registers are grouped into three types:

- A registers, designated A0 through A7, which contain the addresses of operands in SCM
- B registers, designated B0 through B7, which have no connection with SCM, and generally are used for program indexing
- X registers, designated X0 through X7, which contain operands used in calculations or contain results of calculations.

The A and X registers are interrelated. When an address is placed in one of the registers A1 through A5, the contents of the address in SCM are loaded into the corresponding operand register, X1 through X5. That is, references to registers A1 through A5 imply SCM reads. Similarly, placing an address in register A6 or A7 causes the contents of the corresponding X register (X6 or X7) to be written in the SCM address.

Thus, registers X1 through X5 hold operands obtained from SCM for use in calculations; registers X6 and X7 hold results of calculations to be written in SCM. The A0 and X0 registers are used for holding intermediate operands and for executing instructions involving LCM data transfers. They have no functional connection with SCM or with any other registers.

The B registers are used for counters for such functions as program indexing. For example, a B register might be incremented each time a DO loop is traversed in a FORTRAN program, ultimately providing a terminating condition for the loop. The B registers have no functional connection with SCM. The B0 register provides a constant value of zero which can be used for tests against zero or for modifying an unconditional jump instruction in a program.

Now, look at item 9 in Figure G-10. You will see the A registers and their contents. Remember, A registers always contain addresses. Next, look at item 10. Here, you will find the contents of the SCM locations given in each corresponding A register. Notice that register A1 contains address **000347.** The content of SCM address **000347** is 0000 0000 0000 0000 0000.

The addresses in registers A1 through A7 should be within the job field allocated to your job. If an address appears here that equals or exceeds the relative address of the last word in the SCM field (in this case **5400),** an address out of range error is indicated.

When a FORTRAN Extended program calls an external subprogram, subroutine, or function, the contents of A1 indirectly point to the arguments passed to the external routine. The address in A1 is the SCM address of the first word in the parameter address list. Each address in the parameter address list points to an actual argument. For double precision or complex arguments, the argument address plus one contains the least significant or imaginary part of the argument.

The content of X1 is the same as the first word in the parameter address list. Thus, it is usually possible to determine the first argument value by looking at item 14, which says that if the content of X1 is an SCM address, then the content of the address indicated is as given.

Figure G-12 illustrates the relationship of the parameter address list to the contents of the A1 and X registers.

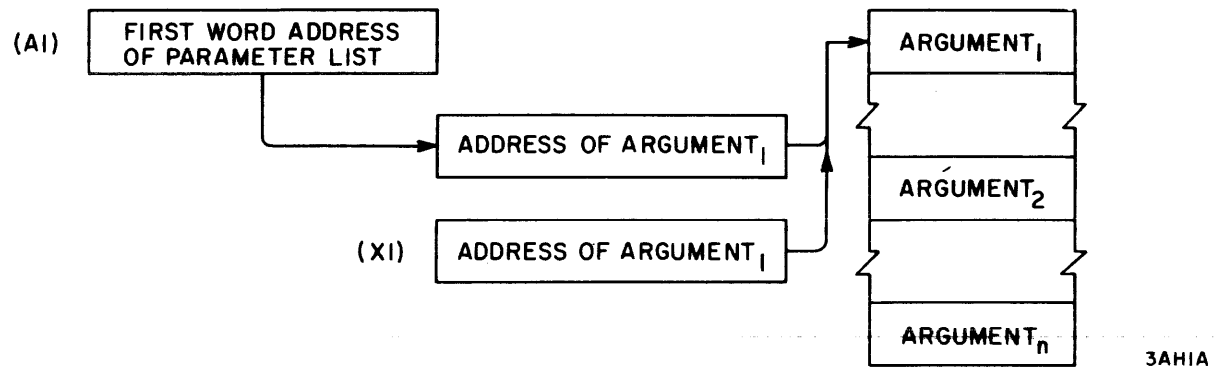


Figure G-12. Parameter Address List

Returning to Figure G-10, item 11 shows the contents of the B registers. If a B register contains an address, item 12 shows the contents of the address. Otherwise, item 12 shows only random numbers.

The X registers and their contents are shown in item 13. Assuming the contents of the X registers are addresses, the contents of the addresses in SCM and LCM (if applicable) are shown in items 14 and 15.

In a User Exchange Package, such as this, the primary areas of concern are the A and X registers. A quick glance at the A registers can often tell that a programming error has resulted in a reference to an out-of-range address. By looking at the X registers, you can frequently tell if an improper value is being used for an operand. In scanning the contents of the A and X registers, check the first four digits of each of these words; they will often tell you that a number is in error.

The special operand forms in octal are:

Positive overflow (+ ∞)	= 3777x------x
Negative overflow (- ∞)	= 4000x------x
Positive indefinite (+IND)	= 1777x------x
Negative indefinite (-IND)	= 6000x------x
Positive underflow (+0)	= 0000x------x
Negative underflow (-0)	= 7777x------x

When the CPU uses one of these six special forms as a floating point operand, only the following octal words can occur as results, and the associated flag is set in the PSD register.

Positive overflow (+∞)	= 37770------0 Overflow condition flag
Negative overflow (- ∞)	= 40007------7 Overflow condition flag
Positive indefinite (+IND)	= 17770------0 Indefinite condition flag
Positive underflow (+0)	= 00000------0 Underflow condition flag
Negative underflow (-0)	= 77777------7 Underflow condition flag

(In 6000 Series computer systems, negative overflow results in 40000-----0; negative underflow results in 00000-----0.)

The job communication area is given in item 16 of Figure G-10. Each line shows four words in 20 octal-digit groups. Item 17 gives the display code conversion of the words.

Appendix B shows how to interpret this area. For FORTRAN Extended programs, words starting with RAS+2 in the job communication area contain the file name and the SCM addresses of the first word of the FITs for the files used by the program. Thus, the FIT of the INPUT file is at SCM address 100, the FIT of the OUTPUT File is at SCM address 120, and the FIT of file TAPE1 is at SCM address 140. A user desiring a closer examination of an FIT could request a dump of the area containing the FIT using the DMP statement. A detailed description of the contents of an FIT is given in the Record Manager Reference Manual.

MEMORY DUMP OF PROGRAM BLOCK

Suppose that an error occurred that was not as easy to isolate as an unsatisfied external. Your Standard Dump, in such a case, will usually include a dump of the contents of the SCM field starting with a location approximately 100_8 words below the terminal address (P) and ending with a location about 100_8 words higher than the terminal address. In this example, the memory dump was not generated because the terminal address was out of range of the SCM field.

For the purpose of illustrating how to obtain and read a memory dump, we can generate a memory dump of the first 300 words of the sample program at the time of termination by adding an EXIT statement and a DMP statement to the job control statements. The control statement section then appears as shown in Figure G-13.

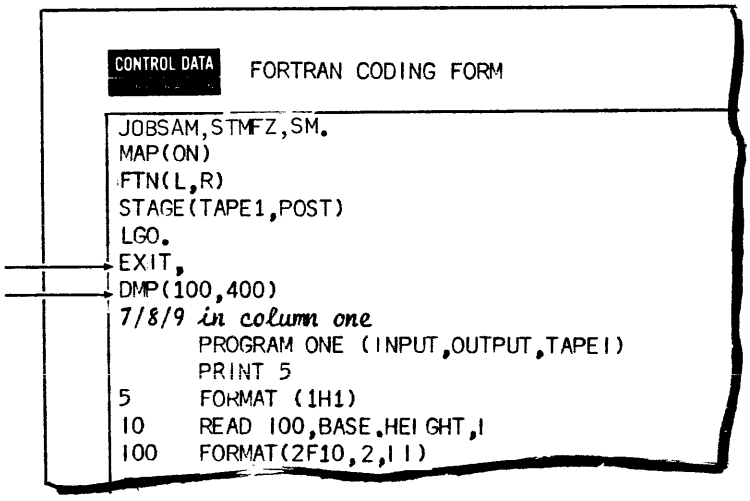
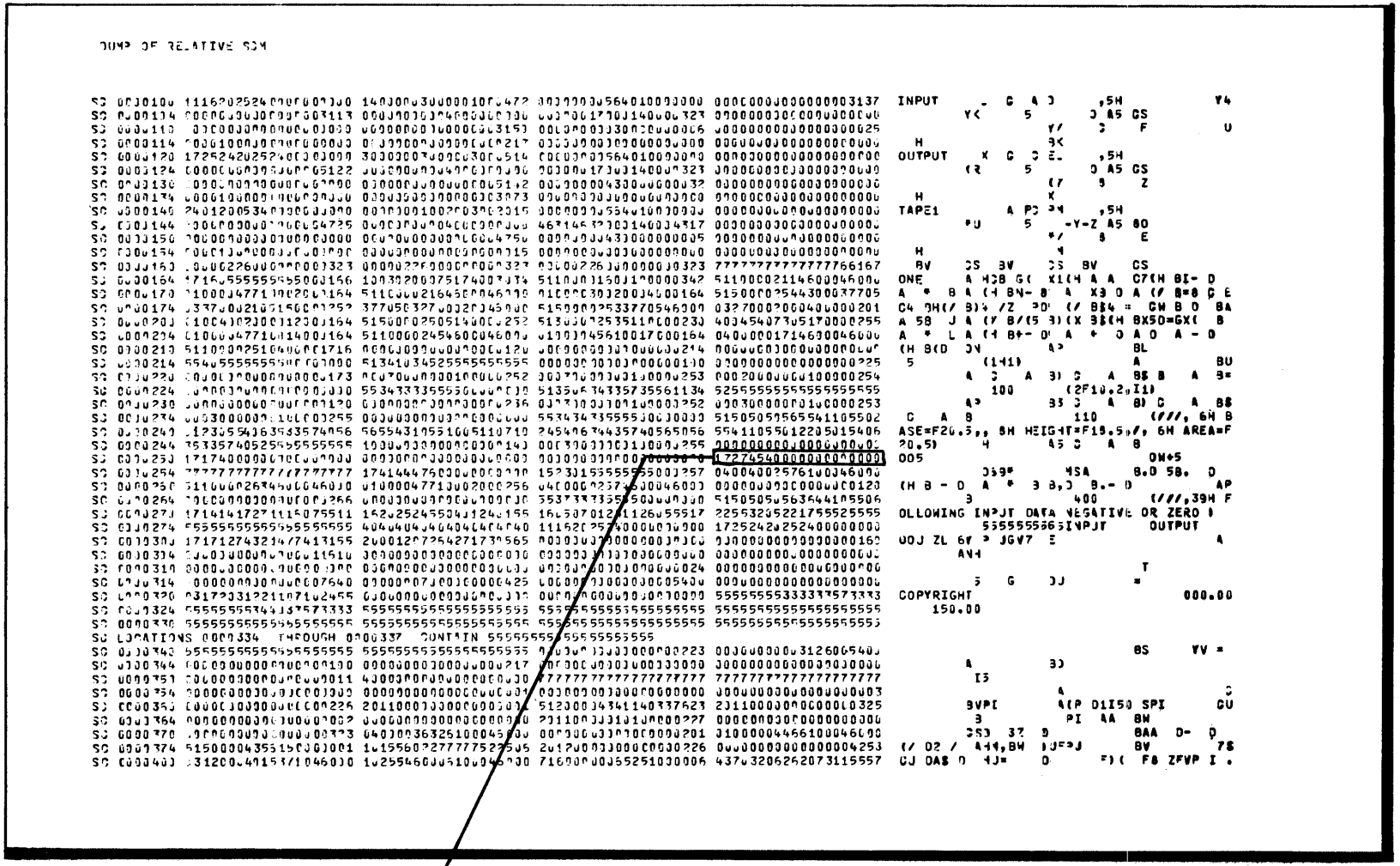


Figure G-13. Request Dump Following Job Abort

Now look at the dump in Figure G-14. This dump is printed before the dayfile in the output of the sample job. Each line shows four words in 20 octal-digit groups, plus the display code conversion of the words.

