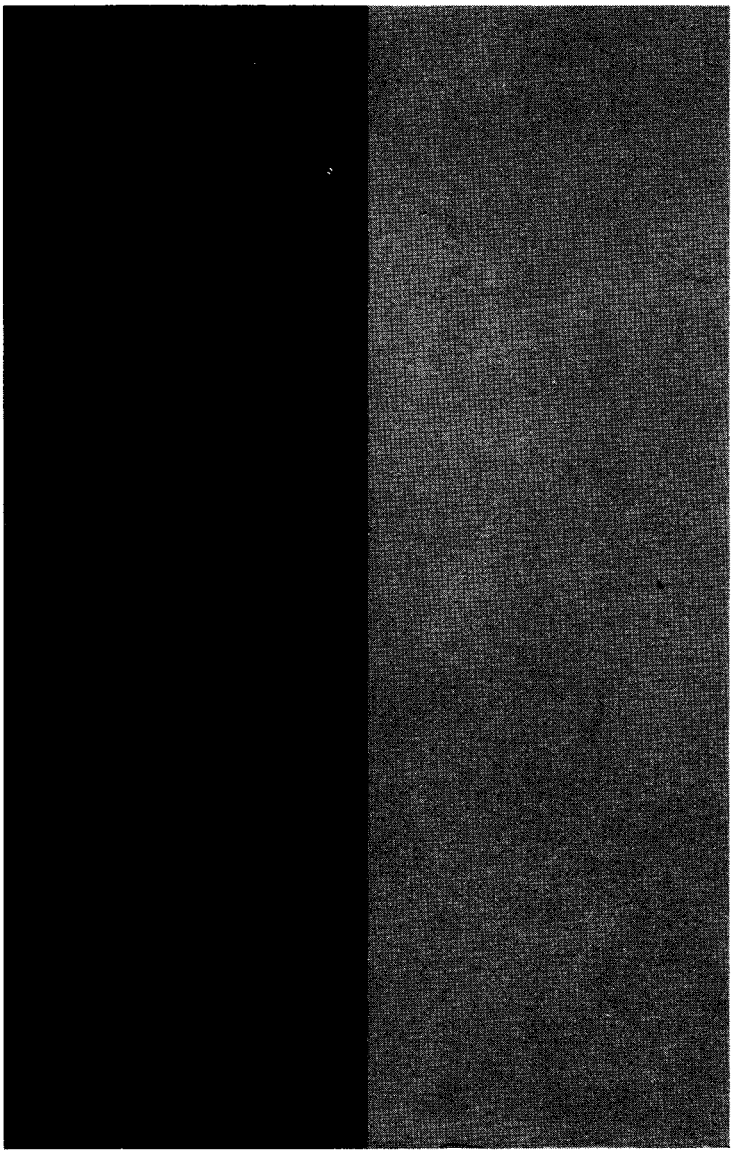
SPERRY RAND

UNIVAC 9400 SYSTEM

COBOL



SUPPLEMENTARY REFERENCE

UP-7709
Rev. 2

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

UNIVAC is a registered trademark of the Sperry Rand Corporation.

Other trademarks of the Sperry Rand Corporation in this publication are:

UNISERVO

PRINTED IN U.S.A.

UNIVAC 9400 System
COBOL Supplementary Reference
UP-7709 Rev. 2

February 1973

UPDATING PACKAGE F

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover & Disclaimer	†	†
Page Status Summary	N. A.	PSS-1 and PSS-2
Contents	1 Rev. 1 and 2 Rev. 1 7 Rev. 1 and 8 Rev. 1	1 Rev. 2 and 2 Rev. 1 7 Rev. 1 and 8 Rev. 2
Appendix P	N. A.	1** thru 18**
Index	1 Rev. 1 and 2 Rev. 1 3 Rev. 1 thru 8 Rev. 1 9 Rev. 1 and 10 Rev. 2 11 Rev. 2 and 12 Rev. 1 13 Rev. 1 and 14 Rev. 1	1 Rev. 1 and 2 Rev. 2 3 Rev. 2 thru 8 Rev. 2 9 Rev. 2 and 10 Rev. 3 11 Rev. 3 and 12 Rev. 2 13 Rev. 2 and 14 Rev. 2

† Destroy old cover and file new cover

** These are new pages

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

UPDATING PACKAGE E

File pages as specified below.

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Section 2	5 Rev. 1 & 6 Rev. 2	5 Rev. 1 & 6 Rev. 3
Section 3	1 Rev. 1 & 2	1 Rev. 1 & 2 Rev. 1
Section 4	3 Rev. 1 & 4 Rev. 1	3 Rev. 2 & 4 Rev. 1
Section 5	9 Rev. 1 & 10 Rev. 1 11 Rev. 2 & 12 Rev. 2 12a & 12b 13 Rev. 1 & 14 15 & 16 21 Rev. 1 & 22	9 Rev. 1 & 10 Rev. 2 11 Rev. 2 & 12 Rev. 3 12a Rev. 1 & 12b 13 Rev. 1 & 14 Rev. 1 15 Rev. 1 & 16 Rev. 1 21 Rev. 2 & 22
Section 6	15 Rev. 1& 16 Rev. 1 17 Rev. 1 & 18 Rev. 2 21 Rev. 1 & 22 Rev. 2 25 & 26 Rev. 1 47 Rev. 2 & 48 Rev. 2	15 Rev. 2 & 16 Rev. 1 17 Rev. 2 & 18 Rev. 2 21 Rev. 2 & 22 Rev. 2 25 & 26 Rev. 2 47 Rev. 3 & 48 Rev. 3
Section 10	1 Rev. 1 & 2 Rev. 1	1 Rev. 2 & 2 Rev. 1
Appendix D	13 & 14 17 & 18 Rev. 1	13 Rev. 1 & 14 17 & 18 Rev. 2
Appendix G	1 & 2 Rev. 1	1 & 2 Rev. 2
Appendix I	5 & 6 Rev. 1 7 Rev. 1 & 8 Rev. 1 9 & 10	5 & 6 Rev. 2 7 Rev. 2 & 8 Rev. 2 9 & 10 Rev. 1
Appendix J	45 & 46 Rev. 1	45 & 46 Rev. 2
Appendix K	3 Rev. 1 & 4 5 Rev. 2 & 6 Rev. 2	3 Rev. 2 & 4 5 Rev. 3 & 6 Rev. 3
Appendix L	3 & 4	3 Rev. 1 & 4 Rev. 1

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

UPDATING PACKAGE D

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover & Disclaimer	†	†
Contents	3 Rev. 2 and 4 Rev. 2	3 Rev. 3 and 4 Rev. 3
Section 1	1 and 2 Rev. 3	1 and 2 Rev. 4
Section 2	5 and 6 Rev. 1 9	5 Rev. 1 and 6 Rev. 2 9 Rev. 1
Section 4	1 Rev. 1 and 2 11 Rev. 2 and 12 Rev. 2	1 Rev. 2 and 2 Rev. 1 11 Rev. 3 and 12 Rev. 2
Section 5	3 Rev. 1 and 4 Rev. 2 9 Rev. 1 and 10 N. A.	3 Rev. 2 and 4 Rev. 2 9 Rev. 1 and 10 Rev. 1 10a**
Section 6	5 and 6 Rev. 1 9 Rev. 1 and 10 11 and 12 Rev. 1 13 and 14 Rev. 1 N. A. 35 and 36 46a and 46b	5 and 6 Rev. 2 9 Rev. 2 and 10 11 and 12 Rev. 2 13 and 14 Rev. 2 14a** 35 and 36 Rev. 1 46a Rev. 1 and 46b Rev. 1
Section 7	1 and 2 N. A.	1 Rev. 1 and 2 Rev. 1 3**
Section 9	1 and 2 Rev. 1 3 and 4	1 and 2 Rev. 2 3 Rev. 1 and 4
Section 10	1 and 2	1 Rev. 1 and 2 Rev. 1
Appendix B	1 Rev. 1 and 2 Rev. 1	1 Rev. 1 and 2 Rev. 2
Appendix E	1 Rev. 2	1 Rev. 3
Appendix J	13 and 14 15 Rev. 1 and 16 23 Rev. 1 and 24 37 and 38 43 and 44	13 and 14 Rev. 1 15 Rev. 2 and 16 23 Rev. 1 and 24 Rev. 1 37 and 38 Rev. 1 43 and 44 Rev. 1
Appendix K	1 Rev. 1 and 2	1 Rev. 1 and 2 Rev. 1
Appendix M	3 Rev. 1 and 4	3 Rev. 2 and 4
Index	9 Rev. 1 and 10 Rev. 1 11 Rev. 1 and 12 Rev. 1	9 Rev. 1 and 10 Rev. 2 11 Rev. 2 and 12 Rev. 1

†Destroy old cover and file new cover.

**These are new pages.

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a sentence indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a sentence indicates a technical change in only that sentence. A horizontal arrow located between two consecutive sentences indicates technical changes in both sentences or deletions.

December 10, 1971

UPDATING PACKAGE "C"

File Pages as Specified Below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Contents	1 and 2	1 Rev. 1 and 2 Rev. 1
	3 Rev. 1 and 4 Rev. 1	3 Rev. 2 and 4 Rev. 2
	5 and 6 Rev. 2	5 and 6 Rev. 3
	7 and 8	7 Rev. 1 and 8 Rev. 1
	N. A.	9*
Section 1	1 and 2 Rev. 2	1 and 2 Rev. 3
	3 Rev. 1	3 Rev. 2
Section 4	7 Rev. 1 and 8	7 Rev. 1 and 8 Rev. 1
	11 Rev. 1 and 12 Rev. 1	11 Rev. 2 and 12 Rev. 2
	13 Rev. 1	13 Rev. 2
Section 5	3 and 4 Rev. 1	3 Rev. 1 and 4 Rev. 2
	5 Rev. 1 and 6	5 Rev. 2 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8 Rev. 1
	9 and 10	9 Rev. 1 and 10
	11 Rev. 1 and 12 Rev. 1	11 Rev. 2 and 12 Rev. 2
	N. A.	12a* and 12b*
	21 and 22	21 Rev. 1 and 22
	23 and 24 Rev. 1	23 and 24 Rev. 2
	25	25 Rev. 1 and 26*
Section 6	3 and 4 Rev. 1	3 Rev. 1 and 4 Rev. 1
	13 and 14	13 and 14 Rev. 1
	17 and 18 Rev. 1	17 Rev. 1 and 18 Rev. 2
	21 and 22 Rev. 1	21 Rev. 1 and 22 Rev. 2
	39 Rev. 1 and 40 Rev. 1	39 Rev. 2 and 40 Rev. 2
	N. A.	40a* thru 40d*
	41 and 42 Rev. 1	41 and 42 Rev. 2
	43 and 44 Rev. 2	43 and 44 Rev. 3
	45 Rev. 1 and 46 Rev. 1	45 Rev. 2 and 46 Rev. 2
	N. A.	46a* and 46b*
	47 Rev. 1 and 48 Rev. 1	47 Rev. 2 and 48 Rev. 2
	49	49 Rev. 1
Section 8	3	3 Rev. 1
Section 9	1 and 2	1 and 2 Rev. 1
Appendix B	1 and 2	1 Rev. 1 and 2 Rev. 1
Appendix E	1 Rev. 1	1 Rev. 2

*These are new pages

UPDATING PACKAGE "C"

File Pages as Specified Below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Appendix F	1 and 2	1 Rev. 1 and 2 Rev. 1
	5 and 6	5 Rev. 1 and 6 Rev. 1
Appendix I	3 and 4	3 Rev. 1 and 4
	5 and 6	5 and 6 Rev. 1
	13 and 14	13 and 14 Rev. 1
Appendix J	9 and 10 Rev. 1	9 and 10 Rev. 2
	11 Rev. 1 and 12 Rev. 1	11 Rev. 2 and 12 Rev. 1
	17 and 18	17 and 18 Rev. 1
	19 and 20 Rev. 1	19 Rev. 1 and 20 Rev. 2
	21 and 22	21 Rev. 1 and 22 Rev. 1
	23 and 24	23 Rev. 1 and 24
	25 and 26	25 and 26 Rev. 1
	27 and 28	27 Rev. 1 and 28
	29 Rev. 1 and 30	29 Rev. 2 and 30
	39 and 40	39 and 40 Rev. 1
	41 and 42	41 Rev. 1 and 42
Appendix K	5 Rev. 1 and 6 Rev. 1	5 Rev. 2 and 6 Rev. 2
Appendix L	1 and 2	1 and 2 Rev. 1
	5 Rev. 1	5 Rev. 2
Appendix N	N. A.	1* thru 8*
Appendix O	N. A.	1* and 2*
Index	1 and 2	1 Rev. 1 and 2 Rev. 1
	3 and 4	3 Rev. 1 and 4 Rev. 1
	5 and 6	5 Rev. 1 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8 Rev. 1
	9 and 10	9 Rev. 1 and 10 Rev. 1
	11 and 12	11 Rev. 1 and 12 Rev. 1
	13 and 14	13 Rev. 1 and 14 Rev. 1

*These are new pages

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a sentence indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a sentence indicates a technical change in only that sentence. A horizontal arrow located between two consecutive sentences indicates technical changes in both sentences or deletions.

October 10,1971

UPDATING PACKAGE "B"

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Contents	3 and 4	3 Rev. 1 and 4 Rev. 1
	5 and 6 Rev. 1	5 and 6 Rev. 2
Section 1	1 and 2 Rev. 1	1 and 2 Rev. 2
	3	3 Rev. 1
Section 2	3 and 4	3 and 4 Rev. 1
	5 and 6	5 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8
Section 3	1 and 2	1 Rev. 1 and 2
Section 4	1 and 2	1 Rev. 1 and 2
	3 Rev. 1 and 4	3 Rev. 1 and 4 Rev. 1
	7 and 8	7 Rev. 1 and 8
	9 Rev. 1 and 10	9 Rev. 2 and 10 Rev. 1
	11 and 12	11 Rev. 1 and 12 Rev. 1
	13	13 Rev. 1
Section 5	3 and 4	3 and 4 Rev. 1
	5 and 6	5 Rev. 1 and 6
	11 and 12	11 Rev. 1 and 12 Rev. 1
	13 and 14	13 Rev. 1 and 14
	23 and 24	23 and 24 Rev. 1
Section 6	3 and 4	3 and 4 Rev. 1
	5 and 6	5 and 6 Rev. 1
	7 and 8	7 and 8 Rev. 1
	9 and 10	9 Rev. 1 and 10
	11 and 12	11 and 12 Rev. 1
	15 and 16	15 Rev. 1 and 16 Rev. 1
	17 and 18	17 and 18 Rev. 1
	19 Rev. 1 and 20	19 Rev. 2 and 20 Rev. 1
	21 and 22	21 and 22 Rev. 1
	27 and 28	27 Rev. 1 and 28 Rev. 1
	29 and 30	29 and 30 Rev. 1
	31 and 32	31 Rev. 1 and 32 Rev. 1
	37 and 38	37 Rev. 1 and 38 Rev. 1
	39 and 40	39 Rev. 1 and 40 Rev. 1
	41 and 42	41 and 42 Rev. 1
	43 and 44 Rev. 1	43 and 44 Rev. 2
	45 and 46	45 Rev. 1 and 46 Rev. 1
	47 and 48	47 Rev. 1 and 48 Rev. 1

(Continued)

UPDATING PACKAGE "B" CONTINUED

File pages as specified below

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Section 9	5 and 6	5 Rev. 1
Section 10	1	1 and 2**
Appendix D	1 and 2 3 and 4 17 and 18	1 Rev. 1 and 2 Rev. 1 3 and 4 Rev. 1 17 and 18 Rev. 1
Appendix E	1	1 Rev. 1
Appendix G	1 and 2	1 and 2 Rev. 1
Appendix I	7 and 8	7 Rev. 1 and 8 Rev. 1
Appendix J	9 and 10 11 and 12 15 and 16 19 and 20	9 and 10 Rev. 1 11 Rev. 1 and 12 Rev. 1 15 Rev. 1 and 16 19 and 20 Rev. 1
Appendix K	5 and 6	5 and 6 Rev. 1
Appendix L	5	5 Rev. 1
Appendix M	1 and 2	1 Rev. 1 and 2 Rev. 1

**This is a new page.

All the technical changes in an update are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a sentence indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a sentence indicates a technical change in only that sentence. A horizontal arrow located between two consecutive sentences indicates technical changes in both sentences.

July 6, 1971

UPDATING SUMMARY SHEET "A"

This announces the release and availability of Updating Package "A" to UP-7709 Rev. 2, "UNIVAC 9400 System COBOL Supplementary Reference."

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Front Cover & Disclaimer	†	†
Contents	5 and 6	5* and 6 Rev. 1
Section 1	1 and 2	1* and 2 Rev. 1
Section 4	3 and 4 9 and 10	3 Rev. 1 and 4* 9 Rev. 1 and 10*
Section 6	19 and 20 25 and 26 43 and 44	19 Rev. 1 and 20* 25* and 26 Rev. 1 43* and 44 Rev. 1
Appendix D	7 and 8 15 and 16	7 Rev. 1 and 8* 15* and 16 Rev.1
Appendix J	1 and 2 29 and 30 45 and 46	1 Rev. 1 and 2* 29 Rev. 1 and 30* 45* and 46 Rev. 1
Appendix K	1 and 2 3 and 4	1 Rev. 1 and 2* 3 Rev. 1 and 4*
Appendix M	1 and 2 3 and 4	1* and 2 Rev. 1 3 Rev. 1 and 4*

†Destroy old cover and file new cover.
*These pages are backups of revised pages and remain unchanged.

PAGE STATUS SUMMARY

ISSUE: Update F to UP-7709 Rev. 2

Section	Page Number	Update Level
Cover/Disclaimer		F
PSS	1, 2	F
Acknowledgment	1	Orig.
Contents	1	F
	2	C
	3, 4	D
	5	Orig.
	6, 7	C
	8	F
	9	C
1	1	Orig.
	2	D
	3	C
2	1 thru 3	Orig.
	4	B
	5	D
	6	E
	7	B
	8	Orig.
	9	D
3	1	B
	2	E
4	1, 2	D
	3	E
	4	B
	5, 6	Orig.
	7	B
	8	C
	9, 10	B
	11	D
	12, 13	C
5	1, 2	Orig.
	3	D
	4 thru 9	C
	10	E
	10a	D
	11	C
	12, 12a	E
	12b	C
	13	B
	14 thru 16	E
	17 thru 20	Orig.
	21	E
	22, 23	Orig.
	24 thru 26	C
6	1, 2	Orig.
	3	C
	4	B
	5	Orig.
	6	D
	7	Orig.
	8	B
	9	D
	10, 11	Orig.
	12	D
	13	Orig.
	14, 14a	D
	15	E
	16	B
	17	E
	18	C
	19, 20	B
	21	E
	22	C
	23 thru 25	Orig.
	26	E
	27, 28	B
	29	Orig.
	30 thru 32	B
	33 thru 35	Orig.
	36	D
	37, 38	B
	39, 40	C
	40a thru 40d	C
	41	Orig.
	42	C
	43	Orig.
	44 thru 46	C
	46a, 46b	C
	47, 48	E
	49	C
7	1 thru 3	D
8	1, 2	Orig.
	3	C
9	1	Orig.
	2, 3	D
	4	Orig.
	5	B
10	1	E
	2	D
Appendix A	1	Orig.
	2	B
Appendix B	1	C
	2	D
Appendix C	1 thru 3	Orig.
Appendix D	1, 2	B
	3	Orig.
	4	B
	5, 6	Orig.
	7	A
	8 thru 12	Orig.
	13	E
	14, 15	Orig.
	16	A
	17	Orig.
	18	E
Appendix E	1	D
Appendix F	1, 2	C
	3, 4	Orig.
	5, 6	C
Appendix G	1	Orig.
	2	E
	3	Orig.
Appendix H	1 thru 3	Orig.
Appendix I	1, 2	Orig.
	3	C
	4, 5	Orig.
	6 thru 8	E
	9	Orig.
	10	E
	11 thru 13	Orig.
	14	C
Appendix J	1	A
	2 thru 9	Orig.
	10, 11	C
	12	B
	13	Orig.
	14, 15	D
	16, 17	Orig.
	18 thru 23	C
	24	D
	25	Orig.
	26, 27	C
	28	Orig.
	29	C
	30 thru 37	Orig.
	38	D
	39	Orig.
	40, 41	C
	42, 43	Orig.
	44	D
	45	Orig.
	46	E
Appendix K	1	A
	2	D
	3	E
	4	Orig.
	5, 6	E
Appendix L	1	Orig.
	2	C
	3, 4	E

PAGE STATUS SUMMARY

ISSUE: Update F to UP-7709 Rev. 2

Section	Page Number	Update Level
	5	C
Appendix M	1 2 3 4	B A D Orig.
Appendix N	1 thru 8	C
Appendix O	1, 2	C
Appendix P	1 thru 18	F
Index	1 2 thru 14	C F

Section	Page Number	Update Level

Section	Page Number	Update Level

ACKNOWLEDGMENT

This manual is based on *American National Standard COBOL, X3.23 – 1968* developed by the American National Standards Institute (ANSI, formerly USASI). In response to their request the following acknowledgment is reproduced in its entirety:

"Any organization interested in using the COBOL specifications as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC®I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

This complete USA Standard edition of COBOL may not be reproduced without permission of the USA Standards Institute."

CONTENTS

TABLES

1. INTRODUCTION

1.1. SCOPE

This supplementary reference manual describes the usage of COBOL with the UNIVAC 9400 System. It is intended for the experienced COBOL programmer who is already familiar with the basic information contained in the *UNIVAC Fundamentals of COBOL Series, UP-7503* (current version). The information provided in this manual is therefore generally restricted to those phases of COBOL of interest to the UNIVAC 9400 COBOL programmer.

For an understanding of file processing facilities available on the 9400 System, refer to *UNIVAC 9200/II/9300 II/9400 Systems P.I.E #1 8411 Disc File Direct Access Subsystem Concepts, UP-7704.1;* and *UNIVAC 9400 System Data Management System Programmers Reference, UP-7629* (current version).

Appendix D contains an explanation of UNIVAC 9400 System COBOL disc procedures.

1.2. SYMBOLS, RULES, AND NOTATIONS USED IN THIS MANUAL

The various language elements that comprise a COBOL program must be written in formats that adhere to fixed and precise rules of presentation. Each format statement indicates the following information:

- order of presentation;
- words that are requisite to the proper functioning of the statement;
- words that are optional and are included at the discretion of the user;
- information which must be supplied by the user;
- elements in the statement that involve a choice by the user;
- functions of the statement that are optional.

In accordance with the above, the following conventions are used in this manual:

(1) The order of presentation is indicated by the format statement itself.

(2) All words inherent or built into the UNIVAC 9400 COBOL language are printed in upper case (COBOL reserved words). Appendix B lists all UNIVAC 9400 COBOL reserved words.

(3) All uppercase words that are underlined are key words. Key words must be present when the functions of which they are a part are used. Those uppercase words not underlined are optional and may be included at the user's discretion to improve readability (no compiler action). All uppercase words, whether underlined or not, are part of the COBOL language and must be spelled exactly as indicated.

(4) All lowercase words in italics represent generic terms to be supplied by the user when the functions of which they are a part are used.

(5) Elements of a statement involving a choice, one of which must be chosen, are enclosed in braces ({ }). If one of the choices within the braces has no key words, it is a "default option"; i.e., if none of the elements within the braces is specified, the action will be the same as if the default option had been specified.

(6) Optional functions which may be included or omitted at the user's discretion are enclosed in brackets ([ ]).

(7) In some statements, certain portions may be used as many times as needed by the programmer. This repeatability is indicated by the ellipsis (. . .). Brackets or braces are used as delimiters to indicate the portion of the statement which is repeatable.

1.3. UNIVAC 9400 COBOL

Both a basic compiler and an extended compiler have been implemented for UNIVAC 9400 COBOL. Each conforms to the specifications of the American National Standards Institute (ANSI), entitled *American National Standard COBOL, X3.23–1968*. The ANSI modules and levels implemented for each are shown in Table 1–1.

MODULE	LEVEL	
	BASIC	EXTENDED
Nucleus	1	2
Table Handling	2	3
Sequential Access	1	2
Random Access	–	2
Sort	–	2
Segmentation	1	2
Library	1	2

Table 1–1. UNIVAC 9400 COBOL Module/Level Implementation

The minimum system configuration required for each compiler is:

BASIC	EXTENDED
3 work tapes + 1 system tape or disc	3 disc work areas + 1 system disc
1 card reader or substitute device	1 card reader or substitute device
1 printer or substitute device	1 printer or substitute device
1 reader	65,536 byte memory**
32,768 byte memory*	

The basic compiler is a true subset of the extended compiler. All subsequent references to "the compiler" in this manual are applicable to both the basic and extended compilers. Appendix E lists the features found only in the extended compiler.

*Compiler requires 20K

**Compiler requires 40K

Those features of UNIVAC 9400 COBOL which exceed ANSI requirements are noted where they occur. All references in this manual to "ANSI requirements" refer to ANSI X3.28 – 1968.

The compiler and all compiler-produced object programs normally operate on data represented in EBCDIC (Extended Binary Coded Decimal Interchange Code) under control of the UNIVAC 9400 Operating System. ←

A COBOL source program can be entered into the compiler from the card reader or from a tape or disc library file. The compiler produces, as its final output, a relocatable object program on tape (basic COBOL) or disc (extended COBOL). This output must be processed by the linker utility program (described in *UNIVAC 9400 System Linkage Editor Programmers Reference, UP-7703* current version) for production of loadable, executable object code.

1.4. COBOL 9400 COBOL COMPILER OPTIONS

The following compiler options are available to the user and are discussed in Appendix G.

(1) Ambiguous reference checking

(2) Source listing

(3) Object program listing

(4) Data Division map

(5) Procedure Division map

(6) Cross-reference listing

(7) Suppress precautionary diagnostics

(8) Inhibit source sequence number check

(9) Inhibit transfer address generation in object module (subprogram)

(10) Inhibit generation of all Linker control cards in object modules produced by compiler

(11) Inhibit generation of object modules

(12) Specify source program input from tape/disc library file

(13) Specify COPY input from tape/disc library file

(14) Single space source listing

(15) Suppress printer mismatch during compilation or object program execution.

2. GENERAL SPECIFICATIONS

2.1. UNIVAC 9400 COBOL CHARACTER SET

The UNIVAC 9400 COBOL character set is a 52 character subset of the UNIVAC 9400 System character set which contains 256 characters.

The COBOL character set consists of the following characters:

0,1,...,9

A,B,...,Z

Blank or space (written on coding form as ␣, or a blank space)

. Period

< Less than

(Left parenthesis

+ Plus sign

$ Currency sign

* Asterisk (if used in column 7 indicates that the entire source statement is commentary)

) Right parenthesis

; Semi-colon

– Minus sign or hyphen

, Comma

. > Greater than

' Apostrophe (alternate character for quotation mark)

= Equal sign

'' Quotation mark (see apostrophe)

/ Slash

The collation sequence for these characters is given in Appendix A.

The UNIVAC 9400 COBOL character set may be used anywhere in a program; however, the additional characters, which together with the COBOL set make up the system set, may be used only in the following instances:

- Anywhere in the Identification Division except in the Program-ID clause.
- In the NOTE statement of the Procedure Division.
- Nonnumeric literals.

The apostrophe is the alternate character for the quotation mark in UNIVAC 9400 COBOL, and either one or both may be used to designate nonnumeric literals. Any UNIVAC 9400 computer character except the apostrophe and the quotation mark may be used within a nonnumeric literal.

The following paragraphs describe the general usage of the various UNIVAC 9400 COBOL characters.

2.1.1. Characters Used for Words

A COBOL word is a sequence of not more than 30 of the following characters:

0,1,...,9

A,B,...,Z

– (hyphen)

A word may neither begin nor end with a hyphen nor contain a space.

2.1.2. Characters Used for Punctuation

COBOL punctuation characters are:

' Apostrophe (alternate character for quotation mark)

(Left parenthesis

) Right parenthesis

Blank or space (written on coding form as ␣, or a blank space)

. Period

, Comma

; Semicolon

" Quotation mark (see apostrophe)

2.1.3. Characters Used in Relational Expressions

The COBOL characters used to represent relational operators are:

= Equals

> Greater than

< Less than

2.1.4. Characters Used in Arithmetic Operations

The characters used in arithmetic operations are:

+ Plus sign (addition)

– Minus sign (subtraction)

* Asterisk (multiplication)

/ Slash (division)

** Two asterisks (exponentiation)

2.1.5. Characters Used in Editing

The characters used in editing consist of the following:

B Blank or space insertion

0 Zero insertion

+ Plus sign

– Minus sign

CR Credit

DB Debit

Z Zero suppression

* Check protection

$ Currency symbol

, Comma

. Decimal point

2.2. TYPES OF WORDS

There are two types of words in UNIVAC 9400 COBOL:

- User-Supplied and
- Reserved.

These are described in the following paragraphs.

2.2.1. User-Supplied Words

Certain words in COBOL fall into the general classification of user-supplied words. These are listed and defined in Table 2–1.

2.2.2. Reserved Words

Reserved words are used for syntactical purposes and may not appear as user-defined words. The various types of reserved words are described in Table 2–2. Appendix B contains a complete list of UNIVAC 9400 COBOL reserved words.