Look at address 000253 which has 17274540000000000000 as its contents. This number is the value of HEIGHT at the time the job terminated (this location can be determined from the reference map, a tool that will be examined later). Heights supplied in the data section included such numbers as 500.76, 600.76, and 150.00. Perhaps you expected an octal representation of one of these numbers to appear at the address 000253. If the numbers had been integers with no fractional part (no decimal point), they would indeed appear in simple octal. Because they are not whole numbers, however, they must be stored internally not only as octal values but also in floating point format.

How a programmer converts a number from floating point format to its decimal equivalent is difficult to describe. Programmers use the process primarily for debugging programs involving scientific applications. The following example of this process is illustrated step-by-step in case you have an occasion to use it.



CONTENTS OF ADDRESS 253

Figure G-14. Dump of Relative SCM

1. First, determine whether 17274540000000000000 represents a positive or negative value. Two methods are possible. First, you can look at the highest order octal digit. If it is 4 to 7, the number is negative. If it is 0 to 3, the number is positive.

   Second, you can prove the first method by converting the highest order digit to its binary equivalent. The leftmost bit is the sign bit. If it is zero, the number is positive. If it is one, the number is negative.

   Inside the computer, the group of three bits making up this digit 1 is:

   o o ● = 1
   Sign Bit

2. Because this number is positive, you can work with this form by continuing with step 3.

   If the number were negative, however, you would have to convert it to its octal complement by subtracting (in octal) the number from a row of 7's equal in length to the number itself. To obtain the complement of the number 5276 3224 0000 0000 0000, subtract as follows:

```
  7777 7777 7777 7777 7777
- 5276 3224 0000 0000 0000
  ----------------------
  2501 4553 7777 7777 7777
```

   Then attach a minus sign to this number and proceed with the following steps with -2501 4553 7777 7777 7777.

3. Next separate the biased exponent of the coefficient from the coefficient itself. This biased exponent is composed of the leftmost four digits (including the highest order digit).

   To help clarify some points, the machine and numerical binary representation of the entire contents of address 00272 is shown in Figure G-15.

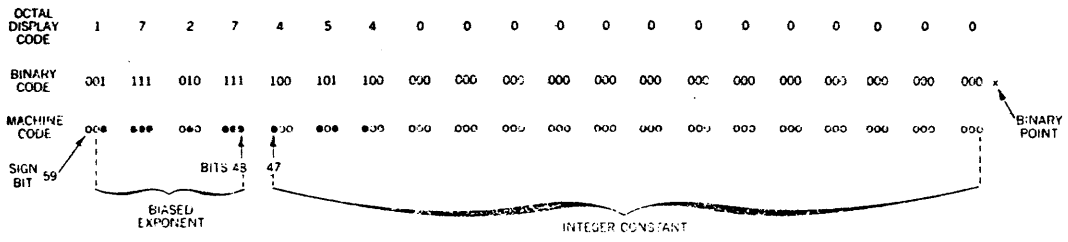


Figure G-15. Octal, Binary, and Machine Code Representation

The biased exponent is 1727, the coefficient is: 454 000 000 000 0000.

4. Now, remove the bias from the biased exponent. For positive numbers, the bias is 1777_8. For negative numbers it is 2000_8. Remove the bias by subtracting the biased exponent from the bias.

   exponent = bias - biased exponent

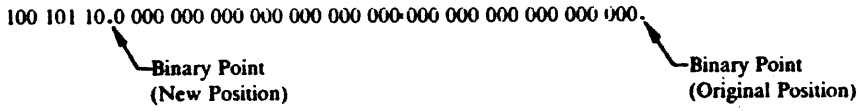
   The unbiased exponent for the example is:

```
  1777
- 1727
------
    50
```

5. Convert the unbiased exponent to a decimal number. Here, $50_8=40_{10}$.

6. The decimal equivalent of the unbiased exponent tells you the number of bits to the left (for a positive number) or right (for a negative number) to move the binary point to obtain the true binary value of the coefficient. In Figure G-14, you can see both the binary equivalent of the coefficient and the location of the binary point.

   According to previous calculation, the binary point must be moved 40 places to the left. Thus:



7. This number can be regrouped to translate into octal digits:

   Binary = 10 010 110. 000 000

   Octal = 2 2 6 . 0 0

   If any bit to the right of the binary point in the octal equivalent were one instead of zero, a fraction would be indicated. Further and more intricate conversion techniques are necessary to obtain the precise number for a fraction.

8. Convert the octal number 226.00 to its decimal equivalent. The correct equivalent is 150.00

   If you examine the data section in your program, you will see that 150.00 was, indeed, one of the values used for HEIGHT, and this value was stored in HEIGHT when the job terminated.

DUMP OF JOB SUPERVISOR AREA IN LCM

The dump of the job supervisor area in LCM (JSLCM) is primarily a SCOPE 2 system analyst's aid. Any user can become proficient in interpreting this dump, but there is little of value in the dump for the average FORTRAN user who is not intimately acquainted with the design of the operating system.

The operating system maintains a JSLCM in LCM for each job in the system. The JSLCM for this example (Figure G-16) is arranged in LCM in 1000$_8$-word increments. Words 1, 2, and 3 of the dump (item 1) give the beginning LCM addresses of each 1000$_8$-word increment (17250000, 1723000, and 1540000, respectively). Addresses in the JSLCM are relative to the beginning of the JSLCM, as if the first word were 0. This is sometimes called the offset address.

The JSLCM contains a File Description Table (FDT) for each file used by the job. Since much of the information in the FDT is the same as in the FIT generated for each file, the user may obtain valuable clues for isolating file or I/O problems by looking at the FDT.

Item 2 of the JSLCM dump contains in bits 35 through 21 the JSLCM address (426) of the first word of the FDT for the job control file. Looking at the contents of 426, the first word of an FDT contains the file name, and in the case of the job control file, the priority initially assigned to the job (item 3). Bits 47-33 of the eighth word of the FDT contain the pointer to the next FDT. Thus, the FDTs form a threaded list through the JSLCM. A zero pointer (item 4) indicates the end of the list.

Now, look at the FDT for TAPE1 (item 5), which begins at JSLCM address 2015. The fourth word contains a 14 in the rightmost position of the word (item 6). This is the current file position expressed in words (that is, it is a word address in the file). It indicates that the next word to be written on the file will be in word 15. (Incidentally, the FDT position is always one less than the corresponding field in the FIT.)

Next, look at the counter in the similar portion of the ninth word in the FDT (item 7). Here, we find another word address, this one indicating the end-of-information as a word address in the file. Notice that the EOI address is the same as the current position. This is normal for an output file. For an input file, however, an attempt to read the file at its current position when the file is at end-of-information results in a Record Manager error. An EOI address of 0 might indicate that the file was never created, possibly because a prestage did not occur.

Another value of interest in the FDT is bit 48 of word one (item 8). This bit indicates that the file is blocked (1) or unblocked (0).

When the control statement section contains FILE, REQUEST, STAGE, or LABEL statements for a file, the FDT contains a pointer to a chain of tables containing information taken from the statements. In this example, TAPE1 is the only file so affected. The pointer to the STAGE table is 002221 and is in the rightmost part of word 5 (item 9).

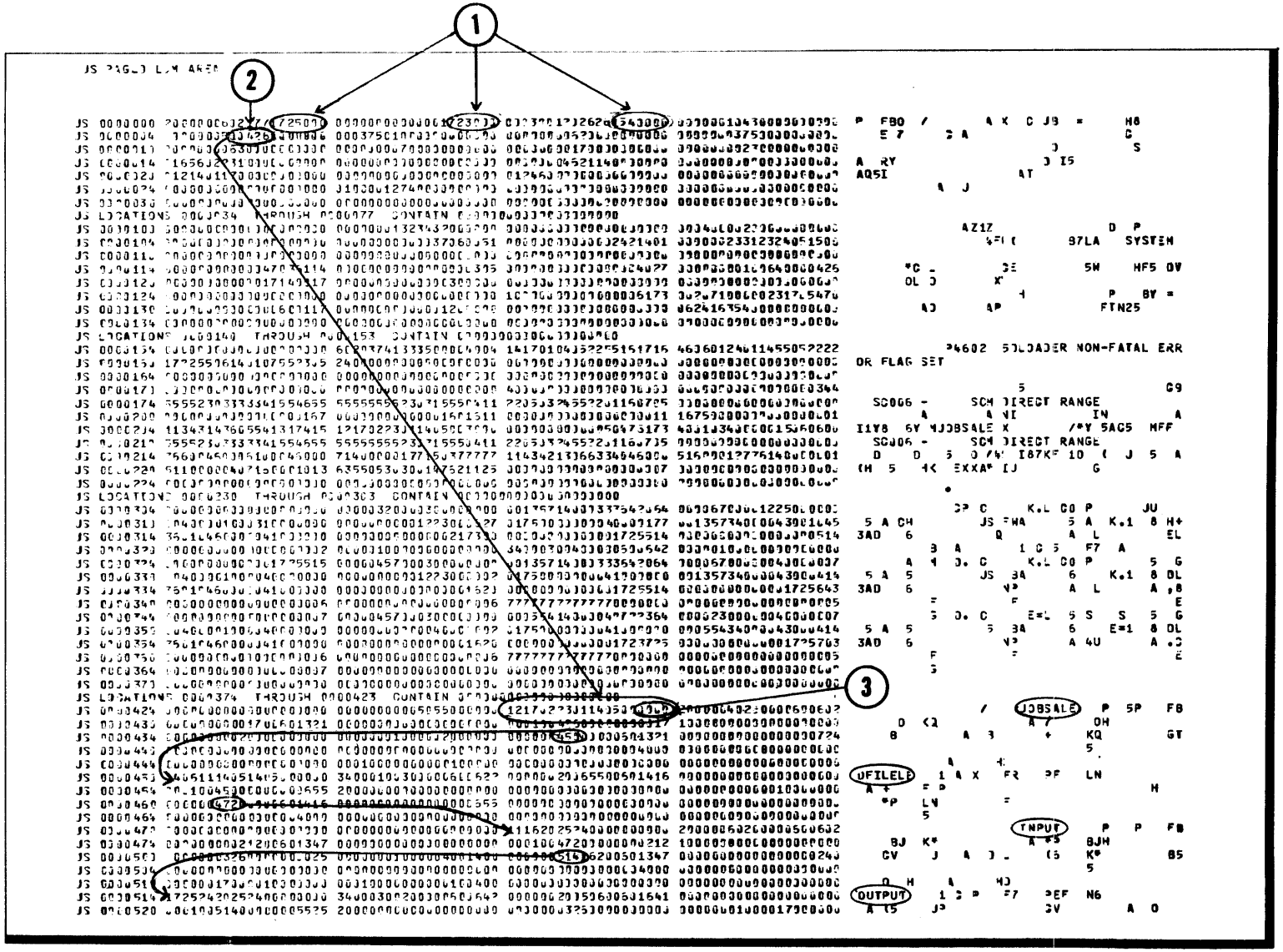


Figure G-16. Dump of Job Supervisor Paged LCM Area

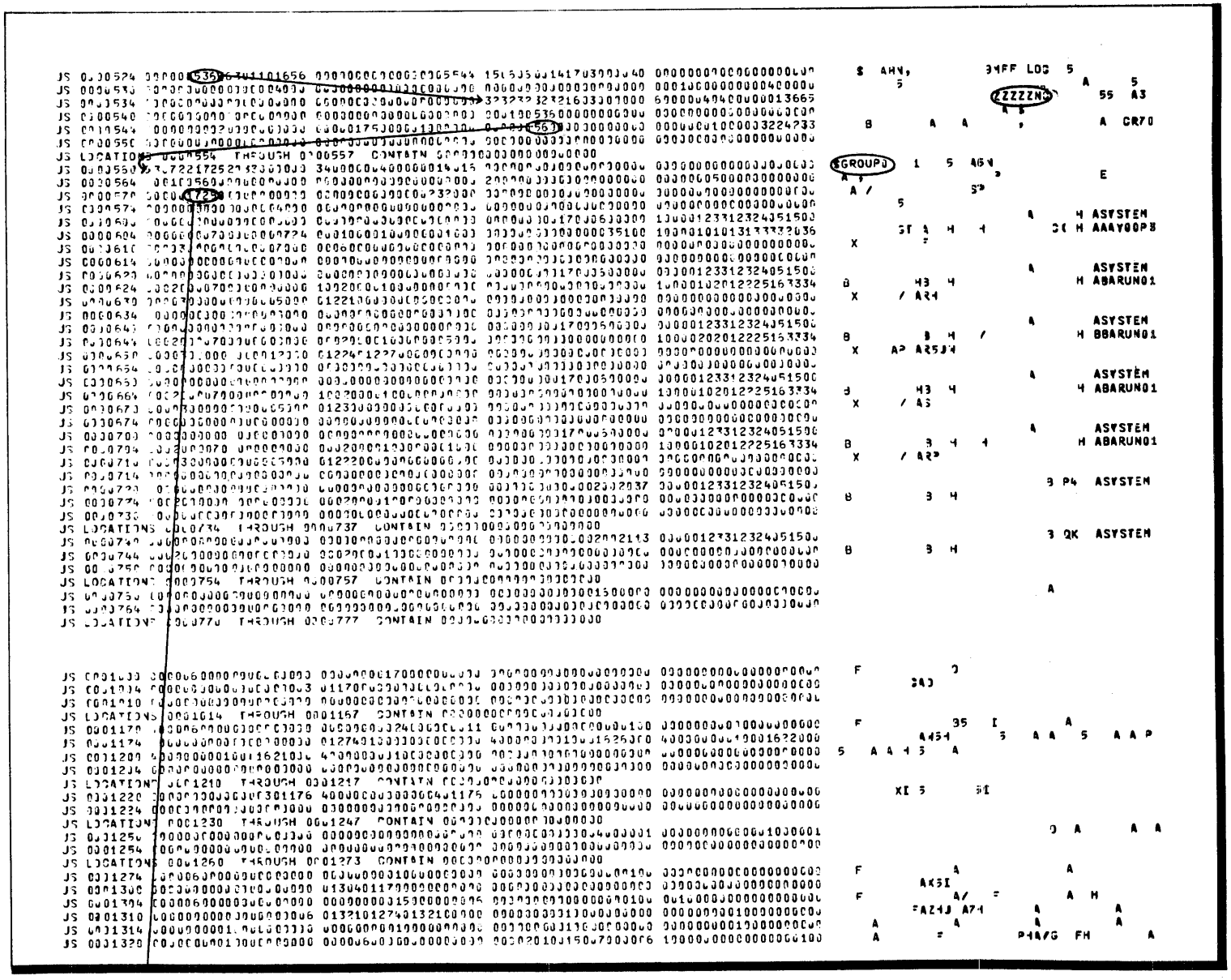


Figure G-16. Dump of Job Supervisor Paged LCM Area (Cont'd)

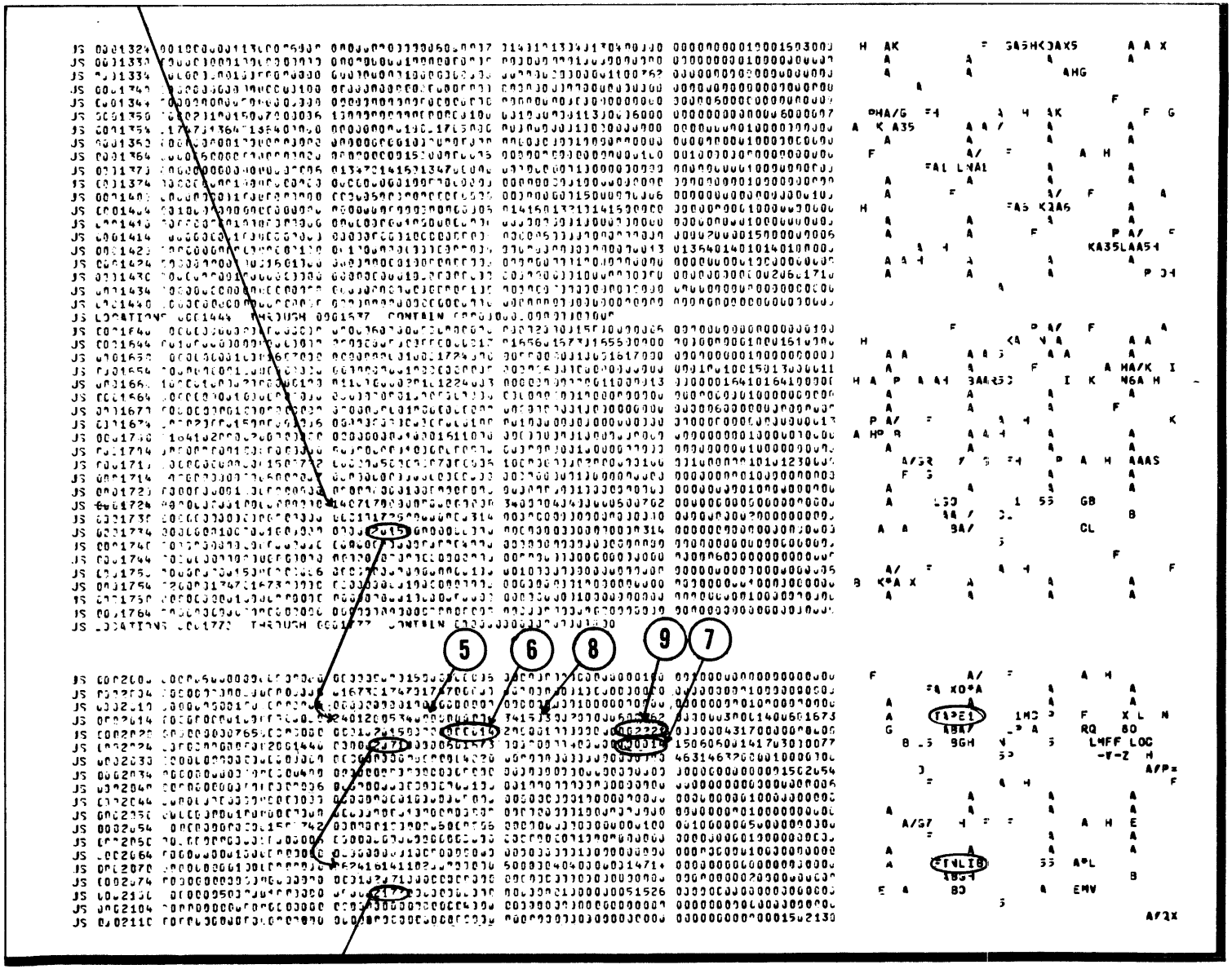


Figure G-16. Dump of Job Supervisor Paged LCM Area (Cont'd)

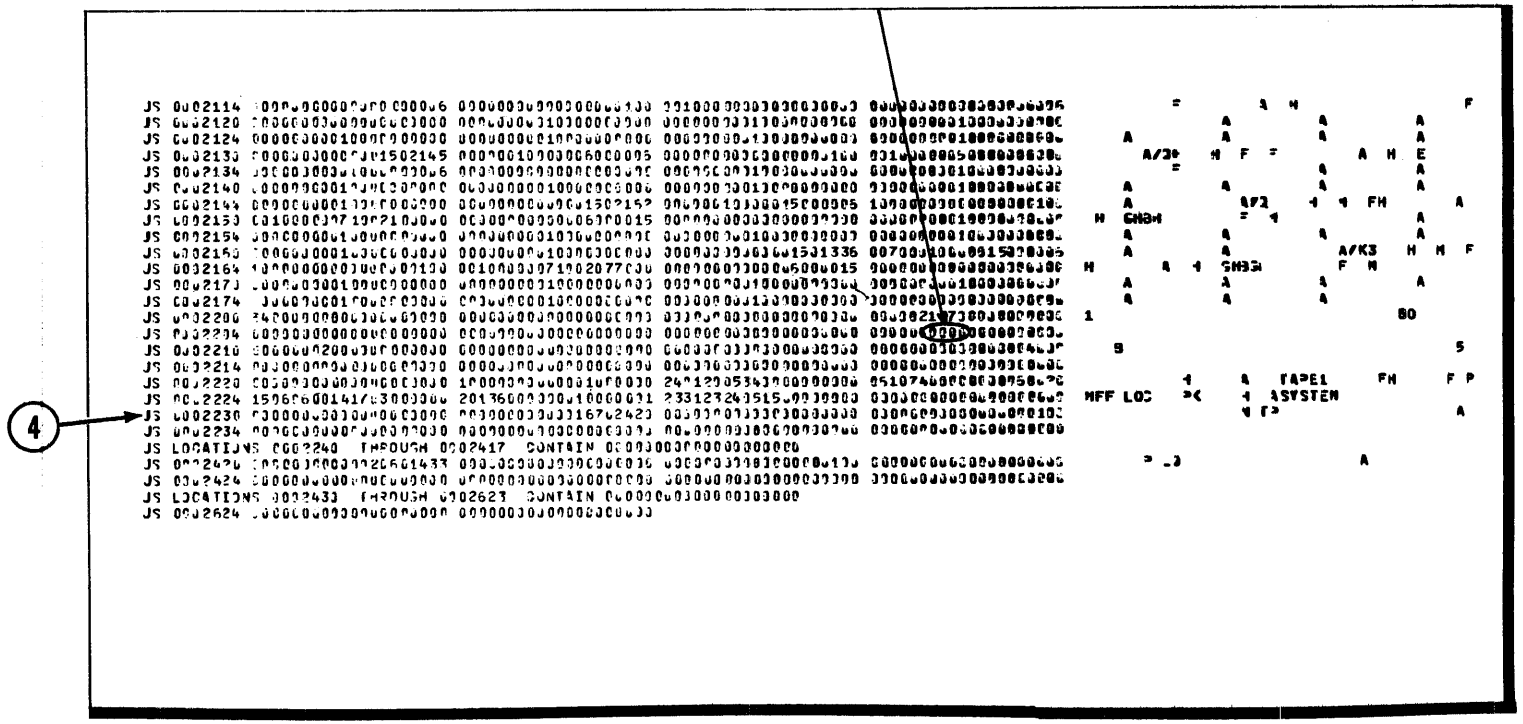


Figure G-16. Dump of Job Supervisor Paged LCM Area (Cont'd)

SYMBOLIC REFERENCE MAP

The reference map is a list of programmer created symbols in a program unit. The reference map appears on a separate page following the source listing of the program and the error dictionary. See Figures G-17 and G-18. Names of symbols generated by the compiler, such as those in library routines called for input/output, do not appear. Names are listed alphabetically within the following classes.