USER-SUPPLIED WORD	RULES
Data-name	(1) 1 to 30 characters. (2) Permissible characters are 0 through 9, A through Z, and hyphen (-). (3) Must include at least one alphabetic character. (4) Hyphen (-) cannot be the first or last character. (5) May be qualified; may not be subscripted.
Unqualified Data-name	(1) Rules 1 through 4 for data-name. (2) May not be qualified; may not be subscripted.
Identifier	(1) Rules 1 through 4 for data-name. (2) May be qualified and/or subscripted.
Condition-name	(1) Rules 1 through 4 for data-name. (2) Value may be established in a level-88 entry or in a SPECIAL-NAMES switch status declaration. (3) Referenced only in conditions.
Conditional Variable	(1) Rules 1 through 4 for data-name. (2) Data-name immediately followed by one or more associated level-number 88 entries.
Procedure-name	(1) Rules 1, 2, and 4 for data-name. (2) Must precede each referenced paragraph. (3) A procedure-name is a section-name if it is followed by the word SECTION. (4) May be composed solely of numeric characters.
External-name	(1) A nonnumeric literal of 1 to 8 characters. (2) A user-supplied label which duplicates the LFD name used in the job control stream to name COBOL file.

Table 2-1. Types of User-Supplied Words (Part 1 of 2)

USER-SUPPLIED WORD	RULES
Index-name	(1) Rules 1 through 4 for data-name. (2) Value of index-name corresponds to an occurrence number for a table dimension. (3) Initialized and modified only by the SET statement. (4) Defined by the INDEXED BY clause. (5) Table references using indexing are specified by the data-name of the table element followed by parentheses including an index-name for each table dimension. (6) Storage areas assigned by compiler.
Index Data Item	(1) Rules 1 through 3 for index-name. (2) Defined by USAGE IS INDEX clause. (3) May be part of a group referred to in a MOVE or I-O statement.
Numeric Literal	(1) A string of not more than 20 characters including 0 through 9, sign (+ or −), and decimal point. (2) Must contain at least one and not more than 18 digits plus a sign and a decimal point. (3) May contain only one sign which must be leftmost character. If unsigned, literal is positive. (4) May contain only one decimal point, which is treated as an assumed decimal point. If no decimal point, the literal is an integer. (5) Decimal point cannot be the last character in a numeric literal. (6) When a literal is restricted to numeric, the only figurative constant permitted is ZERO.
Nonnumeric Literal	(1) A string of any of the UNIVAC 9400 System character set excluding quotation marks and the apostrophe. Reserved words may be used. (2) Must contain at least one and not more than 132 characters. (3) Must be enclosed within quotation marks or apostrophes. (4) Any spaces enclosed in the quotation marks are part of the literal and, therefore, are part of the value. (5) All nonnumeric literals are in the alphanumeric category. (6) A figurative constant can be used wherever a nonnumeric literal appears in the format.

Table 2–1. Types of User-Supplied Words (Part 2 of 2)

RESERVED WORD	RULES
Verbs	(1) Denote actions performed by the object program or the COBOL compiler.
Key Words	(1) A word which must be present in a particular clause. (2) Key words are indicated by underlining where they appear in the general formats.
Optional Words	(1) Used in COBOL to improve readability. (2) Presence or absence does not alter handling of statement during compilation or execution of program. (3) Not underlined when shown in generalized format.
TALLY	(1) TALLY is the name of a special register designated by the compiler whose implicit description is that of a COMPUTATIONAL integer of five digits without an operational sign. (2) TALLY holds the count produced by the EXAMINE statement. (3) TALLY may also be used in the PROCEDURE DIVISION as a data-name wherever an elementary data item of integral value may appear.
Figurative Constants	(1) ZERO, ZEROS or ZEROES generate one or more 0's. (2) SPACE or SPACES generate one or more spaces. (3) HIGH-VALUE or HIGH-VALUES generate one or more hexadecimal FF characters (all 1's). This character has the highest value in the 9400 collating sequence. (4) LOW-VALUE or LOW-VALUES generate one or more hexadecimal 00 characters (all 0's). This character has the lowest value in the 9400 collating sequence. (5) QUOTE or QUOTES generate one or more apostrophes ('), hexadecimal code 7D. QUOTE(S) cannot be used in place of quotation marks (") or an apostrophe to bound a non-numeric literal. (6) The ALL literal generates one or more of the literals following the ALL. The literal must be either a non-numeric literal or a figurative constant other than the word ALL. When a figurative constant is used, the word ALL is redundant and is used for readability only. The ALL literal may not be used with DISPLAY, EXAMINE, STOP, or COPY.
Connectives	(1) The qualifier connectives OF and IN are used to associate a data-name or paragraph-name with its qualifier. (2) The logical connectives AND, OR, OR NOT, and AND NOT are used to form compound conditions (extended compiler only). (3) A series connective is the comma which links two or more consecutive operands or statements. The use of a series connective is optional.

Table 2–2. Types of Reserved Words

2.3. QUALIFICATION

Every name used in a COBOL source program must be unique either because of different spelling or because of qualification.

Definition:

Qualification is a means of making a name within a hierarchy unique by appending a prepositional phrase containing the name of a higher level of the hierarchy. It is accomplished by appending one or more phrases composed of a qualifier preceded by IN or OF to a data-name or paragraph-name. IN and OF are logically equivalent.

Rules:

(1) The name associated with the highest level entry in an hierarchy is the highest level qualifier available for a data-name within that hierarchy.

(2) Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

(3) The same name may not appear at two different levels in the same hierarchy.

(4) If a data-name or condition-name is assigned to more than one item, it must be qualified each time it is referenced.

(5) A data-name cannot be subscripted when it is being used as a qualifier.

(6) A paragraph-name must not be duplicated within a section.

(7) Only a section-name can qualify a paragraph-name; the word SECTION must not appear.

(8) A paragraph-name need not be qualified when referred to from within the same section.

(9) A name may be qualified even though it does not require qualification.

(10) Level indicator names and section-names must be unique in themselves as they cannot be qualified.

(11) A data name being qualified may be subscripted or indexed. The subscripts/indices must appear to the right of the last qualifier name.

Example:

data-name-1 [{ OF / IN } *data-name-2*] . . . [(*sub-1* [, *sub-2* [, *sub-3*]])]

2.4. SUBSCRIPTING AND INDEXING

Definition:

Subscripting and indexing are techniques used to refer to individual table elements within a table of like elements that have not been assigned individual data-names.

Rules:

(1) Up to three levels of subscripting or indexing are permitted.

(2) Subscripted or indexed data-names may not be used as qualifiers.

(3) When condition-names are assigned to items requiring subscripting or indexing, these condition-names must be subscripted or indexed when referenced.

(4) Relative indexing (index-name ± integer) is permitted.

(5) Table handling is discussed in Section 8.

2.5. THE CODING FORM

Figure 2–1 shows the layout of the COBOL programming form. On this form the programmer enters all information needed by the COBOL compiler, observing the rules of format and content defined in this manual. Each line of written information represents the information to be entered into one 80-column punched card. Table 2–3 explains the several divisions of the form.

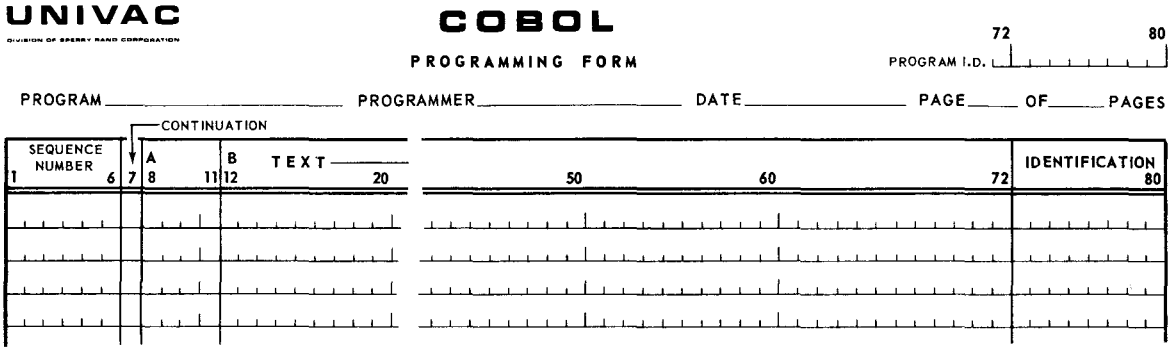


Figure 2–1. The COBOL Programming Form

COLUMNS	DESIGNATION	CONTENTS
1–6	SEQUENCE NUMBER	A numeric entry, used only by the programmer (not the COBOL processor) to establish a sequence among the various lines of coding (optional).
7	CONTINUATION	A hyphen (-) is used when an entry extending past one noncomment line has a break occurring in the middle of a word. The hyphen is written in column 7 of the next contiguous line on which the word is completed. A word may be interrupted in any column, the rest of the line space filled, and completed on the next line. If the continued line contains a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark.
7	COMMENT	An asterisk (*) in column 7 signifies a comment line which will be printed but ignored by the compiler. A comment may appear anywhere in the program and can contain any printable combination of characters including reserved words. If a comment entry extending past one line has a break occurring in the middle of a word, the continuation line must contain an asterisk in column 7. (The hyphen is only used for noncomment continuation lines.)
8–72	TEXT	All COBOL-formatted information, in the form of names, statements, information, instructions, etc., that is to be compiled into the object program. Note that two left-margin limits designated "A" and "B" are shown. These are needed for program alignment. Major definitive names are begun at margin A (column 8). Margin B (column 12) is used for subordinate items and for continuations of entries from the last preceding line.
73–80	IDENTIFICATION	Card deck information (optional).

Table 2–3. Programming Form Column Usage

3. IDENTIFICATION DIVISION

3.1. GENERAL

The IDENTIFICATION DIVISION identifies or labels the source program and provides other pertinent information concerning the program. All information given in this division is listed by the printer during compilation; however, only the PROGRAM-ID clause will affect the object program.

Format:

IDENTIFICATION DIVISION.

PROGRAM-ID. *program-name*.

[AUTHOR. [*comment-entry*]. . .]

[INSTALLATION. [*comment-entry*]. . .]

[DATE-WRITTEN. [*comment-entry*]. . .]

[DATE-COMPILED. [*comment-entry*]. . .]

[SECURITY. [*comment-entry*]. . .]

[REMARKS. [*comment-entry*]. . .]

Rules:

(1) IDENTIFICATION DIVISION must be present in all source programs.

(2) PROGRAM-ID must always be present as the first paragraph of the IDENTIFICATION DIVISION. Program-name may consist of 1 to 30 alphabetic or numeric characters, the first character being alphabetic. The sequence formed by the first six characters must be unique since it will identify the source program, object program elements, and associated documents. Hyphens within the first 6 characters are removed by the compiler due to 9400 System naming conventions.

If the program name is not supplied or not accepted because of an error, the compiler automatically supplies the program name COBOBJ.

(3) AUTHOR is for documentation only.

(4) INSTALLATION is for documentation only.

(5) DATE-WRITTEN is for documentation only.

(6) DATE-COMPILED is for documentation only. Date of compilation appears on listing regardless of whether this paragraph is present. Comment-entry is printed when this paragraph is present.

(7) SECURITY is for documentation only.

(8) REMARKS is for documentation only.

(9) A comment entry can consist of any printable combination of characters including reserved words.

4. ENVIRONMENT DIVISION

4.1. GENERAL

The Environment Division specifies those elements of the COBOL program which depend upon the physical aspects of the UNIVAC 9400 System.

Format:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-9400.
OBJECT-COMPUTER. UNIVAC-9400
[,MEMORY SIZE integer {WORDS      }] [ , SEGMENT-LIMIT IS priority-number ].
                      {CHARACTERS }
                      {MODULES    }
[SPECIAL-NAMES. entry.]
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {entry.}
[I-O-CONTROL.  {entry.}]
```

Rules:

(1) The Environment Division must be present in all source programs. It may have to be rewritten when a program is to be compiled or executed on a different computer configuration.

(2) Section and paragraph headers are required when their associated entries are present.

4.2. CONFIGURATION SECTION

Definition:

The Configuration Section specifies the characteristics of the source and object computers and relates implementor-names to user-names.

Format:

```
CONFIGURATION SECTION.
SOURCE-COMPUTER. entry.
OBJECT-COMPUTER. entry.
[SPECIAL-NAMES. entry.]
```

4.2.1. SOURCE-COMPUTER

Function:

To name the computer that will compile the source program.

Format:

SOURCE-COMPUTER. UNIVAC-9400.

Rule:

The SOURCE-COMPUTER entry is for documentation only and has no effect on the object program.

4.2.2. OBJECT-COMPUTER

Function:

To specify the computer that will execute the object program and its memory size.

Format:

OBJECT-COMPUTER. UNIVAC-9400
[MEMORY SIZE *integer* { WORDS / CHARACTERS / MODULES }] [, SEGMENT-LIMIT IS *priority-number*].

Rules:

(1) The OBJECT-COMPUTER entry has no effect on the object program unless the SEGMENT-LIMIT clause is specified.

(2) MEMORY SIZE is an optional clause defining memory as an integer number (no sign, comma, or decimal point permitted) of WORDS, CHARACTERS, or MODULES. The equivalent number of bytes for each is as follows:

- CHARACTER = 1 byte
- WORD = 4 bytes
- MODULE = 16,384 bytes

(3) The SEGMENT-LIMIT priority number must be an integer ranging in value from 1 through 49.

(4) When the SEGMENT-LIMIT clause is specified, only those sections having priority-numbers from 0 up to, but not including, the priority number designated as the limit, are considered as part of the fixed permanent segment.

(5) Sections having priority numbers from the SEGMENT-LIMIT through 49 are considered as fixed overlayable segments.

(6) When the SEGMENT-LIMIT clause is omitted, all sections having priority numbers from 0 through 49 are considered as belonging to the fixed permanent segment.

(7) The SEGMENT-LIMIT clause is available only in the extended compiler. In the basic compiler, an implicit SEGMENT-LIMIT of 50 is always in effect.

4.2.3. SPECIAL-NAMES

Function:

Provides a method of relating implementor-names to user-supplied mnemonic-names.

Format:

```
SPECIAL-NAMES.
[CURRENCY SIGN IS literal]
[DECIMAL-POINT IS COMMA]
[SYSCOM IS mnemonic-name-1]
[SYSDATE IS mnemonic-name-2]
[SYSTIME IS mnemonic-name-3]
[SYSCONSOLE IS mnemonic-name-4]
[SYSCHAN-t IS mnemonic-name-5] . . .
[SYSLST IS mnemonic-name-7]
[SYSERR[-m]
    {ON STATUS IS condition-name-3  [,OFF STATUS IS condition-name-4]}
    {OFF STATUS IS condition-name-4 [,ON STATUS IS condition-name-3] } ] . . .
[SYSSWCH[-n]
    {[IS mnemonic-name-7][,ON STATUS IS condition-name-5
                          ,OFF STATUS IS condition-name-6
                                [,OFF STATUS IS condition-name-6]
                                [,ON STATUS IS condition-name-5]]
    {ON STATUS IS condition-name-7  [,OFF STATUS IS condition-name-8]
    {OFF STATUS IS condition-name-8 [,ON STATUS IS condition-name-7] } ] . . .
```

NOTES: t = any digit 4 through 15
m = any digit 0 through 15
n = any digit 0 through 7

Rules:

(1) A comma or semicolon may separate each entry, and a period must follow the last entry. Each entry must begin on a new line.

(2) The CURRENCY clause literal is used in the PICTURE clause to represent the currency symbol. Absence of this clause specifies that $ is the currency symbol. The literal must be a nonnumeric literal consisting of one character from the COBOL character set and must not be one of the following characters.

- Digits: 0 through 9
- Alphabetic characters: A, B, C, D, P, R, S, V, X, Z, or space
- Special characters: *, + – . ; ( ) "

(3) The DECIMAL-POINT IS COMMA clause causes the functions of the decimal point and the comma to be interchanged in PICTURE clause character strings and in numeric literals.

Examples:

SPECIAL-NAMES. CURRENCY SIGN IS 'F' DECIMAL-POINT IS COMMA.

Source PICTURE	Source Data	Receiving Field PICTURE	Receiving Field Result
9(6)V99	00003232	FFFFFF,99	~~bb~~bF32,32
9(5)V99	1234567	F**.***,99	F12.345,67
9(9)V9(4)	0000098211289	Z(3).ZZ9,9(4)	~~bb~~9.821,1289

(4) SYSCOM permits accessing the communications region in the preamble of the job in which the object program is being executed via user-supplied mnemonic-name-1. See *UNIVAC 9400 System Supervisor Programmers Reference, UP-7689* (current version) for an explanation of data.

(5) SYSDATE permits access to current date via the user-supplied mnemonic-name-2. Mnemonic-name-2 may not appear in a DISPLAY statement. Date may be set or changed in the job control stream.

(6) SYSTIME permits access to time-of-day via a mnemonic-name-3. Mnemonic-name-3 may not appear in DISPLAY statement.

(7) SYSCONSOLE permits access to the console typewriter (using ACCEPT or DISPLAY statement) via mnemonic-name-4.

(8) SYSCHAN-t equates a particular channel (t) on the printer loop to mnemonic-name-5. Mnemonic-name-5 may appear only in a WRITE statement. SYSCHAN 9 and 15 are normally used for form overflow and top-of-page, respectively.

(9) SYSERR[-m] permits access to system error codes. The status of a particular error (m) or the presence of any error can be checked with the ON/OFF STATUS option. SYSERR[-m] is a feature of the extended compiler, random access module. Condition-names in ON/OFF STATUS phrases are defined and equated with ON or OFF as required by the compiler and should not be defined elsewhere in the COBOL program.

(10) SYSSWCH and its various options permit the programmer to access all or part of the User Program Switch Indicator (UPSI) byte. The eight bits in the UPSI byte (bits 0 through 7) constitute a set of eight programmable software switches, SYSSWCH-0 through SYSSWCH-7. The status of these switches can be set to ON or OFF, altered, or interrogated as required. A switch containing a 1 bit is ON; a 0 bit is OFF. The following examples show the various ways of using SYSSWCH.

(a) To set or change the contents of SYSSWCH, the DISPLAY verb may be used as follows:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
   SYSSWCH IS SWITCH
   SYSSWCH-3 IS SWITCH-3.
PROCEDURE DIVISION.
```

DISPLAY 00010001 UPON SWITCH.	SYSSWCH will now contain 00010001.
DISPLAY 1 UPON SWITCH-3.	SYSSWCH-3 will now contain 1; the other switches remain unchanged.
DISPLAY *identifier* UPON SWITCH.	The eight switches in SYSSWCH (0 through 7) are set ON or OFF, depending on the contents of the 8-character identifier.

NOTE: Any character other than a hexadecimal F0 will set a switch to ON.

(b) An individual switch can be interrogated by using condition-name in the ON/OFF STATUS option. For instance, in the following example control will be transferred to procedure-name-1 if switch 5 is ON.

```
ENVIRONMENT DIVISION.

SPECIAL-NAMES.

SYSSWCH-5 ON STATUS IS FIVON, OFF STATUS IS FIVOFF.

PROCEDURE DIVISION.
     .
     .
     .
IF FIVON GO TO procedure-name-1.
```

In essence, SYSSWCH-5 is a conditional variable with the condition-names FIVON and FIVOFF which are similar to level-88 entries.

The condition-names FIVON and FIVOFF are defined and equated with ON and OFF, respectively, by the COBOL compiler and must not be defined elsewhere in the COBOL program. The compiler uses the hexadecimal characters F0 and F1, respectively, to represent the OFF and ON status of a switch.

(c) The entire UPSI byte may be interrogated by use of the ACCEPT verb. This is shown in the following example where procedure-name-1 will be PERFORMed if the SYSSWCH-2, SYSSWCH-4, and SYSSWCH-6 switches are ON and the others are OFF.

```
ENVIRONMENT DIVISION.

SPECIAL-NAMES.

SYSSWCH IS mnemonic-name-1.

DATA DIVISION.

identifier PICTURE X(8).
```

```
PROCEDURE DIVISION.

ACCEPT identifier FROM mnemonic-name-1.

IF identifier = 00101010 PERFORM procedure-name-1.
    .
    .
    .
```

(d) Another way to interrogate switches is:

```
SPECIAL-NAMES.

SYSSWCH ON STATUS IS OK, OFF STATUS IS NIX.

PROCEDURE DIVISION.
    .
    .
    .
IF OK GO TO procedure-name-1.
```

In this example, if any switch is set to 1 the program will GO TO procedure-name-1.

(e) The mnemonic-name option allows the user to equate his mnemonic-name with the implementor-name SYSSWCH[-n]. For instance:

```
SPECIAL-NAMES.

SYSSWCH IS MYSWITCH, ON STATUS IS MYSWITCHON.
```

or

```
SYSSWCH-4 IS TAKETAX, ON STATUS IS LOFICA; OFF STATUS IS EQFICA.
```

The mnemonic-name option is for use with the ACCEPT or DISPLAY verbs only.

(f) The UPSI switches can also be accessed by the following job control statements:

- SET statement – used to set switches ON or OFF (1 or 0).
- SKIP statement – used to conditionally bypass control statements. If the UPSI switch settings match the bit pattern specified in the SKIP statement, the specified number of statements will be skipped.

The format and usage of these statements are shown in the *UNIVAC 9400 System Job Control Programmers Reference, UP-7585* (current version).

(11) Table 4–1 shows how SPECIAL-NAMES are handled by the compiler. Note that if the PICTURE clause is other than shown in the IMPLIED DESCRIPTION column in the table, the rules for the MOVE statement determine the storing of the result. The effect is that of a MOVE in which the sending item is described as shown in the STORED AS column and the receiving item description is that supplied by the user for identifier when ACCEPTing. The sending and receiving fields are reversed when DISPLAYing.

(12) SYSLST permits access to the printer by way of mnemonic-name-7 for DISPLAY functions.

SPECIAL-NAME	STORED AS	USABLE WITH ACCEPT	USABLE WITH DISPLAY	FORMAT	IMPLIED DESCRIPTION FOR ACCEPT OR DISPLAY (3)	EXPLANATION
SYSCOM	12 alphanumeric characters	YES	YES	12 EBCDIC characters	PIC X(12)	See Supervisor manual, UP-7689
SYSDATE	6 numeric characters	YES	NO	YYMMDD	PIC 9(6)	Current Day
SYSTIME	8 numeric characters	YES	NO	hhmm0000	PIC 9(8)	Time Of Day
SYSCONSOLE	variable length alpha-numeric characters	YES	YES	For DISPLAY: 59 char/line, 4092 max. For ACCEPT: 60 char. max.	PIC X(n)	Typewriter Console
SYSCHAN-t(2)	not applicable	NO	NO	not applicable	not applicable	To assign name to printer loop channel
SYSERR[-m]	NA	NO	NO	not applicable	not applicable	Refer to Appendix D
SYSSWCH	8 alphanumeric characters	YES	YES	8 EBCDIC characters	PIC X(8)	To call or change UPSI bits
SYSSWCH-n	1 alphanumeric character	NO	YES	1 EBCDIC character	PIC X	To change UPSI bits individually
SYSLST	variable length alphanumeric characters	NO	YES	132 char/line	PIC X (n)	Printer with LFD name of SYSLST
ON STATUS (1)	not applicable	NO	NO	not applicable	not applicable	To interrogate user program switch indicators (UPSI) for ON or OFF condition
OFF STATUS (1)	not applicable	NO	NO	not applicable	not applicable	

NOTES:
(1) Can be used only in conditional variable tests.
(2) Can be used only in ADVANCING clause of WRITE statement.
(3) See 4.2.3, rule 11.

Table 4-1. Rules for SPECIAL-NAMES

4.3. INPUT-OUTPUT SECTION

Definition:

This section of the Environment Division is used to specify the input/output media for the files used by the source program and to provide information needed for most efficient transmission of data between this media and the object program.

Format:

```
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {entry.}. . .
 [I-O CONTROL.  entry.]      ]
```

4.3.1. FILE-CONTROL

Function:

To name each file, identify the hardware medium which contains it, permit specific hardware assignments for the program, and specify alternate input/output areas. The clauses following the SELECT and ASSIGN statements under FILE CONTROL may be specified in any order.

Format:

```
FILE-CONTROL. { SELECT [OPTIONAL] file-name
ASSIGN TO [external-name][integer-1] implementor-name-1[OR implementor-name-2]
[ FOR MULTIPLE {REEL}                                                          ]
               {UNIT}
[, RESERVE {integer-2} ALTERNATE [AREA ]                                       ]
           {NO       }           [AREAS]
[, {FILE-LIMIT IS  } {data-name-1} THRU {data-name-2}                          ]
   {FILE-LIMITS ARE} {literal-1  }      {literal-2  }
[, {data-name-3} THRU {data-name-4} ]. . .                                     ]
   {literal-3  }      {literal-4  }
[, ACCESS MODE IS {RANDOM    }                                                 ]
                  {SEQUENTIAL}
[, PROCESSING MODE IS SEQUENTIAL]
[, ORGANIZATION IS {RELATIVE  }                                                ]
                   {DIRECT    }
                   {SEQUENTIAL}
                   {INDEXED   }
[{ACTUAL KEY IS data-name-5  }                                                 ]
 {RELATIVE KEY IS data-name-6}
[SYMBOLIC KEY IS data-name-7]
[, RECORD KEY IS data-name-8]. { . . .
```

Rules:

(1) The comma or semicolon may separate each entry, and a period must follow the last entry.

(2) SELECT clause must be specified for the following:

- All files which are the subject of an FD or SD must also be the subject of a SELECT.
- A SELECT clause must be supplied for a RERUN (external-name operand) for which no FD or SD is supplied.
- The key word OPTIONAL is required for files that are not necessarily present each time the object program is run. The status of the OPTIONAL file at run time is determined by the job control stream. The file must be opened in the program. If the file is not present in the job stream, control takes the AT END path on the first READ statement. The OPTIONAL key word can be applied to INPUT files only.

(3) The ASSIGN clause designates a particular hardware device, or class of devices, to which a specific file is ASSIGNed. External-name is a nonnumeric literal (1 to 8 characters) which is associated with a file. This is the name used in the job control stream to assign devices to the file (using the // LFD card; see *UNIVAC 9400 System Job Control For Disc Systems Programmers Reference, UP-7585* (current version)). The external name must be unique within a job. If external-name is omitted, the first eight characters of file-name are assumed for external-name. Integer-1 serves as documentation only, referring to the number of devices associated with the file. UNIVAC 9400 COBOL assigns the following implementor-names:

DEVICE	IMPLEMENTOR-NAME
51-column card reader	CARD-READER-51
66-column card reader	CARD-READER-66
80-column card reader	CARD-READER
card punch	CARD-PUNCH
line printer	PRINTER
disc (nonspecific)	DISC (DISC and DISC-8411 are equivalent)
8411 disc	DISC-8411
8414 (disc)	DISC-8414
UNISERVO VI-C	TAPE-6
all other tapes	TAPE

The implementor-name, DISC, specifies an assignment to the 8411 disc. Because of track size differences, the user must insure that the proper implementor-name is used when ASSIGNing discs.

(4) The MULTIPLE clause, when present, specifies that the file exists on more than one volume. This clause is accepted for documentation purposes only, since the actual function is provided via the job control stream.

(5) The RESERVE clause indicates the number of additional I/O areas desired. The NO option causes no additional I/O areas to be reserved. The integer-2 option reserves one additional I/O area. Integer-2 must be a 1; if not and the word NO is not specified, a warning diagnostic will be issued. Omission of this clause may result in the allocation of one additional I/O area as indicated in the following chart:

DEVICE		NO. OF ADDITIONAL I/O AREAS ALLOCATED IF CLAUSE NOT SPECIFIED	RESERVE INTEGER ALLOWED
CARD-READER		1	YES
CARD-PUNCH		1	
PRINTER		1	
TAPE		1	
DISC	ORGANIZATION SEQUENTIAL (or omitted)	1	
	ORGANIZATION {RELATIVE / DIRECT / INDEXED}	0	NO

(6) FILE-LIMIT clause serves as documentation only.

(7) ACCESS MODE specifies the manner in which the records of a file are read and/or written. Random access mode is an extended compiler feature. Absence of this clause results in assumption of sequential access.

(8) PROCESSING MODE clause is for documentation only. Sequential processing is always assumed, regardless of the absence or presence of this clause.

(9) The ORGANIZATION clause designates the physical structure of the file. Sequential organization is assumed if the clause is omitted. This clause is an extension to American National Standard COBOL.

(10) ACTUAL KEY data-name-5 is used with direct and relative file organizations. When used in conjunction with a relative file, the location associated with data-name-5 contains a relative record number (i.e, record 1, 2, 3, ...). When used in conjunction with a direct file, the location associated with data-name-5 contains a file relative track address. For direct organization files, SYMBOLIC KEY, is used in conjunction with ACTUAL KEY. Data-name-5 must be defined as an unsigned integer according to the rules for numeric items.

For compatibility with 9300 COBOL, ACTUAL KEY may be specified in place of SYMBOLIC KEY when used with indexed file organizations.