Entry points, variables, file names, external references, inline functions, NAMELIST group names, statement and FORMAT statement numbers, DO loops, common blocks, equivalence classes.

The programmer may select from three types of reference maps or suppress the map completely. The type of map produced is determined by the R option of the FTN control card.

R=0	No map
R=1	Short map (symbols, addresses, properties)
R=2	Long map (symbols, addresses, properties, references by line number, and DO loop map)
R=3	Long map, printout of common block members, and equivalence classes
blank	Implies R=1

If no selection is made, the default option is R=1. However, if the control statement option L is 0, no map is produced.

Fatal errors in the source program cause certain parts of the map to be suppressed, incomplete, or inaccurate. The DO loop map is suppressed, and assigned addresses are different when fatal execution or compilation errors occur. References are not accumulated for statements containing syntax errors.

The number of references that can be accumulated and sorted for the reference map is eight times the number of symbols in the source program if the field length is 50000 octal. For example, in a source program containing 1000 (decimal) symbols, with a field length of 50000 octal, approximately 8000 (decimal) references can be accumulated.

Although formats for portions of the reference map differ, they all contain the following information:

The symbol as it appears in the FORTRAN program

Properties associated with the symbol

List of references to the symbol; line numbers in the list refer to the first line of the statement. Multiple references on a line are printed as n*k where n is the number of references and k is the line number.

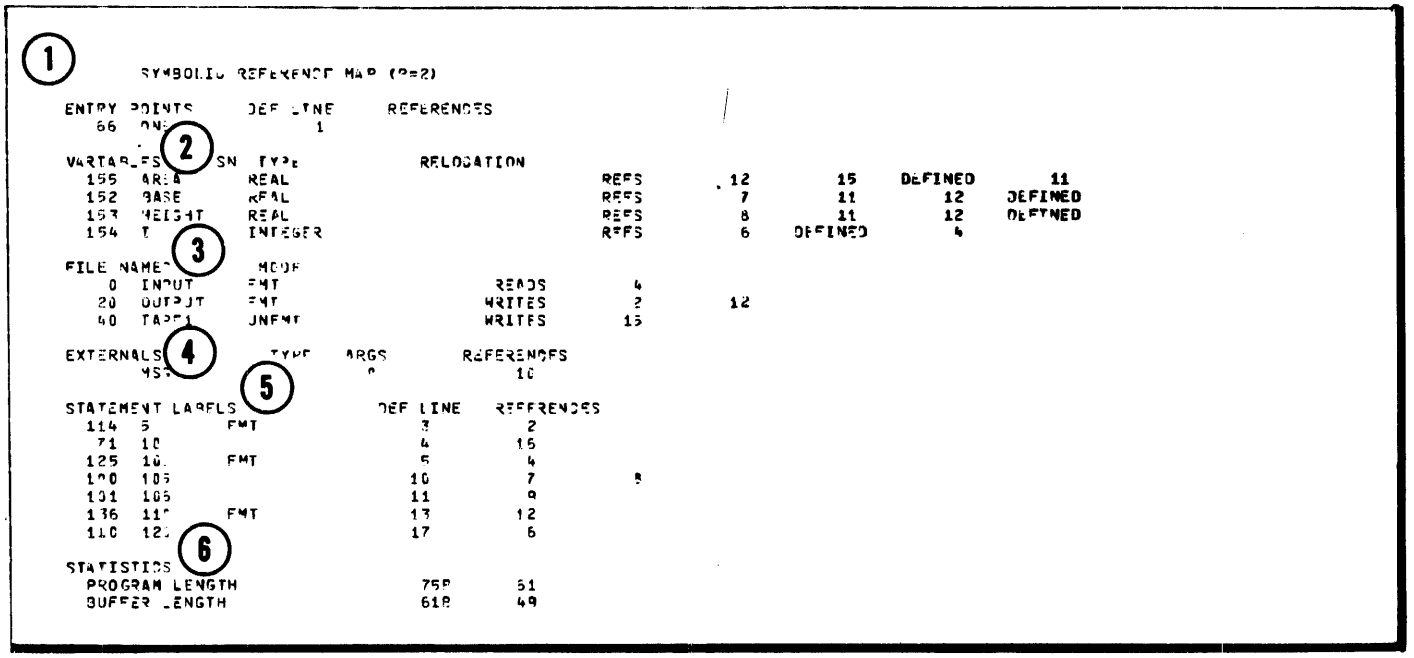


Figure G-17. Symbolic Reference Map for Main Program

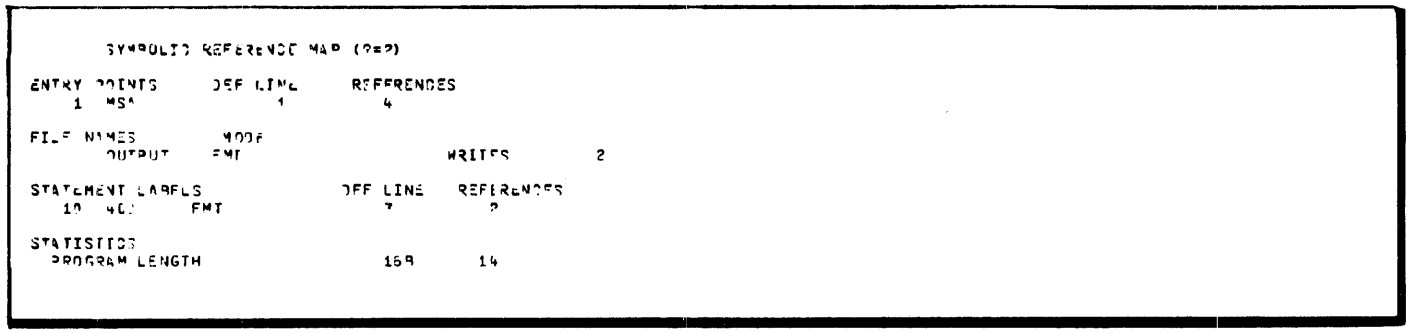


Figure G-18. Symbolic Reference Map for Subroutine

CLASSES

ENTRY POINTS

Entry point symbols (item 1, Figure G-17) include subprogram names and names appearing in ENTRY statements; they are printed in the reference map under the headings: ENTRY POINTS, DEF LINE, and REFERENCES.

ENTRY POINTS	Entry point name as it appears in the source program, and the program relative address
DEF LINE	Line number of subprogram statement on which entry point is defined
REFERENCES	Line number at which entry point is referenced (none for main program). RETURN statements constitute a reference to an entry point. References to a function value, in a function subprogram, appear in the variable map.

VARIABLES

Variable symbols (item 2) include variables and arrays (including those in common), dummy arguments, RETURNS names, and for a function subprogram, the function name when it is used as a variable.

VARIABLES	Object program or common relative address; 0 for dummy arguments, and variable name as it appears in source program
SN	Stray name flag. Variable names that appear only once in a subprogram are indicated by *. They are classified as stray, since they may be keypunched errors, misspellings, etc. A legal usage that would cause a name to be called stray is a DO loop in which the control variable is not referenced. (DO loops are mapped for R=2 and R=3 only.)
TYPE	Type of variable (logical, integer, real, double precision, complex). RETURNS is printed for RETURNS dummy arguments.
RELOCATION	Subdivided into properties, blockname, and references
PROPERTIES	The keywords UNDEF, ARRAY, UNUSED may be printed in this column.
*UNDEF	Symbol has not been defined. Variables used before definitions are not listed as undefined. For reference map purposes, a symbol is considered to be defined if any of the following conditions hold. • It appears in a COMMON or DATA statement. • It is a member of an equivalence group other than the base member. (The base member of an equivalence group is the member with the smallest address. In an array X(10), X(1) has the smallest object program address, X(10) the largest.) An undefined non-base member is not detected by the compiler.

- It is a simple variable or array element on the left-hand side of an assignment statement.
- It appears in an ASSIGN statement.
- It is the control variable in a DO loop.
- It is a simple variable or array element which appears as an argument in a subroutine or function call.
- It appears in an input/output list.
- It is a dummy argument.

ARRAY	Symbol is an array name.
*UNUSED	Symbol is an unused dummy argument. If no further information appears on the line and it is not a RETURNS argument, it is a simple variable.

BLOCKNAME

blank	Symbol is not in common; address is relative to program unit
F.P.	Dummy argument
//	Symbol is in blank common; address is relative to blank common
name	Name of labeled common block where symbol appears

REFERENCES

REFS	Number of times variable names appear in specification or assignment statements
DEFINED	Number of times names are defined. Definitions are listed for names appearing in DATA statements, control variables of DO loops, names defined in an ASSIGN or assignment statement, and names defined by READ or ENCODE/DECODE statements. Dummy arguments are defined in the subprogram header line.
I/O REFS	Input/output references are collected for symbols used as variable file names in input/output statements.

References to the function name in a function subprogram are listed in the VARIABLE map rather than the EXTERNALS map.

References are collected after statement functions are expanded; they are not collected for the arguments before the expansion.

FILE NAMES

File names (item 3, Figure G-17) include those used as logical file names (unit number) in the input/output statements or names declared as files on the PROGRAM statement in a main program. They are printed in the reference map under the following headings: FILE NAMES and MODE.

FILE NAMES	Object program relative address of the file information table (FIT) and name of file
MODE	Type of input/output operations performed. They may be formatted (FMT), unformatted (UNFMT), buffered (BUF), or MIXED (combination of FMT, UNFMT, BUF).

References are divided into categories:

READS	Input operations
WRITES	Output operations
MOTION	Positioning operations; rewind, backspace, and ENDFILE

EXTERNAL REFERENCES

External references (item 4, Figure G-17) include names of subroutines or functions external to a program unit. If the T or D option is specified on the FTN control card, intrinsic functions are compiled as external references. Library functions appear as external references when T or D is specified, as they are called by name. Names of system routines not explicitly called in the source program, such as those used for input/output and exponentiation, are suppressed.

External references are printed in the reference map under the following headings: EXTERNALS, TYPE, ARGS, and REFERENCES.