(11) SYMBOLIC KEY data-name-7 is used with direct and indexed file organizations to supply the record identification. For indexed-sequential files, the information associated with the RECORD KEY clause must be identical to the information associated with the SYMBOLIC KEY clause. Data-name-7 must be described as an alphabetic, alphanumeric, or a numeric (display) item with a length not greater than 255 characters (may be numeric computational if used with an indexed file). This clause is an extension to American National Standard COBOL.

(12) RELATIVE KEY data-name-6 is used with relative organization files to supply the physical position of a record with respect to the beginning of the file. Records in a relative organization file are addressed as record 1, 2, 3, and so on. As noted in Rule (10), the ACTUAL KEY clause may be substituted for the RELATIVE KEY clause. Data-name-6 must be defined as an unsigned numeric integer according to the rules for numeric items. This clause is an extension to American National Standard COBOL.

(13) RECORD KEY data-name-8 is used for indexed-organized files to supply the record identification field. Data-name-8 must be described as alphabetic, alphanumeric, or numeric item consisting of 3 to 255 characters. If data-name-8 is defined as numeric computational, it must contain at least 4 digits (3 bytes). This clause is an extension to American National Standard COBOL.

A detailed explanation of the various keys and types of file organization is given in Appendix D.

4.3.2. I-O-CONTROL

Function:

To specify the following:

- input/output techniques,
- memory area shared by various files,
- location of each file on multiple-file reel, and
- intervals at which rerun is to be established.

Format:

```
I-O-CONTROL.
[; RERUN ON external-name
 EVERY integer-1 RECORDS OF file-name-1 [, file-name-2]]...
[; SAME [RECORD / SORT] AREA FOR file-name-3 [, file-name-4]...]...
[; MULTIPLE FILE TAPE CONTAINS file-name-5
 [POSITION integer-2] [, file-name-6 [POSITION integer-3]]...]...
[APPLY RESTRICTED SEARCH OF {ALL / integer-4}
 TRACKS ON file-name-7 [, file-name-n]...]...
[APPLY VERIFY ON file-name-8 [, file-name-n]...]...
[APPLY BLOCK-COUNT ON {file-name-9 [, file-name-10]... / TAPES}]...
[APPLY MASTER-INDEX ON file-name-11 [, file-name-12]...]...
[APPLY CYLINDER-INDEX AREA OF integer-5 INDICES ON file-name-13
      [, file-name-14]...]...
[APPLY CYLINDER-OVERFLOW AREA OF integer-6 PERCENT ON file-name-15
      [, file-name-16]...]...
[APPLY EXTENDED-INSERTION AREA ON file-name-17 [, file-name-18]...]...
[APPLY FILE-PREPARATION ON file-name-19 [, file-name-20]...]...
[APPLY ASCII [WITH BUFFER-OFFSET {FOR BLOCK-LENGTH-CHECK / OF integer CHARACTERS}]
 ON file-name-21 [,filename-22] ...]...
```

Rules:

(1) The comma or semicolon may separate each entry, and a period must follow the last entry.

(2) RERUN specifies that checkpoint records are to be written on the disc or tape unit specified by external-name. A checkpoint record is the recording of the status of the computer at a given point during the execution of the object program. All the information required to restart the program at that point is contained in the checkpoint record. These records will be written whenever integer-1 records occur for file-name-1. File-name-1, file-name-2 ... can appear in only one RERUN statement; external-name can appear in any number of RERUN statements. The allowable range of integer-1 is 1 to 9,999,999.

(3) SAME AREA specifies that two or more files are to use the same memory area during processing. When the key word RECORD is omitted, the area being shared includes all storage areas assigned to the files; therefore, only one file may be OPEN at a time. If RECORD is specified, any number of files may use the same storage area for processing the current logical record (the record format of such files must not conflict). The SAME RECORD AREA clause should be used only when necessary as it reduces efficiency.

If the SAME SORT AREA clause is used, at least one of the file-names must be an SD. Storage area assigned to files that are not sort files will be allocated in the sort file area if they appear in this clause. These files must not be OPEN during the execution of a SORT.

Files that appear in a SAME AREA and a SAME SORT AREA clause share the same space within the sort file area. If any nonsort file is mentioned in both clauses, all files in the SAME AREA clause must appear in the SAME SORT AREA clause.

(4) The MULTIPLE FILE clause is for documentation only. This feature is supported by Job Control (see Appendix I).

(5) APPLY RESTRICTED SEARCH is used with direct organization disc files to permit searching for a record through ALL tracks or integer-4 tracks on file-name-7. When a record cannot be located for a READ command, the INVALID KEY option will be executed. If this clause is omitted, the ALL option is assumed. The allowable range of integer-4 is 1 to 254.

(6) The APPLY VERIFY clause requests verification (READ after WRITE) of disc records after they have been written. Absence of this clause results in no verification of records written.

(7) The APPLY BLOCK-COUNT causes a three-byte block number to be inserted at the beginning of each block on tape for each file-name designated. If the TAPES option is specified, all tape files present are affected. This clause must be present for all input files which contain a BLOCK-COUNT.

(8) The APPLY FILE-PREPARATION clause indicates that the tracks allocated to a relative or direct organized file are to be recorded with initializing data prior to creation of a file. The track initialization occurs after an OPEN OUTPUT is issued. (Refer to Appendix D for more detailed information.)

(9) The APPLY MASTER-INDEX clause, used only with indexed files, indicates that a master index is to be generated when an indexed file is created.

(10) The APPLY CYLINDER-INDEX integer-5 clause, used only with indexed files, indicates that sufficient main storage area is to be allocated to contain integer-5 cylinder index entries.

(11) The APPLY CLINDER-OVERFLOW integer-6 clause, used only with indexed-sequential files, indicates that integer-6 percent of each cylinder in the prime data area is to be reserved for the purpose of cylinder overflow. If this clause is omitted, 20 percent of the cylinders specified are automatically allocated. If no overflow is desired, 0 percent should be specified. Integer-6 is an unsigned number.

(12) The APPLY EXTENDED INSERTION clause, used only with indexed files, indicates that sufficient main storage is to be allocated to contain all records on a prime data area track. This area is used when records are inserted into a file. The use of the main storage area reduces the number of accesses necessary to complete the insertion process.

Integer-6 should be specified in percentages that indicate whole tracks. For an 8411 disc, integer-6 should be in multiples of 10 percent. For an 8414 disc, integer-6 should be in multiples of 5 percent. If integer-6 is not specified as such it will be rounded to the nearest appropriate multiple.

(13) The APPLY ASCII clause identifies each file that contains or receives ASCII data. See Appendix N.

NOTE: APPLY clauses (rules 5 through 13) are extensions to American National

5. DATA DIVISION

5.1. GENERAL

Every data item referenced in the Procedure Division must be described in the Data Division, except for the special register TALLY, index-names, figurative constants, and literals. File structures are described by File Description entries; data items and records are described by Record Description or single item entries.

Format:

```
DATA DIVISION.
[FILE SECTION.
      .
      .
      .    ]
[WORKING-STORAGE SECTION.
      .
      .
      .    ]
[LINKAGE SECTION.
      .
      .
      .    ]
```

Rules:

(1) The header DATA DIVISION must be present in all COBOL programs.

(2) Sections are written in the order shown; if a section is not required, it may be omitted entirely.

(3) Data-names used in FD, SD, or 77 level entries must be unique since they cannot be qualified. The same is also true for data-names used in 01 entries within the Working-Storage and Linkage Sections of the source program.

5.1.1. Data Definition

Table 5–1 shows the allowable sizes of data items in UNIVAC 9400 COBOL. Data type is determined by the PICTURE clause. See 5.3.4 for legal PICTURE character for each data type.

DATA TYPE	COBOL CHARACTERS		AREA IN BYTES	
	MINIMUM	MAXIMUM	MINIMUM	MAXIMUM
GROUP (WORKING-STORAGE)	1	65,535	1	65,535
GROUP (FILE or LINKAGE SECTION)	1	4092	1	4092
ALPHANUMERIC	1	4092	1	4092
ALPHABETIC	1	4092	1	4092
ALPHABETIC EDITED	2	132	2	132
NUMERIC EDITED	2	132	2	132
NUMERIC DISPLAY	1	18	1	18
NUMERIC COMP	1 (plus sign)	18 (plus sign)	1	10
INDEX NAME ITEM	N/A	N/A	8	8
INDEX DATA ITEM	N/A	N/A	8	8

Table 5–1. Allowable Memory Size

5.2. FILE SECTION

The File Section consists of:

- File Description (FD) entries describing the structure of all files and naming the data records contained in each.
- Record description entries immediately follow each File Description entry describing in detail each record format used in the file.

Format:

FD *file-name-1 (file description clauses)*
01 *record-name-1 (record description clauses)*
.
.
.
[01 *record-name-m (record description clauses)*]
.
.
.
[FD *file-name-n*
.
.
.
]

5.2.1. File Description

Function:

To provide information concerning the physical structure, labeling, and record names of a given file.

Format:

FD *file-name*

[; BLOCK CONTAINS [*integer-1* TO] *integer-2* {RECORDS / CHARACTERS}]
[; RECORD CONTAINS [*integer-3* TO] *integer-4* CHARACTERS]
; LABEL {RECORDS ARE / RECORD IS} {STANDARD / OMITTED / *data-name-1* [, *data-name-2*] . . .}
[; RECORDING MODE IS {F / U / V / D}]
[; VALUE OF {*unqualified-data-name* IS {*data-name-3* / *literal-1*} . . .}]
[; DATA {RECORD IS / RECORDS ARE} *data-name-4* [, *data-name-5*] . . .].
SD *file-name*
[; RECORD CONTAINS [*integer-5* TO] *integer-6* CHARACTERS]
[; RECORDING MODE IS {F / V / D}]
[; DATA {RECORD IS / RECORDS ARE} *data-name-7* [, *data-name-8*] . . .].

Rule:

The various clauses may appear in any order after file-name.

5.2.1.1. BLOCK CONTAINS

Function:

To specify the size of a physical record.

Format:

BLOCK CONTAINS [*integer-1* TO] *integer-2* { RECORDS / CHARACTERS }

Rules:

(1) Integer-1 and integer-2 must be unsigned integers other than zero.

(2) When RECORDS is specified, this clause specifies the number of records per block.

(3) When CHARACTERS is specified, this clause specifies the number of characters (bytes) per block. This number does not include the three bytes added when the APPLY BLOCK-COUNT clause is used, but does include the block and record length characters that are present when RECORDING MODE V is specified.

(4) When CHARACTERS and RECORDS are both omitted, CHARACTERS is assumed.

(5) When this clause is omitted, it is assumed that records are recorded one per block and the record size is fixed.

(6) If integer-1 and integer-2 are both specified, they refer to the minimum and maximum number of characters or records per block. Block size ranges are given in Table 5–2.

HARDWARE DEVICE AND IMPLEMENTOR-NAME	BYTES PER BLOCK	
	MINIMUM	MAXIMUM
UNISERVO VI-C TAPE (TAPE-6)	18	4096
UNISERVO VI-C TAPE (TAPE-6) WITH BLOCK NUMBERING	18	4092
OTHER TAPE (TAPE)	18	32,767
OTHER TAPE (TAPE) WITH BLOCK NUMBERING	18	32,763
PRINTER	1	132
CARD-READER	1	80
CARD-PUNCH	1	80
CARD-READER-51	1	51
CARD-READER-66	1	66
8411 DISC (DISC or DISC-8411)	1	3625
8414 DISC (DISC-8414)	1	7294

Table 5–2. Block Size Ranges

5.2.1.2. RECORD CONTAINS

Function:

To specify the size of data records.

Format:

RECORD CONTAINS [*integer-1* TO] *integer-2* CHARACTERS

Rules:

(1) Integer-1 and integer-2 must be unsigned integers other than zero; integer-2 must be greater than integer-1.

(2) The size of each data record is completely defined within the Record Description entry; therefore, this clause is optional. When present, however, the following notes apply:

(a) If integer-2 is used alone, all the data records in the file must have the same size. In this case, integer-2 represents the exact number of characters in the data record.

(b) If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size data record. Record size ranges are given in Table 5–3.

HARDWARE DEVICE AND IMPLEMENTOR-NAME	BYTES PER RECORD		
	MINIMUM	MAXIMUM	
		FIXED OR UNDEFINED RECORD FORMAT	VARIABLE RECORD FORMAT
UNISERVO VI-C TAPE (TAPE-6)	18	4092	4088
UNISERVO VI-C TAPE (TAPE-6) WITH BLOCK NUMBERING	18	4092	4084
OTHER TAPE (TAPE)	18	4092	4092
PRINTER	1	132	132
CARD-READER	1	80	N/A
CARD-PUNCH	1	80	80
CARD-READER-51	1	51	N/A
CARD-READER-66	1	66	N/A
8411 DISC (DISC or DISC-8411)	1	3625	3617
8414 DISC (DISC-8414)	1	4092	4092

Table 5–3. Record Size Ranges

5.2.1.3. LABEL RECORDS

Function:

To enable the compiler to cross-reference the description of a label record with its associated file.

Format:

$$\underline{\text{LABEL}} \left\{ \begin{matrix} \underline{\text{RECORDS}} \text{ ARE} \\ \underline{\text{RECORD}} \text{ IS} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \\ \textit{data-name-1} \text{ [, } \textit{data-name-2}\text{] } \ldots \end{matrix} \right\}$$

Rules:

(1) OMITTED specifies that no standard labels exist for the file or the device to which the file is assigned. Any nonstandard labels must be described and processed as data records.

(2) STANDARD specifies that standard file labels exist for the file or the device to which the file is assigned, and the labels conform to UNIVAC 9400 label specifications. Standard user labels may also be present, but STANDARD specifies that they are not to be checked on input file, or written on output files.

(3) Data-name-1 [data-name-2]... specifies that standard labels are to be checked (or created), and that standard user labels are present. User labels must conform, in content and format, to the Univac 9400 standard user label specifications.

The following rules apply when data-name-1 is specified:

(a) Data-name-1 [data-name-2]... must have a record description subordinate to this file description.

(b) For input files, Data Management provides access to standard user label information in the area described by data-name-1.

(c) For output files, the user moves user label information into the area described by data-name-1 for Data Management to write to the output file.

(d) User label records can be referenced only in USE procedures in the Declaratives portion of the Procedure Division.

(4) The label record specifications for the various device types are as follows:

DEVICE		LABELS OMITTED	LABELS STANDARD	LABELS DATA-NAME
PRINTER		YES	NO	NO
CARD-READER				
CARD-PUNCH				
TAPE		YES	YES	YES
DISC	ORGANIZATION SEQUENTIAL	NO	YES	YES
	ORGANIZATION RELATIVE			YES*
	ORGANIZATION DIRECT			YES*
	ORGANIZATION INDEXED			NO

5.2.1.4. RECORDING MODE

Function:

To specify the format of the logical record comprising the file.

Format:

RECORDING MODE IS $\left\{\begin{matrix}F\\U\\V\\D\end{matrix}\right\}$

Rules:

(1) The F mode (fixed-length format) is specified when all the logical records in the file are of the same length.

(2) The U mode (undefined format) states that the records of this file are not blocked and may vary in length.

(3) The V mode (variable-length format) is specified when records within a file vary in length.

(4) The D mode may be specified for ASCII tape files with variable-length records.

(5) The following chart describes the recording mode assumed when the clause is omitted:

*Only BEGINNING user labels are allowed

DEVICE		ASSUMED FORMAT
PRINTER		F
CARD-READER		
CARD-PUNCH		
TAPE		V
DISC	ORGANIZATION SEQUENTIAL	
	ORGANIZATION RELATIVE	F
	ORGANIZATION DIRECT	
	ORGANIZATION INDEXED	

(6) RECORDING MODE is an extension to American National Standard COBOL.

5.2.1.5. VALUE OF

Function:

To describe a particular item in the standard file label record associated with a file. This clause serves as documentation only in the UNIVAC 9400 and is also accepted for compatibility with the UNIVAC 9300.

Format:

$$\underline{\text{VALUE OF}} \left\{ \textit{unqualified-data-name} \ \text{IS} \left\{ \begin{matrix} \textit{data-name-3} \\ \textit{literal-1} \end{matrix} \right\} \right\} \ldots$$

5.2.1.6. DATA RECORDS

Function:

To specify the names of the logical records in a file.

Format:

$$\underline{\text{DATA}} \left\{ \begin{matrix} \underline{\text{RECORDS ARE}} \\ \underline{\text{RECORD}} \ \text{IS} \end{matrix} \right\} \textit{data-name-1} \ [, \textit{data-name-2}] \ \ldots$$

Rules:

(1) This clause is optional and serves as documentation only.

(2) Each data-name specified must appear at a 01 level number following the FD entry.

5.2.2. Sort File Description

Function:

To identify the beginning of a Sort File Description (SD) and supply the name of the file.

Format:

SD *file-name*

[; RECORD CONTAINS [*integer-1* TO] *integer-2* CHARACTERS]

[; RECORDING MODE IS {F / V / D}]

[; DATA {RECORD IS / RECORDS ARE} *data-name-1* [, *data-name-2*]...]

Rules:

(1) An SD clause is required for each file to be sorted.

(2) Each data-name specified must appear as a 01 level-number following the SD entry.

(3) The RECORD CONTAINS, RECORDING MODE, and DATA RECORD clauses are described under the FD Entry.

(4) Recording mode V is assumed when the RECORDING MODE clause is omitted.

(5) File-name may appear only in the SORT and RETURN statements within the Procedure Division, and only those file-names which appear in SD entries may be used in those statements.

(6) A summary of the UNIVAC 9400 COBOL SORT formats is given in Section 9.

5.3. DATA DESCRIPTION

Function:

To define the characteristics of a particular data item.

Formats:

Format 1:

level-number { *unqualified-data-name-1* / FILLER }
[; REDEFINES *unqualified-data-name-2*]

[
Format 1:
; OCCURS *integer-2* TIMES [{ ASCENDING / DESCENDING } KEY IS *data-name-2* [, *data-name-3*] . . .] . . . [INDEXED BY *index-name-1* [, *index-name-2*] . . .]
Format 2:
; OCCURS [*integer-1* TO] *integer-2* TIMES DEPENDING ON *data-name-1*
[{ ASCENDING / DESCENDING } KEY IS *data-name-2* [, *data-name-3*] . . .] . . .
[INDEXED BY *index-name-1* [, *index-name-2*] . . .]
]

[; { PIC / PICTURE } IS *character-string*]

[; [USAGE IS] { COMP / COMPUTATIONAL / COMP-3 / COMPUTATIONAL-3 / DISPLAY / INDEX }]

[; MAP IS *integer-2* CHARACTERS]

[; { SYNC / SYNCHRONIZED } [{ LEFT / RIGHT }]]

[; { JUST / JUSTIFIED } RIGHT]

[; VALUE IS *literal*]

[; BLANK WHEN ZERO]

[
Format 1:
[SIGN IS] { LEADING / TRAILING } SEPARATE CHARACTER
Format 2:
[SIGN IS] TRAILING
]

5.3.1. Level Number and Unqualified-data-name/FILLER

Function:

The level number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for condition-names, noncontiguous working-storage items, and the RENAMES clause.

Format:

level-number { *unqualified-data-name* / FILLER }

Rules:

(1) A level number is required as the first element in each data description entry.

(2) Level-numbers 01 through 09 can be expressed without the leading zeros.

(3) Level-number 01 identifies the first entry in each Record Description.

(4) Level numbers start at 01 for records, and become successively higher for subsets of records, such as group and elementary items. The maximum level-number permitted is 49, except for 66, 77, 88.

(5) Level-number 66 is used only for the RENAMES clause.

(6) Level-number 77 is used in the Working-Storage Section to describe noncontiguous data items and constants.

(7) Level-number 88 is assigned to entries which define condition-names associated with a conditional variable.

(8) FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to directly. Also, FILLER must not be used with a level-number 88, but may be used to name the associated conditional variable.

5.3.2. REDEFINES

Function:

To allow the same area of computer memory to be described by different data descriptions.

Format:

level-number unqualified-data-name-1 [; REDEFINES *unqualified-data-name-2*]

Rules:

(1) The REDEFINES clause must immediately follow unqualified-data-name-1.

(2) The level numbers of unqualified-data-name-1 and unqualified-data-name-2 must be identical, and may not be 66 or 88.

(3) REDEFINES must not be applied to level 01 entries in the File Section, although this is permissible in the Working-Storage Section.

(4) Redefinition begins at unqualified-data-name-2 and continues until a level number less than, or equal to, that of unqualified-data-name-2 is encountered. A REDEFINES clause may be used within the range of another REDEFINES with a maximum of five levels permitted.

(5) When the level number being redefined is other than 01, unqualified-data-name-1 must specify a storage area equal to the storage area for unqualified-data-name-2.

(6) Unqualified-data-name-2 must not contain, or be subordinate to, and OCCURS clause.

(7) Entries described under unqualified-data-name-1 must not contain VALUE clauses except in condition-name entries (level-number 88).

(8) Multiple redefinition of the same storage area is permitted. The entries giving the new descriptions of the storage area must follow the entries defining the area being REDEFINEd; no intervening entries defining new storage are permitted. Multiple redefinitions of the same storage area must use the data-name of the entry that originally defined the area.

5.3.3. OCCURS

Function:

To eliminate the need for separate entries for repeated data, and to supply information required for the application of subscripts or indices.

Formats:

Format 1:

```
OCCURS integer-2 TIMES [ { ASCENDING  } KEY IS data-name-2
                       { DESCENDING }
[, data-name-3] . . . ] . . . [ INDEXED BY index-name-1 [, index-name-2] . . . ]
```

Format 2:

```
OCCURS [integer-1 TO] integer-2 TIMES DEPENDING ON data-name-1
[ { ASCENDING  } KEY IS data-name-2 [, data-name-3] . . . ] . . .
  { DESCENDING }
[INDEXED BY index-name-1 [, index-name-2] . . .]
```

Rules:

(1) The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than SEARCH. Further, if the subject of this entry is the name of a group item, all data-names belonging to the group must be subscripted or indexed whenever they are used as operands.

(2) An INDEXED BY clause is required if the subject of this entry, or a group item within it, is to be referenced by indexing. Index-name is not defined elsewhere by the user, because its format is dependent on the hardware and storage is allocated by the compiler.

(3) The data description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item described.

(4) The OCCURS clause cannot be specified in a data description entry that:

(a) Has an 01, 66, 77, or an 88 level-number.

(b) Describes an item of variable size. The size of an item is variable if the data description of any subordinate item contains format 2 of the OCCURS clause.

(5) Three levels of subscripting and indexing are permitted.

(6) Data-name-1, data-name-2, data-name-3, . . . may be qualified.

(7) The KEY IS phrase indicates that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The data-names are listed in their descending order of significance.

(8) Data-name-2 must be either the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause. If data-name-2 is not the subject of this entry, then:

(a) All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

(b) None of the items identified by data-names in the KEY IS phrase can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

(9) Data-name-3, etc. must be the name of an entry subordinate to the group item which is the subject of this entry.

(10) In format 1, the value of integer-2 represents the exact number of occurrences. The area allocated multiplied by the number of occurrences cannot exceed 65,535.

(11) Format 2 specifies that the subject of this entry contains a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject is variable but that the number of occurrences is variable. Integer-2 must be a positive or unsigned integer (not zero). The area allocated, multiplied by the number of occurrences, cannot exceed 65,535. Integer-1 may be positive or zero but must be less than integer-2. If integer-1 is not specified, a value of zero is assumed for the minimum number of occurrences. Integer-1 is required in ANS COBOL.

(12) A data description entry containing format 2 of the OCCURS clause may be followed only, within that record description, by data description entries subordinate to it.

(13) Any entry which contains, or has a subordinate entry which contains, format 2 cannot be the object of the REDEFINES clause.

(14) In format 2, the data item defined by data-name-1 must not occupy a computer storage position within the range of the first computer storage position defined by the data description entry containing the OCCURS clause and the last computer storage position defined by the record description entry containing that OCCURS clause.

(15) The value of data-name-1 is the count of the number of occurrences of the subject and its value must fall within the range integer-1 through integer-2. Reducing the value of data-name-1 makes the contents of data items, whose occurrence number now exceeds the value of data-name-1, unpredictable. The data description of data-name-1 must describe a positive integer.

(16) When a group item, having subordinate to it an entry that specifies format 2 of the OCCURS clause, is referenced, only that part of the table area that is specified by the value of data-name-1 will be used in the operation.

(17) Format 2 is available only in the extended compiler.

(18) The DEPENDING option (format 2) is only required, and should only be used, when the end of the occurrences cannot otherwise be determined.

(19) The VALUE clause must not be stated in a data description entry which contains an OCCURS clause or in any entry which is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

5.3.4. PICTURE

Function:

To describe the general characteristics and editing requirements of an elementary data item.

Format:

$$\left\{\begin{matrix}\underline{\text{PIC}} \\ \underline{\text{PICTURE}}\end{matrix}\right\} \text{IS } \textit{character-string}$$

Rules:

(1) PICTURE clause can be present only with an elementary item.

(2) Character-string can consist of one to 30 characters.

(3) There are five categories of data which can be described with a PICTURE clause:

- Alphabetic
- Numeric
- Alphanumeric
- Alphanumeric Edited
- Numeric Edited

Table 5–4 lists the allowable PICTURE symbols and the rules for their usage.

(4) To define an item as alphabetic:

(a) Its PICTURE character-string may consist of only the symbol A.

(b) Its contents, when represented in standard data format, must be any combination of the 26 letters of the alphabet and the space.

(c) Maximum number of character positions allowed is 4092.

(5) To define an item as numeric:

(a) Its character-string may consist of only the symbols 9, P, S, V, and H.

(b) The character-string must contain at least one 9.

(c) The maximum number of digits permitted in a numeric item is 18.

(d) The maximum number of occurrences of the symbol P in a picture-string is 17.

(e) Its contents, when represented in standard data format, must be any combination of the numerals 0 through 9, and the item may include an operational sign.

(f) H in a picture-string specifies that the USAGE of this item is COMP. H is an extension to ANS COBOL.

(6) To define an item as alphanumeric:

(a) Its character-string is restricted to X's or at least two of the symbols A, X, and 9, and is treated as if the picture-string were X's.

(b) Its contents, when represented in standard data format, are any combination of characters in the UNIVAC 9400 System character set.

(c) Maximum number of permitted character positions is 4092.

(7) To define an item as alphanumeric edited:

(a) Its character-string is restricted to combinations of the symbols A X 9 B 0 and must contain:

- at least one B and one X
- at least one zero and one X
- at least one zero and one A
- at least one B and one A

(b) Its contents, when represented in standard data format, are any combination of characters in the UNIVAC 9400 System character set.

(c) Maximum number of character positions permitted is 132.

(8) To define an item as numeric edited:

(a) Its character-string is restricted to certain combinations of the following symbols:

B P V Z CR DB 9 , . * + – 0 (zero) $ (currency sign)

The allowable combinations are determined by the sequence in which the symbols appear, and by the editing rules. The number of digit positions must not exceed 18.

(b) The maximum number of P's permitted is 17.

(c) Its contents, when represented in standard data format, must consist only of the numerals 0 through 9, plus editing symbols indicated.

(d) Maximum number of permitted character positions is 132.

(9) The following symbols may appear only once in a given picture-string.

S V . CR DB H

(10) An integer enclosed in parentheses and following any of the symbols A , X 9 P Z * B 0 + – $ indicates the number of consecutive occurrences of the symbol.

(11) The order of precedence for characters used as symbols in a character-string is listed in Table 5–5.

(12) Examples of Source Fields and Receiving Fields are listed in Table 5–6.