EXTERNALS	Symbol as it appears in source program
TYPE	
blank	Subroutine
NO TYPE	Conversion follows the same rule as for octal or Hollerith data
other	Real, integer, double precision, complex, logical
ARGS	Number of arguments used to reference the external symbol, followed by:
blank	Programmer defined function or subroutine
F.P.	Dummy argument
LIBRARY	Call by value library function
REFERENCES	Line numbers on which symbol was referenced

INLINE FUNCTIONS

Inline functions (not shown) include names of intrinsic and statement functions appearing in the subprogram. They are printed under the following headings: INLINE FUNCTIONS, TYPE, ARGS, INTRIN, SF, LINE, and REFFRENCES. Since INLINE functions were not used, this portion of the map is suppressed.

INLINE FUNCTIONS	Symbol name as it appears in the listing
TYPE	Arithmetic type; NO TYPE means no conversion is performed in mixed mode expressions
ARGS	Number of arguments with which the function is referenced
INTRIN	Intrinsic function
SF	Statement function
LINE	Blank for intrinsic functions; definition line for statement functions
REFERENCES	Lines on which function is referenced

NAMELIST GROUP NAMES

The NAMELIST group name (not shown) line on which it was defined and all references to the name are included in this class. Since NAMELIST names were not used, this portion of the reference map is suppressed.

STATEMENT AND FORMAT LABELS

Statement and format labels (item 5, Figure G-17) are listed under the headings STATEMENT LABELS, DEF LINE, and REFERENCES.

STATEMENT LABELS — Relative address of statement, followed by statement number. Inactive labels are printed with a zero address. A label becomes inactive when the compiler has deleted all references to it through optimization.

Inactive labels and DO loop terminators are listed as INACTIVE with a zero address.

The type and activity are listed after the statement number:

Type	blank	Executable statement label
	FMT	Format label
	UNDEF	Label is undefined
Activity	blank	Label is active or referenced
	INACTIVE	Label is an inactive statement label
	NO REFS	Label is defined as a format label, and it is not referenced

DEF LINE	Line number on which label appeared in source program
REFERENCES	Line in which label was referenced

DO LOOP MAPS

A DO LOOP MAP (not shown) is generated by the R=2 or R=3 option and is a printout of all DO loops in the source program and their properties. Loops are listed in order of appearance in the program.

LOOPS	First word object program address in octal of beginning of loop
LABEL	Label associated with loop terminator. If none is present, it is an implied DO loop in an input/output list. The index of the DO follows. If preceded by an asterisk, the index is kept in memory during the loop.
FROM-TO	Initial and terminal line numbers of DO loop
LENGTH	Number of object program words generated for body of loop
PROPERTIES	If loop can be fully optimized, one of these messages is printed:
OPT	Loop has no properties which inhibit full optimization.
INSTACK	The instruction stack is a group of 60-bit registers (12 in the 7600) in the CPU computation section that holds program instruction words for execution. INSTACK means the loop is small enough to fit in this instruction stack and will usually run two to three times faster than loops that do not fit in the stack.

If loop is not fully optimized by the compiler, the reasons are listed:

EXT REFS	Loop contains references to an external subroutine or function, or it is an input/output loop.
ENTRIES	Loop is entered from outside its range.
EXITS	Loop contains references to labels outside its range.
NOT INNER	Loop is not loop innermost in a nest.

COMMON BLOCKS

Common block symbols (not shown) include common block names and names declared in COMMON statements to be variables and arrays in common.

COMMON BLOCKS	Block name
LENGTH	Total block length

When R=1 or R=2 is specified, only the above information is listed. When R=3, the following details appear for each member declared in a COMMON statement.

MEMBERS	Relative address (distance from origin of common block)
BIAS NAME	Name of member of common block
(LENGTH)	Number of words allocated for member

If an equivalence class is linked to common, all members of the equivalence group become members of the common block. They are listed in the equivalence class printout.

EQUIVALENCE CLASSES

Equivalence symbols appear only when R=3 is specified. Thus, they are not shown in this example. All members of an equivalence group are listed. Any symbols added through linkage to common are not included.

EQUIV	*ERROR* class is in error (more than one member of an equivalence group is in common, or common block origin is extended by EQUIVALENCE statement or conflicting equivalencing attempted).
	BASE MEMBER — Class is in common
	blank — Other
CLASSES	If the equivalence group is not in common, the first member of the group (the member with the smallest object code address) is printed. If the group is in common, the name of the symbol in common which linked the equivalence group to the common block is printed. When an equivalence group is in common, the base member of the equivalence group is the first member of the common block.
MEMBERS	Equivalence group length
BIAS	Distance between an equivalence group member and the first member of the group. In an equivalence group A, B, C(10), C(1) is the base member, C(2) has a bias of 1, C(3) has a bias of 2, A and B have a bias of 9.
NAME	Name of equivalence group member
(LENGTH)	Number of words allocated for the member

Members of an equivalence group are printed in order of increasing bias. If the class is in error, the numbers associated with the class length and bias are meaningless.

PROGRAM STATISTICS

At the end of the reference map, program statistics (item 6, Figure G-17) are printed in octal and decimal.

PROGRAM LENGTH	Program length including executable code, storage for variables not in common, constants, temporaries, etc., but excluding buffers and common blocks
BUFFER LENGTH	Total space occupied by input/output buffers and FITs
COMMON LENGTH	Total common length, excluding blank common
BLANK COMMON	Length of blank common

DEBUGGING USING THE REFERENCE MAP

New Program:

The reference map can be used to find names that have been punched incorrectly as well as other items that will not show up as compilation errors. The basic technique consists of using the compiler as a verifier and correcting the FE errors until the program compiles.

Using the listing, the R=3 reference map, and the original flowcharts, the following information should be checked by the programmer.

Names incorrectly punched

Stray name flag in the variable map

Functions that should be arrays

Functions that should be inline instead of external

Variables or functions with incorrect mode

Unreferenced format statements

Unused formal parameters

Ordering of members in common blocks

Equivalence classes

Existing Program:

The reference map can be used to understand the structure of an existing program. Questions concerning the loop structure, external references, common blocks, arrays, equivalence classes, input/output operations, and so forth, can be answered by checking the reference map.

Item 1, labeled ENTRY POINTS, includes subprogram names and names appearing in ENTRY statements. For example, ONE, which is an entry point name as it appears in the FORTRAN Extended program, has a program relative address of 101. Under DEF LINE the 1 refers to the line number of subprogram statement or line on which ENTRY statement occurs. By referring to the source listing of PROGRAM ONE we see that ONE is referenced in line one.

Item 2, labeled VARIABLES, includes local and common variables and arrays, formal parameters, RETURN names, and for a function subprogram, the function name when used as a variable. In our sample program, the arithmetic mode (type) for the variables AREA, BASE, and HEIGHT is defined as REAL whereas the variable I is defined as integer.

The relative address for each variable is listed to the left of the variable. For example, the RAS of AREA is stored at address **000155** and is relative to the first address in PROGRAM ONE, address 000100. Remember, to obtain the address relative to the RAS, we must add the address in item 2 to the relative address at which the program was loaded. Therefore, to find the address of the variable AREA relative to the RAS we add:

000155 + 00100 = **000255**

Go back to the variable AREA in the symbolic reference map and notice the following information:

REFS 12 15 DEFINED 11

This information tells us that AREA is referenced by lines 12 and 15 in PROGRAM ONE and that AREA is defined by line 11 of the main program.

Note the statement DEFINED 4 for the variables BASE, HEIGHT, and I. This means that definitions are listed for names appearing in DATA statements and names defined by READ statements. By referring to line 4 in the PROGRAM ONE source listing, we encounter the READ statement: READ 100, BASE, HEIGHT, I.

Item 3, headed FILE NAMES, includes those names used as logical file names (unit number) in the input/output statements or names declared as files on the program statement in a main program. For example, consider the following information.

```
20      OUTPUT      FMT      WRITES      2      12
```

20	Address of the file information table associated with the file
OUTPUT	Formatted I/O
WRITES	Output operations
2 12	Print statements in lines 2 and 12 of PROGRAM ONE

Item 4, headed EXTERNALS, is external references that include names of subroutines or functions external to a subprogram. For example, MSG is the symbol name as it appears in the source program one. There are no arguments used to reference MSG. The number 10 under REFERENCES refers to the line in the source program on which the symbol MSG was referenced. Upon checking line 10 in the source program, we see the statement CALL MSG.

Item 5, headed STATEMENT LABELS DEF LINE and REFERENCES, is a label and format map which includes all statement labels used in the source program. For example, consider the following statement.

```
114      5      FMT      3      2
```

114	Program relative address
5	Label
FMT	Format number
3	Line number in which label appears
2	Line number in which label is referenced

Item 6, labeled STATISTICS, is a listing of program statistics printed in octal and decimal. Notice the length of the main program is octal 75 (or decimal 61). The buffer length is octal 61 (or decimal 49). The PROGRAM LENGTH includes code, storage for load variables, arrays, constants, temporaries, etc., but excludes buffers and common blocks. BUFFER LENGTH is the total space occupied by input/output buffers and file information tables.

SUMMARY

You have just been introduced to the basic techniques in using a dump to detect errors in your program. With these techniques and your familiarity with each program you write, you can usually trace errors without consulting a system analyst. But, as we pointed out at the beginning of this discussion, the best way to develop your ability to use a dump in error analysis is through practice.

In the sample job, the error can be easily corrected by repunching the card SUBROUTINE MSA to be SUBROUTINE MSG, exchanging these cards in the deck, and resubmitting the job to the computer. Other types of logical errors often can be corrected by simply rearranging the order of the cards. Still others demand major rewriting of sections of the program.

To summarize the steps you should take to detect errors in your program:

- Examine the printed output from your job to see if the kind of data that you expected is present.
- Check the dayfile to see if any error messages appear. If not, you can assume that your program ran successfully. If errors are noted, however, proceed with the following steps.
- Try to pinpoint the error by scanning the instructions in the printed listing. Mispunched cards can be detected by checking the listing against your coding sheet.
- Examine the program map to detect error messages that may be more explicit than those in the dayfile. If such an error message appears, return to the listing and try again to find the erroneous instruction. For programs written in the FORTRAN (RUN) language, errors noted often refer you to numbers assigned by the compiler to each instruction on the listing.
- Refer to the standard dump, checking the contents of the arithmetic registers and the words in memory to which they refer. Use the map and any other diagnostic aids, such as a symbolic reference table, to help interpret the dump.
- If the cause of your error does not appear within the limits of the standard dump, a more extensive dump might be helpful. This can be obtained by re-submitting the job with the appropriate DMP request.
- If you still cannot determine the cause of error, consult a systems analyst familiar with SCOPE 2.

GLOSSARY H

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	1. To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium). 2. To request from the device scheduler a drive on which to mount a mass storage device or a reel of magnetic tape.
ALLOCATION UNIT	The smallest amount of storage space (for example, five sectors of mass storage on a 7638 Disk Storage Subsystem or 819 High Capacity Disk Subsystem, or 1/8 of a cylinder for an 844-2 Subsystem) 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.)
CCL	Refer to CYBER Control Language.
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.
CLAIM	To indicate on the job statement how many drives are required for a job (refer to MT, NT, and YD parameters).
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.
CYBER CONTROL LANGUAGE (CCL)	Control statements that initiate tests, substitutions, transfers, and loops within the control statement section of a job. CCL assigns values to symbolic names, prints results in job's dayfile, determines status of a file, and processes control statements in a file other than the original job file.