PICTURE SYMBOL	REPRESENTS	CAN BE USED IN COMBINATION WITH	SPECIAL PICTURE POSITION
9	A numeric character	Any other symbol	None
S	An operational sign is associated with the data item.	P V 9 H	Can be preceded only by H. Only one S is permitted.
V	Assumed decimal point in data item.	Any symbol except: A X and is redundant with P.	Only one is permitted, can precede leading P or follow trailing P.
P	Assumed decimal point outside of data item. Each P represents one character position.	Any symbol except: A X	Must be first or last symbol or symbols of PICTURE except for S, CR, DB, V or single +, - or $ but cannot be both first and last.
A	An alphabetic character or space.	X 9 B 0	None
X	An alphanumeric character	A 9 B 0	None
Z	Suppression of leading 0's (replaced by blanks or spaces).	Any symbol except: * A X S H or more than one: $ + -	Can be preceded only by: V . , $ + - P B 0 (zero)
*	Check protection, replaces leading 0's with asterisks.	Any symbol except: Z A X S H or more than one: $ - +	Can be preceded only by: V . , $ + - P B 0 (zero)
, (comma)	Insert comma in character position unless the preceding position has been blanked.	Any symbol except: A X S H	None
. (period)	Actual decimal point to be inserted in character position unless following positions have been blanked.	Any symbol except: A X P V S H	None
B	Insert a blank or space in character position.	Any symbol except: S H	None
CR	Insert the two characters CR if data item is of negative value; insert two blanks or spaces if value is positive.	Any symbol except: A X + - S DB H	Must be last symbol except for P or V
DB	Insert the two characters DB if data item is of negative value; insert two blanks if value is positive.	Any symbol except: A X + - S CR H	Must be last symbol except for P or V
$ (currency sign)	Insert $ sign in character position. If more than one, indicates floating $ sign.	Any symbol except: one $ cannot be used with: A X S H; more than one $ cannot be used with: S H A X * Z or more than one + -	Must be first symbols when more than one except for single + or - PB0 (zero). If only one used, it can only be preceded by + - or P or V
0 (zero)	Insert 0 in character position.	Any symbol except: S H	None
+	Insert + in character position if data item value positive; - if value negative. If more than one +, indicates floating sign.	Any symbol except: one + cannot be used with: A X - S CR DB H; more than one + cannot be used with A X - S CR DB Z H* or more than one $ sign.	If only one +, must be either first or last except for P or V. If more than one +, must be first symbol except for $ sign.
- (minus)	Insert - in character position if data item value negative, blank if positive. If more than one - , indicates floating sign.	Any symbol except: one - cannot be used with: A X + S CR DB H; more than one - cannot be used with: A X + S CR DB * Z H or more than one $ sign.	If only one - , must be either first or last except for P or V. If more than one - , must be first symbol except for $ sign.
H	Computational-usage	S P V 9	None

Table 5–4. PICTURE Symbols

		FIXED INSERTION								OTHER SYMBOLS											
		B	0	,	.	{+ −}	{+ −}	{CR DB}	cs	{A X}	P	P	S	V	{Z *}	{Z *}	9	{+ −}	{+ −}	cs	cs
FIXED INSERTION	B	X	X	X	X	X			X	X	X			X	X	X	X	X	X	X	X
	0	X	X	X	X	X			X	X	X		X	X	X	X	X	X	X	X	X
	,	X	X	X	X	X			X		X			X	X	X	X	X	X	X	X
	.	X	X	X		X			X		X				X		X	X		X	
	{+ −}										X										
	{+ −}	X	X	X	X				X		X			X	X	X	X			X	X
	{CR DB}	X	X	X	X				X		X			X	X	X	X			X	X
	cs					X					X			X							
OTHER SYMBOLS	A X	X	X							X							X				
	P										X		X	X							
	P	X	X	X		X	X	X	X			X	X		X		X	X		X	
	S																				
	V	X	X	X		X			X			X	X		X		X	X		X	
	{Z *}	X	X	X		X			X						X						
	{Z *}	X	X	X	X	X			X		X			X	X	X					
	9	X	X	X	X	X			X	X	X		X	X	X		X	X		X	
	{+ −}	X	X	X					X									X			
	{+ −}	X	X	X	X				X		X			X				X	X		
	cs	X	X	X		X														X	
	cs	X	X	X	X	X					X			X						X	X

NOTES:

1. This chart shows the order of precedence when using characters as symbols in a character-string. An X at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol cs.

2. At least one of the symbols A X Z 9 * or at least two of the symbols + - or cs must be present in a PICTURE string.

3. P, fixed insertion + and - appear twice. The first occurrence represent their use to the left of the PICTURE's numeric character positions and the second their use to the right of the PICTURE's numeric character positions.

4. Z * non-fixed insertion cs + and - appear twice. The first occurrence represents the use before the decimal point position, the second the use after the decimal point position.

Table 5–5. Precedence Rules in PICTURES

SOURCE FIELD		RECEIVING FIELD	
PICTURE	DATA TO BE MOVED	PICTURE	DATA AFTER MOVE
9(5)V99	1234500	ZZ,ZZZ.99	12,345.00
9(5)	00123	ZZ,ZZZ.99	123.00
9(4)V99	123456	$$,$$$.99	$1,234.56
9(4)	0012	$$,$$$.99	$12.00
S9(4)	+1234	$$,$$$.99DB	$1,234.00
S9(4)	-1234	$$,$$$.99DB	$1,234.00DB
S9(4)V99	+001209	$$,$$$.99CR	$12.09
S9999V99	-000123	$$,$$$.99CR	$1.23CR
S9(4)	+1234	++,+++.99	+1,234.00
S9(4)	-0010	--,---.99	-10.00
S999V99	001234	$****.99	$**12.34
9999	1234	990099	120034
9(5)	12345	9B9B9B99	1b2b3b45
X(5)	A1B2C	XBX00XXX	Ab100B2C
A(5)	ABCDE	ABB0AAA0BX	Abb0BCD0bE
9(4)	1234	9(5)	01234
9(5)	12345	999.99	345.00
9V9(5)	123456	9(5).99	00001.23
AA	AB	A(5)	ABbbb
A(5)	ABCDE	AA	AB
99PPP	12	9(5)	12000
VPPP99	12	.9(5)	.00012
V9(5)	12345	Z(5).99	bbbbb.12
V9(5)	12345	9(5).999	00000.123

Table 5–6. Source and Receiving Fields

5.3.5. USAGE

Function:

Specifies the format of a data item in memory.

Format:

```
[USAGE IS] { COMP                }
           { COMPUTATIONAL       }
           { COMP-3              }
           { COMPUTATIONAL-3     }
           { INDEX               }
           { DISPLAY             }
```

Rules:

(1) The USAGE clause can be written at any level. At a group level, it applies to each elementary item in that group. The USAGE clause of an elementary item cannot contradict the USAGE clause stated for the group to which the item belongs. The USAGE clause of an elementary item cannot contradict the PICTURE clause for that item.

(2) The DISPLAY option specifies that the item is stored in character form, one character per byte.

(3) An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which corresponds to the occurrence number of a table element. PICTURE clause must not be present in this instance.

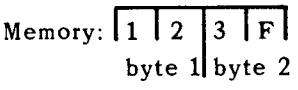
(4) An INDEX data item can be referred to directly only in a SET statement or in a relation condition. Also, an INDEX data item can be part of a group which is referred to in a MOVE or an I/O statement, in which case no conversion will take place.

(5) Except for the level number and data-name necessary for definition, no additional clauses are used to describe INDEX data items.

(6) COMP and COMP-3 specify packed decimal format, where:

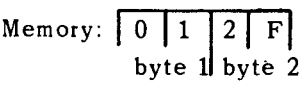
(a) If the number of digits in the item is odd, the object program memory area allocated for this item is an even number of half-bytes.

Example: PIC H999 VALUE 123.



(b) If the number of digits in an item is even, there will be an extra half-byte in the object program memory allocated for this item. The item's PICTURE is unchanged.

Example: PIC H99 VALUE 12.



The compiler ensures that the unused half-byte is always set to zero when information is stored in this item. The compiler assumes that when the item is referenced it will contain a valid packed decimal number, with zero in the leftmost half-byte.

(7) If the USAGE clause is omitted, DISPLAY is assumed unless the PICTURE clause contains an H in its character-string.

5.3.6. SYNCHRONIZED

Function:

To specify the positioning of data items within a computer word or words.

Format:

$$\left\{\begin{matrix}\underline{\text{SYNC}}\\ \underline{\text{SYNCHRONIZED}}\end{matrix}\right\} \left[\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right]$$

Rule:

The SYNCHRONIZED clause has no effect on the object program in UNIVAC 9400, however, it is acceptable to the compiler for compatibility purposes.

5.3.7. JUSTIFIED

Function:

To specify nonstandard positioning of data within a receiving data item.

Format:

$$\left\{\begin{matrix}\underline{\text{JUST}}\\ \underline{\text{JUSTIFIED}}\end{matrix}\right\} \text{RIGHT}$$

Rules:

(1) JUSTIFIED may be specified only at the elementary item level.

(2) This clause may not be used for numeric or numeric-edited data, since numeric data is aligned by its decimal point, when present, or right JUSTIFIED when not present.

(3) Alphabetic, alphanumeric, and alphanumeric-edited data is left JUSTIFIED with space fill when the JUSTIFIED clause is not specified.

(4) When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger, the leftmost characters are truncated. When the receiving data item is JUSTIFIED and larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill.

5.3.8. VALUE IS

Function:

To define the initial VALUE of a Working-Storage item, or to specify the values associated with a condition-name.

Formats:

Format 1:

VALUE IS *literal*

Format 2:

$$\left\{\begin{matrix}\text{VALUE IS}\\ \text{VALUES ARE}\end{matrix}\right\} \textit{literal-1}\ [\text{THRU}\ \textit{literal-2}]$$

$$\left[, \textit{literal-3}\ [\text{THRU}\ \textit{literal-4}]\right]\ldots$$

Rules:

(1) Format 1 is used to specify the initial VALUE of a data item in the Working-Storage Section. The following rules apply to format 1:

(a) This option causes the item to assume the specified VALUE at the start of the object program. If the VALUE clause is not used in an item description, the initial value may be unpredictable.

(b) The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item.

(c) In the File Section and the Linkage Section, the VALUE clause must not be used except for condition-name entries.

(d) The VALUE clause cannot be used in a Record Description entry containing a REDEFINES clause or in an entry subordinate to an entry containing a REDEFINES clause.

(e) The VALUE clause must not be stated in a Record Description entry containing an OCCURS clause or in an entry subordinate to an entry containing an OCCURS clause.

(f) The VALUE clause must not be specified for a group item containing items with descriptions including JUST, SYNC, USAGE COMP, or USAGE INDEX.

(g) If the VALUE clause is used in an entry at the group level, literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause must not be stated at the subordinate levels within the group.

(2) Format 2 can be used only in a condition-name entry, and is available only in the extended compiler. Format 1 is used for condition-names when compiling with the basic tape compiler. The following rules apply to format 2:

(a) All condition-name entries are level-number 88. See 5.3.12 for a full description of condition-name.

(b) When the THRU option is used, literal-1 must be less than literal-2, literal-3 less than literal-4, and so on.

(3) In the File Section, only the VALUE clause(s) stated for condition-name entries is valid.

(4) A figurative constant may be substituted in either format 1 or format 2 when a literal is specified.

(5) During compilation, a diagnostic is issued when the VALUE and PICTURE clauses conflict in any manner. Compilation continues with the VALUE clause ignored. (Exception: unsigned VALUE with signed PICTURE gets a + sign.)

5.3.9. BLANK WHEN ZERO

Function:

Causes the value of a receiving item to be set to space when the value of the sending item is zero.

Format:

BLANK WHEN ZERO

Rules:

(1) This clause can be specified only at the elementary item level, and can be used only with a numeric or numeric-edited item. When used with a numeric item, the category of the item is considered numeric-edited.

(2) The effect is not necessarily the same as zero suppression editing via the PICTURE clause, because the item is affected only when its numeric value is zero.

5.3.10. MAP

Function:

Specifies the memory size of a data item in bytes.

Format:

MAP IS *integer*, CHARACTERS

Rule:

The MAP clause has no effect on the object program in the UNIVAC 9400 System; however, it is acceptable to the compiler for compatibility purposes.

5.3.11. RENAMES

Function:

Permits alternate, possibly overlapping, groupings of elementary items.

Format:

66 *unqualified-data-name-1* ;RENAMES *data-name-2* [THRU *data-name-3*]

Rules:

(1) All RENAMES entries associated with a given logical record must immediately follow its last data description entry.

(2) Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the associated logical record, and cannot be the same data-name.

(3) Level-numbers 66, 77, 88, and 01 cannot be RENAMEd.

(4) Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description, nor can it be subordinate to an entry with an OCCURS clause.

(5) Data-name-2 must precede data-name-3 in the Record Description.

(6) Data-name-3 cannot be subordinate to data-name-2.

(7) Data-name-2 and data-name-3 may be qualified.

(8) One or more RENAMES entries can be written for a logical record.

5.3.12. Condition-name

Function:

To assign a name for a specific value or range of values. The condition-name with multiple values, including the THRU option, is implemented only in the extended compiler.

Format:

88 *condition-name* ; {VALUE IS / VALUES ARE} *literal-1* [THRU *literal-2*]

[,*literal-3* [THRU *literal-4*]] . . .

Rules:

(1) The VALUE clause is used as described in 5.3.8.

(2) Each condition-name requires a separate entry with a separate level-number 88.

(3) The condition-name entries for a particular conditional-variable must immediately follow the entry describing the conditional-variable item with which the condition-name is associated.

(4) A condition-name may be associated with any group or elementary item except a level-number 66 item, or an index data item.

(5) Examples of use of condition-name:

(a) Elementary item:

```
02 data-name-1.
   03 data-name-2 PIC XX.
   88 condition-name VALUE 'AB'.
02 data-name-3.

PROCEDURE DIVISION.

IF condition-name GO TO procedure-name.
   Instead of:
IF data-name-2 = 'AB' GO TO procedure-name.
```

(b) Group Item:

```
02 data-name-1.
88 condition-name  VALUE is '20' THRU '25'.
   03 data-name-2  PIC 9.
   03 data-name-3  PIC 9.
02 data-name-4.
PROCEDURE DIVISION.

IF condition-name GO TO procedure-name.
   Instead of:

IF data-name-1 NOT < '20' AND NOT > '25' GO TO
procedure-name.
```

5.3.13. SIGN

Function:

Specifies the position and the mode of representation of the optional sign when it is necessary to describe these properties explicitly.

Formats:

Format 1:

[SIGN IS] {LEADING / TRAILING} SEPARATE CHARACTER

Format 2:

[SIGN IS] TRAILING

Rules:

(1) The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

(2) The numeric data description entries to which the SIGN clause applies must be described, either explicitly or implicitly, as USAGE IS DISPLAY.

(3) At most, one SIGN clause may apply to any given numeric data description entry.

(4) If format 1 is used, the letter 'S' in the PICTURE is counted in determining the size of the item. The operational signs for positive and negative are the characters '+' and '−' respectively.

(5) If the optional SEPARATE CHARACTER clause is not present, the letter 'S' in the PICTURE is not counted in determining the size of the item. Format 2 specifies that the operational sign is in the zone portion of the least significant digit position of the item. A positive sign is represented by a hexadecimal 'C', a negative sign by a hexadecimal 'D'.

(6) A numeric data item whose PICTURE contains the character 'S', but to which no optional SIGN clause applies, has an operational sign in the zone portion of the least significant digit position. The sign representation is as described for format 2 of the SIGN clause.

5.4. WORKING-STORAGE SECTION

Definition:

That section of the Data Division used to describe areas of memory which are to contain intermediate results of processing and other temporarily stored data at object program run time, as well as named constants.

Format:

WORKING-STORAGE SECTION.

[77 *(name and description of single-item area)*
.
.
.]

[01 *(record-name and description)*
  03 *(name and description of group item)*
  .
  .
  .
  nn *(name and description of elementary item)*]

[66 *data-name* ;RENAMES *data-name-n*]

[01 *(record-name and description)*
.
.
.]

[88 *(condition-name entry)*]

5.4.1. Independent Entries

Function:

To describe noncontiguous single items in Working-Storage, each of which is neither subdivided nor a subdivision of another data-name.

Format:

77 *unqualified-data-name* ; { PIC / PICTURE } IS *picture-string*

Rules:

(1) Level-number 77 is assigned only to single item areas.

(2) Each independent entry must have a unique data-name.

(3) All level-number 77 entries should be grouped together in the beginning of the Working-Storage Section.