CYCLE	One of the separate and distinct files under a permanent file name. Each cycle is identified by the permanent file name, cycle number, and user id.
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	See job dayfile.
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 trouble-shoot.
DEFAULT SET	The set to which files are assigned when no set is specified.
DEVICE, MASS STORAGE	A disk pack, an 819 subsystem, or one-half (a unit) of a 7638 Subsystem.
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.
DISMOUNT	A logical operation that disassociates a set member from a job.
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).
DRIVE	An equipment defined by an entry in the Equipment Status Table (EST)
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, 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.

EQUIVALENCE MODE

Keywords on the call statement in a procedure are matched to identical alphanumeric parameters on the header statement.

EXPLICIT DISMOUNT REQUEST

A user-initiated (control statement) dismount of a set member or system-initiated dismount following an ADDSET, DELSET, or LABELMS.

EXPLICIT MOUNT REQUEST

A user-initiated (control statement) request for a device, the device containing a master or member of a removable set.

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 in 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. Tape labels are not considered part of file data.

FILEMARK

Refer to tapemark.

FORMAL KEYWORDS

Parameters that can be substituted or passed from a call statement to the procedure body.

GLOBAL LIBRARY SET	A library set to be used for all subsequent loads in your job until you give the loader further notice.
IMPLICIT DISMOUNT REQUEST	A system-generated dismount of a member device occurring when a drive is required on which to mount another set member. This function can only occur if all files on the member device for this job are closed.
IMPLICIT MOUNT REQUEST	A system-generated request for a set member occurring automatically when access to the device is initially required by the job.
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:

VOL1
HDR1
EOV1
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 memory 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 memory 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.

LOGICAL FILE NAME — A symbolic name assigned to a file.

NUCLEUS LIBRARY — The library that contains most of the operating and product set members as program 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.

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 — A file that 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
4. Purged

Permanent files may have certain restrictions for their access such as:

1. Access only with a special password identifier
2. Read only access

PHYSICAL RECORD

Refer to block.

POSITIONAL MODE

One-to-one correspondence between the call statement keyword in a procedure and the keyword on the header statement.

PROCEDURE

Group of control statements that exist on a file separate from the job control statement section; a procedure can be common sequences of control statements that are saved as generalized procedures, or a complex sequence of control statements that can be given a name and called by a user who may not know how the procedure is done.

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 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 program 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 program image occupies only that portion of the field length specified by the user as being available.
PROGRAM IMAGE MODULE	The program image module is loader tables consisting of the program image. It can be saved on a file for subsequent reloading and execution.
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.
REMOUNT	To mount a member device that has been previously mounted and implicitly dismounted.
REMOVABLE DEVICE	A device that is logically removable from the system.
REQUEST	To specify the need for a system resource such as an on-line magnetic tape unit.
RETURN	The request by a job to the job scheduler to reduce a job's requirements for a particular type of device.
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 utilities that consider 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.

SET

A group of mass storage devices. Every mass storage device whether removable or nonremovable belongs to one - and only one - set.

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.

SYMBOLIC NAME

Alphanumeric character string that identifies the storage location of a numeric value.

SYSTEM LIBRARY

That file or group of files containing the program 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 SET

The set of mass storage devices containing the operating system and all system scratch and permanent files. The system set is logically nonremovable.

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 as 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 record format except S or X).

INDEX

A parameter 7-1
    REQUEST 7-12
    STAGE 7-12
Absolute modules 1-6; 3-10
Accept error 5-29
Access methods 5-26
Accessibility label field C-4, 6, 7
ACCOUNT statement
    Refer to SCOPE Reference Manual
Accounting information
    Control 12-21
    Dayfile 1-17
ADDSET control statement 7-6
AF on REQUEST 7-5
ALGOL compiler introduction 1-9
ALTER statement 8-16
Allocation unit 7-12
APEX 1-10
APT IV 1-10
Assembly 3-1
.* procedure command 13-32
ATTACH statement
    Removable set 8-16
    SCOPE 2 8-4
Automatic memory allocation 4-3
AY on REQUEST 7-5

Banner page 9-11
Bad data
    Accept 5-29
    Display 5-30
    Drop 5-30
BEGIN statement 13-19
Binary, free-form 9-8, 21
Binary, SCOPE 9-7, 20
Binary machine language program 3-10
BLOCK CONTAINS clause 5-9
Blocking
    C-type 5-14
    Character count 5-14
    Default on magnetic tape 5-14
    Exact 5-15
    E-type 5-15
    I-type 5-15
    Internal 5-15
    K-type 5-15
    Maximum block size 5-16, 17, 18
    Records per block 5-17
    Rules 5-18
    Unblocked 5-12
BT parameter on FILE statement
    Blocked 5-17
    Unblocked 5-14

C parameter on LABEL 11-3, 6
Cards, punched 9-1, 19
CATALOG
    Initial 8-2
    No passwords 8-4
    Passwords 8-8
    Removable set 8-15
    Subsequent 8-14
    SCOPE 2 8-1
Cards, punched 9-1, 19
Carriage control 9-11
CCL 13-1
Central processor unit (CPU) 1-2
CF parameter on FILE statement 10-12
Character sets
    ASCII A-1; 6-7; 9-3
    Display code A-1
    EBCDIC A-1; 6-7; 11-8
    Labels 11-8
    Magnetic tape 6-7
    Printer 9-10
    Punched card 9-3
Checkpoint/restart 1-7
CL parameter on FILE statement 5-6; D-13; E-10
Clock time on dayfile 1-17
CM parameter on FILE statement
    Labels 11-8
    Magnetic tape data 6-7, 19
CM parameter on job statement
    General description 2-3
    User control of SCM 4-5
CN control password/permission 8-11
COBLIB library 3-20
COBOL compiler
    Block types 5-17
    File description entries 5-9, 17
    Introduction 1-9
    Object-time file names 1-23
    Record types 5-9
    Statement 3-1
Coded mode on magnetic tape
    CM parameter 6-7; 11-8
Comma as separator 2-6

COMMENT statement 4-11
Comments
After terminator 2-7; 4-11
COMMENT statement 4-11
Dayfile 1-18
PAUSE statement 4-12
To operator 4-12
VSN statement
Refer to SCOPE 2 Reference Manual
Configuration
Hardware 1-2
Software 1-5
Continuation of statement 2-7
COMPARE statement 10-12
COMPASS assembler
Introduction 1-10
Statement 3-1
Compilation 3-1
Computation section 1-2
Computer system
7600 1-2
CDC CYBER 70/model 76 1-2
Continuation cards 2-7
Control password/permission 8-11
Control statements
ACCOUNT (refer to SCOPE 2 Reference Manual)
ADDSET 7-7
ALGOL 3-1
ALTER 8-16
ANALYZE (refer to SCOPE 2 Reference Manual)
ATTACH 7-7; 8-4,20
AUDIT (refer to SCOPE 2 Reference Manual)
BEGIN 13-19
BKSP 10-10,11,12
CATALOG 8-3,14,20
COBOL 3-1
COMMENT 4-11
COMPARE 10-12
COMPASS 3-1
CONTENT (refer to SCOPE 2 Reference Manual)
COPY 10-2
COPYBCD (refer to SCOPE 2 Reference Manual)
COPYBF 10-4
COPYBR 10-3
COPYCF 10-4
COPYCR 10-3
COPYL (refer to SCOPE 2 Reference Manual)
COPYLB (refer to SCOPE 2 Reference Manual)
COPYLM (refer to SCOPE 2 Reference Manual)
COPYP 10-4
COPYR 10-3
COPYS 10-3
COPYSBF 9-14
COPYSP 9-14
DELSET 7-7,11
DISPLAY 13-6
DISPOSE 9-15,21
DMP 12-10
DMPECS 12-12
DMPFILE 12-17
DMPL 12-12
DSMOUNT 7-7,11
DUMPF (refer to SCOPE 2 Reference Manual)
ELSE 13-12
ENDIF 13-14
ENDW 13-15
Entry point name 3-9
EXECUTE 3-12
EXIT 12-1
EXTEND 8-16
FILE 5-1
File name call 3-8
FTN 3-1
IFE 13-10
INPUT 3-7
Job identification 2-1
LABEL 11-2
LDSET 3-17,21; 4-10,17; 12-8,17
LGO 3-4
LIBEDT (refer to SCOPE 2 Reference Manual)
LIBLOAD 3-22
LIBRARY 3-20
LIMIT 7-19
Listed in dayfile 1-18
LOAD 3-12,23
Loader 3-8
LOADPF (refer to SCOPE 2 Reference Manual)
MAP 12-14
MODE 12-6
MOUNT 7-9
Name call 3-7
NOGO 3-13
On dayfile 1-18
PASSWRD (refer to SCOPE 2 Reference Manual)
PAUSE 4-12
Processing 3-15
PURGE 8-18
REDUCE 4-9
REQUEST 6-15; 7-1
RETURN 6-21; 7-18
REVERT 13-22
REWIND 10-11
RFL 4-7,8

RTRVSIF (refer to SCOPE 2 Reference Manual)
RUN 3-1
Section in job deck 2-6,10
SET 13-5
SETNAME 7-5,10; 8-16
SIMI5 3-1
SKIP 13-12
SKIPB 10-10,11,12
SKIPF 10-7,8,9
SLOAD 3-16,23
SORTMRG (refer to Sort/Merge Reference Manual)
STAGE 6-3
SUMMARY 12-21
SWITCH 4-13
System verb 3-8
Syntax 2-6
TRANSF 4-15
TRAP (refer to Loader Reference Manual)
UNLOAD 6-21; 7-18
UPDATE (refer to Update Reference Manual)
VSN (refer to SCOPE 2 Reference Manual)
WHILE 13-15
Conversion
7-track 6-8
9-track 6-8
COPY statement 10-2
COPYBF statement 10-4
COPYBR statement 10-3
COPYCF statement 10-4
COPYCR statement 10-3
COPYP statement 10-4
COPYR statement 10-3
COPYS statement 10-3
COPYSBF statement 9-13
COPYSP statement 9-14
Copying labeled tapes 11-6
CP parameter on job statement 2-3,4
CP parameter on FILE statement 5-5; D-13; E-10
CPU 1-2
CPU time
Dayfile 1-18
Total for job 1-19
Creation date, label 11-3,6
Creator identification 8-2
CYBER Control Language (CCL) 13-1
CYBER 70/model 76 1-2
Cycle number 8-1
CY parameter 8-7
D record type 5-6; D-8, E-7
D parameter on job statement 2-3; 4-14
.DATA procedure command 13-28
Date, label creation 11-3,6
Date, label expiration 11-3,6
Date on dayfile 1-17
Dayfile 1-17; 4-1
Decimal count records 5-5; D-8; E-8
DELSET statement 7-11
Density 6-5,15
Dependency count 4-14
Device, mass storage 7-5,6
Directives 2-8
Disk storage subsystem 1-4
Display bad data 5-30
DISPLAY statement 13-6
Dispose file return 7-18
DISPOSE statement
Delayed 9-15
Forms control 9-15,22
Placement 9-15,21
Printer codes 9-15
Punch codes 9-21
Station identification 9-16
Use 9-15,21
DMP statement
Placement in load sequence 3-14
Use 12-10
DMPECS statement 12-12
DMPFILE statement 12-17
DMPL statement 12-12
Dollar sign for literals 2-7
Double EOS 3-10
Drop bad data 5-30
DSMOUNT statement 7-11
DT function (refer to SCOPE 2 Reference Manual)
Dynamic field assignment 4-3
E parameter on LABEL 11-3,4
E parameter on REQUEST 11-2
EB parameter
REQUEST 6-9,17
STAGE 6-9
EC parameter
Introduction 2-3
User control of LCM 4-8
ELSE statement 13-12
ENDIF statement 13-14
End-of-data exit 5-27
End-of file label C-6
End-of-file label group C-1
ENDW statement 13-15
EO parameter on FILE statement 5-29
.EOF procedure command 13-31
.EOR procedure command 13-31

End-of-file
Refer to end of partition
End-of-information 2-9
End-of-partition
Blocked files 5-19
Card 2-9
End load 3-10
S records 5-19
Z records, C blocked 5-19
W records 5-19
End-of-record card (refer to end-of-section)
End-of-section
Card 2-8
Double to end load 3-10
W records 5-21
Z records, C blocked 5-21
End-of-volume label C-7
End-of-volume label group C-1
Entry point name call 3-9
Entry point search 3-19
EO parameter on FILE statement 5-29
EOF
Refer to end-of-partition
EOF1 label C-6
EOI
Refer to end-of-information
EOP
Refer to end-of-partition
EOR
Refer to record type
EOS
Refer to end-of-section
EOV1 label C-7
Equal sign as separator 2-7
ERR loader option 3-17; 12-8
Error exit conditions 5-28
EX Extend password/permission
ATTACH 8-9
Except read password 8-12
EXECUTE statement 3-12
Execution
Explicit call 3-12
Inhibited 3-13
Load and execute 3-4
Execution time limit 2-4
Limit 2-4
Used 1-19
Exit conditions 5-27
EXIT statement 12-1
Expiration date, label 11-3,6
Extend password 8-9
EXTEND statement 8-16
Externals, search for 3-19
F record type 5-5; D-7; E-6
Fabricated job name
Banner page 9-11
Formation 2-1
Lace card 9-19
Features of SCOPE 2 1-1
Features, undescribed v
File
Refer to logical file
File definition 5-1
FILE function 13-8
File header label C-5
File identifier field 11-3,4
File information table 5-1
File name 1-20; 5-1,2
File name substitution
FORTRAN PROGRAM statement
1-21; 3-6
Load-and-go statement 3-6
File section number field 11-3,4
File sequence number field 11-3,4
FILE statement
BT parameter 5-14,17,18
CF parameter 10-12
CL parameter 5-6; D-13; E-10
CM parameter 6-7; 11-8
CP parameter 5-6; D-13; E-10
EO parameter 5-29
FL parameter 5-4,12; D-5,8; E-1,6,7
FO parameter 5-25
HL parameter 5-5; D-13; E-10
LL parameter 5-5; D-9; E-8
Logical file name 5-1
LP parameter 5-5; D-9; E-8
MBL parameter 5-16
MRL parameter 5-11; E-1
Multiple 5-2
OF parameter 10-12
PD parameter 5-26
Placement 5-1
RB parameter 5-16
RMK parameter 5-5; D-11; E-9
RT parameter 5-2; D-1
SPR parameter 6-16
TL parameter 5-6; D-13; E-10
FILMPL 1-21
FILMPR 1-21
FIT 5-1
Fixed length records 5-5
FL parameter on FILE statement
F record type 5-5; D-8; E-7
Related to MRL 5-12; E-1
Z record type 5-5; D-5; E-6
FLPP 1-4
FO parameter on FILE statement 5-26

Forms control 9-15, 22
FORTRAN compiler
Equating file names 1-21; 3-6
Introduction 1-8
Object-time file names 1-20
Record types 5-7
Statement 3-1
FORTRAN libraries 1-9; 3-20
Functional units 1-2
Function (CCL) 13-8

G parameter on LABEL 11-3, 4
Generation number 11-3, 4
Generation version 11-3, 4
GETPF
SCOPE 2 8-5
SCOPE 3.4 8-20
Global library set 3-20
Guide, purpose of iii

HARDPL 1-20
HARDPR 1-20
Hardware configuration 1-2
HDR1 label C-5
HI parameter
REQUEST 6-6, 15
STAGE 6-6
HL parameter on FILE statement 5-6; D-13; E-10
HY parameter
REQUEST 6-6, 15
STAGE 6-6

ID parameter for permanent file 8-2
I/O buffers in LCM 4-1
INPUT file
ASCII coded 9-4
Free-form binary cards on 9-8
Hollerith coded 9-4
Introduction 1-14, 20
Load from 3-7
Positioning 9-3; 10-11
Punched card format 9-3
Rewind 3-7
SCOPE binary cards on 9-7
Source statements on 3-3
Unblocked required 9-1
W records required 9-1
INPUT statement 3-7
Input/output multiplexer 1-3
Instruction stack 1-2

Job
Dayfile 1-17
Flow 1-13
Initiation 1-14
Maximum number 1-1
Priority 2-5
Processing 1-16
Standard files 1-20
Step 2-8
Termination 1-17
Job communication area 3-10; B-1
Job control file 1-14; 4-1
Job identification statement 2-1
Parameters 2-3
Syntax 2-1
Job name 2-1
Job on dayfile 1-17
Job supervisor 4-1

Keyword parameters 2-7
Keyword substitution 13-25

L parameter on LABEL 11-3, 4
L parameter on REDUCE 4-8
Label
Character conversion 11-8
Content C-1
Density 11-8
Exit conditions 5-27
Fields checked 11-4
Fields generated 11-2
Groups C-1
Magnetic tape 6-1
Parity 11-8
Protection 11-6
Standard 11-1; C-1
LABEL statement 11-2
Lace card 9-19
LCM
Allocation 4-1
Maximum available 4-2, 3
Used by job 1-19; 4-1
User control 4-8
LDSET loader statement
ERR option 3-17; 12-8
Introduction 3-17
LIB option 3-17, 21
MAP option 3-17; 12-17
NOREWIN option 3-17; 4-15
PRESET option 3-17; 4-10
REWIND option 3-17; 4-17

Level Number
COMPARE statement 10-13
Partition 2-9
SCOPE logical record 5-18,20; 10-13
Section 2-8
SKIPB statement 10-10
SKIPF statement 10-7
Z record 5-19,21
LGO file
Load from 3-4
Used by compilers, assemblers 3-4
LGO statement 3-4
LIB loader option 3-17,21
LIBEDT 3-18
LIBLOAD loader statement 3-22
Libraries
Definition 3-18
Global 3-19
Local 3-19
NUCLEUS 3-19
Sets 3-19
System 3-19
User 3-19
Library organization of a file 5-26
Library set definition
Global 3-20
Local 3-21
LIBRARY statement 3-20
LIMIT control statement 7-19
Literals 2-7
LL parameter on FILE statement 5-5; D-9; E-8
LO parameter
REQUEST 6-6,15
STAGE 6-6
Load
Call for 3-12
Followed by execution 3-4
From libraries 3-18,22,23
From multiple files 3-12
From INPUT 3-7
Introduction 1-6
Maps 12-13
Order of search 3-8
Sequence 3-14
Statements 3-8,14
Without execution 3-13
Load-and-go file 3-4
LGO 3-4
Renamed 3-5
Substituted names 3-6
Load sequence 3-14
LOAD statement
Load from file 3-12,23
Load from library 3-16
Loader statements
EXECUTE 3-12
INPUT 3-7
LDSET 3-17,21; 4-8,15; 12-8,17
LGO 3-4
LIBLOAD 3-22
LOAD 3-12,23
Name call 3-7
NOGO 3-13
SLOAD 3-16
Local library set 3-21
LOD on dayfile 1-17
Logical file
Active for job 1-20
Blocking 5-14
COBOL block types 5-17
COBOL object-time name 1-23
COBOL record types 5-9
Conversion, character 6-7; 11-8
Conversion, record or block E-1
Copying 10-1
Data transfer requests 1-18
Equating file names 1-21
Error recovery 5-28; 6-21
Even mode 6-7
FORTRAN object-time name 1-20; 3-4
FORTRAN record constraints 5-7
INPUT file 1-14,20; 3-7; 9-1
Introduction 1-19
Labeled 11-1
Magnetic tape 6-1
Mass storage 7-1
Maximum used by job 1-18
Naming conventions 1-20; 3-4; 5-1
No recovery 6-21
Odd mode 6-7
Open/close requests 1-18
OUTPUT file 1-16,20; 9-10
Parity 6-7
Positioning requests 1-18
Processing direction 5-27
PUNCH file 1-20; 9-20
PUNCHB file 1-20; 9-20
Record types 5-2,8; D-1; E-1
Suppress read-ahead 6-19
System files 1-20
Unblocked 5-12
Write check 7-16
LP parameter on FILE statement 5-6 D-9; E-8

M parameter on LABEL 11-3, 4
Magnetic tape files 6-1
Magnetic tape units
  Multiple amount 6-19
  On-line 1-4; 6-15
  Recovery 6-21
  Requesting 6-15
  Staged 6-3
  Staged on-line 6-6
  Types 6-5
Maintenance control unit 1-4
MAP loader option 3-17; 12-17
MAP statement 12-14

  Inside load sequence 3-14
Mass storage
  Job limit 7-19
  Sets 7-2
  System 1-4; 7-3
  Used 1-19
MAU 7-12
Maximum block length 5-14, 15
Maximum record length 5-12; 10-6; E-1
MBL parameter on FILE statement 5-17
MCU 1-4
MD modify password/permission 8-10
Memory
  Allocation 2-3; 4-1
  Instruction stack 1-2
  LCM 1-2; 4-6
  Presetting of 4-10
  SCM 1-2; 4-5
  Size 1-2
Messages
  Dayfile 1-17
  Operator 4-10
    (refer to Comments)
MF parameter on REQUEST 6-18
Minimum allocation unit 7-12
MODE control statement 12-6
Modify password/permission 8-10
Mount option 6-19
MOUNT statement 7-9
MR parameter 8-13
MRL parameter on FILE statement 5-12; 10-6; E-1
MT parameter
  Job statement 2-3; 6-15
  REQUEST 6-15
  STAGE 6-6
Multifile name 6-18
Multifile volumes 6-18; C-2
Multiprogramming 1-1
Multiread access 8-4, 5, 13
Multivolume file C-2
Multivolume multifile C-3
MUX 1-4
M2 parameter on REQUEST 6-19

N parameter
  REQUEST 11-2
  STAGE 11-2
NDFILE control statement 1-16
Nine-track tapes 6-5, 15
No recovery 6-21
NOGO statement 3-13
Noise blocks 6-22
NOREWIN option 3-17; 4-17
NR option on LOAD 4-17
NR parameter on REQUEST 6-21
NR parameter on STAGE 6-21
NT parameter
  Job statement 2-4; 6-15
  REQUEST 6-15
  STAGE 6-15
NUCLEUS library 1-11; 3-19; 4-1
NUM function (refer to SCOPE 2 Reference Manual)

Object module 1-6; 3-10
OF parameter on FILE statement 10-12
On-line tapes
  Accounting information 1-18
  Configuration 1-3
  Multifile 6-18; C-2
  Scheduling 6-14
  Staged 6-6, 20
  Use 6-14
  Used by job 1-19
Operating registers 1-2
OUTPUT file
  Dayfile 1-17
  Default for assemblers/compilers 1-20
  Default for copy routines 10-1
  Default for map 12-17
  Disposition of 9-15
  Format 9-10
  Introduction 1-20
  I/O buffer 4-1
  Printer control 9-10
  Return 7-18
  Unblocked required 9-10
  W records required 9-10
  Writing on 9-10

P parameter on job statement 2-5
P parameter on LABEL 11-3, 4
Parameters
  Keyword 2-7
  Positional 2-7
  Rules 2-6
Parenthesis, left as separator 2-6
Parenthesis, right as terminator 2-6
Partition
  Comparing 10-13
  Copying 10-4
  File structure 5-19
  Punched 9-19
  Separator card 2-9; 9-19
Passwords 8-8 through 13
PAUSE statement 4-12
PB on DISPOSE 9-21
PD parameter on FILE statement 5-27
PE parameter
  REQUEST 6-6, 15
  STAGE 6-6
Period as terminator
  Control statement 2-6
  Job statement 2-1
Peripheral processor unit 1-4
Permanent file
  Altering 8-16
  Attaching 8-4, 16
  Cataloging 8-3, 8
  Cycles 8-1
  Extending 8-16
  GETPF 8-5, 20
  Manager 1-7
  Modifying 8-10
  Name 8-1
  Passwords 8-8
  Permissions 8-1, 3, 8
  Purging 8-18
  Retention period 8-7
  Return 7-18
  SAVEPF 8-3, 20
  SCOPE 2 8-1
  SCOPE 3.4 8-20
POST on STAGE 6-4
PPU 1-4
PR on DISPOSE 9-14
PRE on STAGE 6-4
PRESET option 3-17; 4-10
Printer output 9-10
Priority, job 2-5
Privacy procedures 8-20
.PROC statement 13-16
Procedures 13-16
  Body 13-17
  Call 13-18
  Header statement (.PROC) 13-16
  Return 13-18
Procedure Commands 13-28
  .* 13-32
  .DATA 13-28
  .EOF 13-31
  .EOR 13-31
Processing direction 5-27
Processor code 2-4
Product set, SCOPE 2 1-5
Program image module
  Introduction 1-5
  Loading 3-18
  NOGO generation 3-14
  NUCLEUS library 3-19
PU on DISPOSE 9-12
Publications, CDC iv
PUNCH file
  Disposition 9-20
  Format 9-20
  Introduction 1-21
  Return 7-18
  Unblocked required 9-20
  W records required 9-20
PUNCHB file
  Disposition 9-20
  Format 9-20
  Introduction 1-21
  Return 7-18
  Unblocked required 9-20
  W records required 9-20
Punched cards
  ASCII input 9-3
  Coded input 9-3
  Coded output 9-20
  Disposition 9-21
  EOI 2-9; 9-19
  EOP 2-9; 9-19
  EOS 2-8; 9-19
  Forms control 9-22
  Free-form binary input 9-8
  Free-form binary output 9-21
  Hollerith input 9-3
  Hollerith output 9-20, 21
  INPUT 3-7
  Input 9-3, 7, 8
  Lace card 9-19
  Mispunched 9-19
  Output 9-19
  SCOPE binary input 9-7
  SCOPE binary output 9-20
  Separator cards 9-19
PURGE statement 8-18
PW parameter 8-12
P1 on DISPOSE statement 9-15
P2 on DISPOSE statement 9-15
P8 on DISPOSE statement 9-21

R option on LOAD 4-17
R parameter on job statement 2-3; 4-15
R parameter on LABEL 11-2
R record type 5-6; D-10; E-8
RB parameter on FILE statement 5-17
RD password/permission 8-8
Read-ahead 6-19
Read password/permission 8-8
Record
  Logical 5-2; D-1
  Physical (refer to Blocking)
RECORD CONTAINS clause 5-9
Record manager 17-; 5-1; D-1; E-1
Record mark character 5-6; D-10; E-8
Record mark records 5-6; D-10; E-8
Record type
  D Decimal count 5-6; D-8; E-7
  F Fixed length 5-5; D-7; E-6
  R Record mark 5-6; D-10; E-8
  S SCOPE logical 5-4; D-1; E-4
  Specification 5-2
  T Trailer 5-6; D-12; E-9
  U Undefined 5-7; D-14; E-10
  W Word control 5-3; D-1; E-2
  Z Zero byte 5-5; D-4; E-5
Record type specification 5-2
Recovery
  Magnetic tape 6-21
REDUCE statement
  In load sequence 3-14
  Use 4-8
Relocatable modules 1-6; 3-10
Removable sets 7-3, 6, 10; 8-16
Rerun job 4-15
REQUEST statement
  Mass storage 7-1
  On-line tape 6-14
Retention period 8-7
  Permanent file 8-7
  Set member 7-8
RETURN statement
  Disposed file 7-18; 9-14
  Magnetic tape 6-21
  Mass storage 7-18
  On-line tape 6-21
  Permanent file 7-18
  Staged file 7-18
  System file 7-18
REVERT statement 13-22
Rewind before load 3-4; 4-17
Rewind of load file 4-17
REWIND option on LDSET 3-17; 4-17
REWIND statement 4-17; 10-11
RFL statement 4-7, 8
RMK parameter on FILE statement 5-6; D-11; E-9
RP parameter on CATALOG 8-7
RT parameter on FILE statement 5-2; D-1; E-1
RUN compiler
  Introduction 1-9
  Object-time file names 1-20; 3-4
  Statement 3-1
RUNLIB library 3-20

S parameter on REDUCE 4-8
S record type 5-4, 14; D-1, E-4
Satisfying of externals 3-10, 14, 19
SAVEPF
  SCOPE 2 8-3
  SCOPE 3.4 8-20
SCM
  Maximum allowed 4-3, 5
  Minimum required 4-5
  Used by job 1-18; 4-1
  User control 4-5
SCOPE logical records 5-4, 14; D-1; E-4
SCOPE 2
  Features 1-1
  Introduction 1-1
  Meaning of acronym 1-1
  Operating environment 1-2
  Product set 1-5
Scratch file
  Creation 7-4
  Return 7-18
Section
  Comparing 10-13
  Control statement 2-6
  Copying 10-3
  File structure 5-21
  Punched 9-19
  Separator card 2-8
SEGLOAD 1-6
Segment loader 1-6
SEGRES 1-6
Selective load 3-16, 23
Separator cards
  End-of-information 2-9; 9-19
  End-of-partition 2-9; 5-19; 9-19
  End-of-section 2-8; 5-21; 9-19
  Free-form binary 9-8
Separators on control statements 2-6
Sequential file organization 5-26
Sets
  Addition of members 7-8
  Assignment of files 7-3, 10
  Creation 7-6
  Deleting of members 7-11
  Device scheduling 7-6
  Dismounting members 7-11
  Mass storage 7-2
  Member limit 7-8

Mounting members 7-9
Names 7-5
Permanent file devices 7-8
Permanent file limit 7-8
Removable 7-2,3,5
Request use of 7-5,10
Retention period 7-8
System 7-2,3,5
Setname 7-5
SETNAME control statement 7-7
SET statement 13-5
SF parameter on STAGE 6-13
SIMI5 statement 3-1
SIMSCRIPT 1-10
Single volume file C-1
SKIP statement 13-12
SKIPB statement 10-10
SKIPF statement 10-7
Slant bar as separator 2-7
SLOAD loader statement 3-16,23
Small central memory (refer to SCM)
SM parameter 2-4; 6-3
SN parameter 2-4; 6-3
SP parameter 2-4
Software configuration 1-5
Sort/merge program 1-9
Spanning of blocks 5-16
Spooled files 9-1
SPR on FILE statement 6-19
ST parameter
GETPF 8-20
SAVEPF 8-20
DISPOSE 9-15
On-line tape stage 6-6
PAUSE 4-12
STAGE 6-6
STAGE 6-3
Stage by blocks 6-12
Stage by file 6-13
Stage by tapemark 6-12
Stage by volume 6-10
Stage file return 7-18
Staging 6-3
Station
General description 1-4
Maintenance 1-4
On day file 1-18
6000 or CYBER 1-11
7611-1 I/O station 1-12
Substitution of keyword parameter 13-25
SUMMARY statement 12-21
Swap count 1-19
Swap space in LCM 4-2
SWITCH statement 4-13
Switch, program 4-13
Switch, pseudo sense 4-13
System controlled mode 4-3
System mass storage 1-4; 7-3
System set 7-3,5
System verb table 3-8

T on job statement 2-3
T parameter on LABEL 11-3,6
T record type 5-6; D-12; E-9
T transfer unit parameter 7-14
Tape unit 6-5
Tape density
REQUEST 6-6,15
STAGE 6-6
Termination on error 5-28
Time limit for job 2-3,4
Time used by job 1-17
TK password/permission 8-11
TL parameter on FILE statement 5-6; D-13; E-10
Trailer count records 5-6; D-12; E-9
TRANSF statement 4-13
Transfer unit size 7-14
TRAP statement 12-1
Turnkey password/permission 8-11

U parameter on LABEL 11-3,6
U record type 5-7; D-14; E-10
Unblocked files 5-12
Undefined records 5-7; D-14; E-10
UNLOAD statement
Disposed file 7-18
Magnetic tape 6-21
Mass storage 7-18
On-line tape 6-21
Permanent file 7-18
Staged file 7-18
System file 7-18
UPDATE program 1-11
US parameter
REQUEST 6-7; 11-8
STAGE 6-7; 11-8
USR on dayfile 1-17
User, identified iii

V parameter on LABEL 11-3,4
Verb, control statement 3-8
VF parameter on FILE 6-17
Volume header label C-4
Volume/header label group C-1
VOL1 label C-4
VSN parameter
REQUEST 6-18; 7-5,10
STAGE 6-10,14

W parameter on LABEL 11-2
W record type 5-3, 16; 9-1; D-1; E-2
WCK parameter on REQUEST 7-16
WHILE statement 13-15
Word addressable organization 5-26
Write behind 6-19

XR password 8-12

YD parameter on job statement 2-4; 7-6
YL parameter on job statement 2-4; 7-6

Z record type 5-5; D-4; E-5
Zero byte 5-5
Zero-length record
    S record 5-4
    W record 5-3
ZZZZZEF file 5-30
ZZZZZxx files 3-8

COMMENT SHEET

MANUAL TITLE CDC SCOPE 2.1 User's Guide

PUBLICATION NO. 60372600 REVISION E

FROM: NAME:

BUSINESS ADDRESS:

CUT ALONG LINE

AA3419 REV. 7/75 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD

FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION

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 or HD, PE)	———	200, 556, 800, or 1600 bpi
Tape label density	———	200, 556, 800, or 1600 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)
Mainframe identifier (xxx)	———	
Station identifier (ggg)	———	
Terminal identifier (ttt)		