(4) The VALUE clause may be used to specify the initial or constant value of any level-number 77 entry.

5.4.2. Record Description

Function:

To describe contiguous data areas which are not part of a file.

Format:

01 *record-name*

(subordinate data items and clauses)

Rules:

(1) Data elements in Working-Storage which bear a definite relationship to each other may be grouped into records. This is done through the same descriptive clauses used in Data Description entries in the File Section, including the OCCURS and REDEFINES clauses.

(2) Each record-name must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be qualified.

5.5. LINKAGE SECTION

Definition:

That section of the Data Division used to describe data that is available in a CALLing program, but is referenced in both the CALLing and the CALLed programs.

Rules:

(1) Organization and structure follow the rules described under the Working-Storage Section, with one exception: the VALUE clause may not be specified for other than 88 level entries.

(2) Record Description entries in the LINKAGE SECTION provide names and descriptions, but storage within the program is not reserved since the data exists elsewhere.

(3) The LINKAGE SECTION is required in any program containing an ENTRY statement with a USING option or Procedure Division USING.

(4) See Appendix F for examples of CALLing and CALLed programs.

6. PROCEDURE DIVISION

6.1. GENERAL

The Procedure Division contains the instructions or steps necessary to solve a given problem.

Format:

```
PROCEDURE DIVISION [USING {data-name}. . . ].
[DECLARATIVES.
 .
 . (declarative sections)
 .
END DECLARATIVES.]
[section-name-1 SECTION [priority-number] . ]

 paragraph-name-1.
      sentence-1.
     [.
      .
      .
      sentence-n.]
[paragraph-name-2.
 .
 .
 .]
[section-name-n SECTION [priority-number] . ]
```

6.1.1. USING

Function:

USING immediately following the header PROCEDURE DIVISION serves as an entry point declaration and can appear only in a CALLed subprogram.

Format:

USING *unqualified-data-name-1* [*unqualified-data-name-2*]. . .

Rules:

(1) If USING is present, the external symbol (ENTRY name) associated with this entry point is the same as PROGRAM-ID.

(2) If USING is not present, the beginning of the PROCEDURE DIVISION is not one of the entry points in this particular subprogram.

(3) Data-names present refer to data items described in this subprogram. Their level numbers are restricted to 01 or 77, and they must be defined in the Linkage Section.

6.2. DECLARATIVES

Function:

The Declaratives Section of the Procedure Division contains compiler-directing statements that specify the circumstances under which a procedure is to be executed.

Format:

DECLARATIVES.
section-name-1 SECTION.
  declarative-sentence-1.
paragraph-name-1.
  sentence-1.
  [*sentence-n.*]
[*section-name-n* SECTION.]. . .
.
.
.
END DECLARATIVES.

Rules:

(1) Declarative Sections are grouped at the beginning of the Procedure Division.

(2) The key word DECLARATIVES must immediately follow the header PROCEDURE DIVISION on a separate line. The key words END DECLARATIVES must follow the last line of the declaratives on a separate line.

(3) Each Declarative Section must begin with a section-name, followed by a USE sentence. The remainder of the section consists of one or more procedural paragraphs.

(4) No priority number is allowed on section-names in the Declaratives.

6.3. SECTIONS

Definition:

The most inclusive procedural unit in the Procedure Division to which a procedure name can be assigned.

Format:

section-name SECTION [*priority-number*].

paragraph-name.

.

.

.

Rules:

(1) The Procedure Division must be divided into sections with appropriate priority numbers when the program is to be segmented or when the Declarative Section is present.

(2) Priority-number must be an unsigned integer ranging in value from 0 through 99.

(3) Section priority-numbers must be in ascending sequence, and sections with the same priority-number must be contiguous.

(4) Sections belonging to the Declaratives portion of the Procedure Division are associated with the fixed segment, and must not contain priority numbers in their section headings.

(5) Priority numbers 0 through 49 are used for the fixed segment, and priority numbers 50 through 99 designate independent segments. (See Section 7 for a complete discussion on segmentation.)

(6) Sections comprising the fixed segment, if any, must precede all sections with priority numbers greater than 49.

6.4. PARAGRAPHS

Definition:

A body of one or more procedural sentences with a procedure name by which it may be identified and referenced.

Format:

paragraph-name.

sentence-1.

[*sentence-2.*]

.

.

.

Rules:

(1) A paragraph must contain at least one sentence, and may consist of any practical number of sentences. It must be headed by an identifying procedure name, since transfer references within the Procedure Division are made to entire paragraphs.

(2) Any practical number of paragraphs may be combined into a section.

(3) Generally, the object coding for a single sentence must be less than 4096 bytes.

6.5. STATEMENTS AND SENTENCES

Definition:

A statement consists of a verb and any other reserved words and user-supplied words necessary to fulfill one of the valid verb formats.

A sentence consists of one or more statements terminated by a period.

Format:

statement-1 [{*statement-2*} . . .].

6.5.1. Imperative Statements

Definition:

Statements which indicate specific unconditional actions to be taken by the object program.

Format:

verb word-string.

Rules:

(1) Verb must be one of the allowable UNIVAC 9400 COBOL verbs listed in 6.6.

(2) Word-string consists of all words (reserved words, names, literals, and punctuation) necessary to complete a valid format for that verb.

6.5.2. Conditional Statements

Definition:

Statements which begin with the verb IF and specify alternative courses of action, depending upon the outcome of a test or comparison.

The AT END conditional is discussed with the READ and RETURN verbs; the ON SIZE ERROR conditional is discussed with the arithmetic verbs; and the INVALID KEY conditional is discussed with the READ, WRITE, REWRITE, and INSERT statements.

Format:

IF *condition;* { *statement-1* / NEXT SENTENCE } [; { ELSE / OTHERWISE* } { *statement-2* / NEXT SENTENCE }]

Rules:

(1) Condition can be simple or compound. A simple conditional expression would contain only one of the following: a relational expression, condition-name, or item class test. A compound conditional expression would contain more than one condition. Compound conditions are not permitted in the basic compiler.

(2) All other rules for format and application of conditional and compound conditional statements are discussed under the IF and PERFORM verbs.

6.5.3. Compiler-Directing Statements

Definition:

Statements which direct the compiler to take certain actions at compilation time.

Format:

verb word-string.

Rules:

(1) All rules for compiler-directing statements are stated in the discussion of the following verbs:

COPY, ENTER, NOTE, USE

(2) Word-string consists of all reserved words and user-supplied words necessary to complete a valid format for that verb.

(3) Compiler-directing statements must not appear within conditional statements.

**OTHERWISE is an extension to American National Standard COBOL.*

6.6. VERB TYPES

Definition:

A verb is a reserved word used in the Procedure Division that denotes action to be performed by the computer or the compiler. There are seven general categories of verbs in UNIVAC 9400 COBOL. These categories, and the verbs in each, are as follows (*denotes extended compiler only):

- Arithmetic: ADD, DIVIDE, MULTIPLY, SUBTRACT, COMPUTE*
- Procedure Branching: ALTER, GO TO, PERFORM, EXIT
- Data Movement: EXAMINE, MOVE, SET, TRANSFORM
- Input-Output: ACCEPT, CLOSE, DISPLAY, INSERT*, OPEN, READ, RELEASE*, RETURN*, REWRITE*, SEEK*, SORT*, WRITE
- Ending: STOP
- Conditional: IF, SEARCH
- Compiler Directing: COPY, ENTER, NOTE, USE

A description of the various categories and the verbs contained in each, is presented in the ensuing paragraphs.

6.6.1. Arithmetic Verbs

The arithmetic verbs permit basic calculations to be performed on the data. Four verbs corresponding to the four basic arithmetic operations are provided: ADD, SUBTRACT, MULTIPLY, and DIVIDE. In the extended compiler only, a fifth verb, COMPUTE, is provided to allow the programmer to specify arithmetic calculations through the use of arithmetic expressions.

Appendix H describes the manner in which intermediate results of arithmetic operations are handled by the compiler.

Rules:

(1) All data items referenced in arithmetic statements must represent numeric elementary data items previously defined in the Data Division.

(2) All literals used in arithmetic statements must be numeric.

(3) The maximum size of each operand is 18 decimal digits. The composite of operands (the data item resulting from the superimposition of all operands, aligned by decimal points) must not contain more than 18 digits.

(4) The data descriptions (PICTURE) of the operands may differ from each other. Decimal point alignment is supplied automatically throughout computations. Conversion of items with unlike usage is also automatic.

(5) The PICTURE of any arithmetic operand cannot contain editing symbols. An appropriate error diagnostic will be issued during compilation if this occurs. Operational signs and implied decimal points are not considered editing symbols. When the GIVING option is used, data items to the right of the word GIVING may contain editing symbols.

(6) If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant identifier, truncation is relative to the size provided for the resultant identifier. When the ROUNDED option is used, the absolute value of the resultant identifier is increased by one whenever the most significant digit of the excess is equal to or greater than five.

(7) If, after decimal point alignment, the value of result exceeds the largest value that can be contained in the associated resultant identifier, a size error condition exists. In the event of a size error condition, one of two possibilities will occur, depending on whether the ON SIZE ERROR option has been specified.

(a) In the event that ON SIZE ERROR is not specified, and a size error condition arises, the effect is unpredictable.

(b) If ON SIZE ERROR has been specified, and a size error condition arises, the value of the resultant identifier will not be altered. The imperative-statement associated with the ON SIZE ERROR option will be executed after the last resultant identifier is considered.

(8) The CORRESPONDING option may be used with the ADD and SUBTRACT verbs. In the following paragraphs, d_1 and d_2 refer to the group items involved. A pair of data items, one from each group item, CORRESPOND if the following conditions exist:

(a) A data item in d_1 and a data item in d_2 have the same name and qualification up to, but not including, d_1 and d_2.

(b) Both of the data items are elementary numeric items.

(c) Neither d_1 nor d_2 can be a data item with level number 66, 77, or 88.

(d) A data item subordinate to d_1 or d_2 and containing a RENAMES, REDEFINES, or OCCURS clause is ignored. However, d_1 and d_2 may have REDEFINES or OCCURS clauses, or be subordinate to data items with REDEFINES or OCCURS clauses.

(9) Statements having multiple results are considered by the compiler as though they were written in the following manner:

(a) As a statement which performs all the arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

(b) As a sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to have been written in the same left to right sequence in which the multiple results are listed. For example, the result of the statement

```
ADD A, B, C TO C, D(C), E
```

is equivalent to

```
ADD  A, B, C  GIVING  temp

ADD  temp  TO  C

ADD  temp  TO  D(C)

ADD  temp  TO  E
```

where: temp is an intermediate result item.

6.6.2. Procedure Branching Verbs

Normally,. the statements in the Procedure Division are executed consecutively in order of their appearance. This is also true of the execution of each paragraph and section. However, it is often necessary to alter this normal sequence of operation and branch to a different point in the program to execute a number of statements before returning to the next statement. The procedure branching verbs permit this sequencing of logical operations:

ALTER, GO TO, PERFORM, EXIT

6.6.3. Data Movement Verbs

Three verbs are provided by UNIVAC 9400 COBOL for the specific purpose of moving or manipulating data:

EXAMINE, MOVE, SET

This is in addition to the several verbs which have, as a secondary function, the ability to move or manipulate data in some manner. For example, an arithmetic verb may cause some data movement and/or manipulation. This, however, is secondary to its main function of effecting an arithmetic calculation.

6.6.4. Input/Output Verbs

In any data processing application, quantities of data passed between storage and external media such as card, tape, or disc devices. The input/output verbs control and coordinate the flow of data, enabling the COBOL programmer to obtain records for processing and return the processed record to the external media. The input/output verbs are:

ACCEPT, CLOSE, DISPLAY, OPEN, READ, WRITE

In addition to these verbs, the extended compiler contains the

INSERT, RELEASE, RETURN, REWRITE, SEEK, and SORT verbs.

6.6.5. Ending Verb

The STOP verb is used to halt execution of the object program either permanently or temporarily.

6.6.6. Conditional Verb

Conditional expressions are used in situations where the outcome of a test will determine the next logical step to be performed. The verb IF is the principal conditional verb used with conditional expressions to determine the truth or falsity of a statement or comparison. (See also PERFORM statement, 6.7.21.)

6.6.7. Compiler-Directing Verbs

Certain verbs direct the compiler to perform a specific action and do not directly cause any object coding to be produced. These verbs affect the object program indirectly, except for the verb NOTE which has absolutely no effect on the object program.

The compiler-directing verbs are:

COPY, ENTER, NOTE, USE.

6.7. VERBS

Each UNIVAC 9400 COBOL verb is described in alphabetical order in this section.

6.7.1. ACCEPT

Function:

Causes low volume data to be read from an appropriate hardware device, system memory location, or UPSI switch (user program switch indicator).

Format:

ACCEPT *identifier* [FROM *mnemonic-name*]

Rules:

(1) The ACCEPT statement causes the next set of data available at the mnemonic-name to replace the contents of the data item named by the identifier.

(2) The job stream is assumed to be the input source when the FROM option is not specified. The description of identifier determine the number of cards ACCEPTed. One card from the job stream contains up to 80 characters. The maximum length specified by identifier is 4095 characters, which would require 52 cards.

(3) To indicate that input is to be accepted from the console typewriter, the following message is displayed:

CA10 ACCEPT READY.

Program operation is suspended until a typein occurs (CA10 indicates a COBOL ACCEPT).

(4) The mnemonic-name must be associated with an implementor-name in the SPECIAL-NAMES paragraph of the Environment Division. SPECIAL-NAMES that can be the source of ACCEPTed data are:

SYSCOM

SYSDATE

SYSTIME

SYSCONSOLE

SYSSWCH

See Table 4–1 for specific interpretation of implementor-names.

6.7.2. ADD

Function:

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

Formats:

Format 1:

ADD { *identifier-1* / *literal-1* } [, *identifier-2* / , *literal-2*] . . . TO *identifier-m* [ROUNDED]

[, *identifier-n* [ROUNDED]] . . . [; ON SIZE ERROR *imperative-statement*]

Format 2:

ADD { *identifier-1* / *literal-1* }, { *identifier-2* / *literal-2* } [, *identifier-3* / , *literal-3*] . . . GIVING

identifier-n [ROUNDED] [; ON SIZE ERROR *imperative-statement*]

Format 3:

ADD { CORR / CORRESPONDING } *identifier-1* TO *identifier-2* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Rules:

(1) In formats 1 and 2 each identifier must refer to an elementary numeric item except for identifiers to the right of the word GIVING which may contain editing symbols.

(2) Each literal must be a numeric literal.

(3) The maximum size of each operand is 18 decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data items that follow the word GIVING, aligned on their decimal points, must not contain more than 18 digits.

(4) If format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value in each identifier, identifier-m, identifier-n, . . . , and the result is stored in each resultant identifier, identifier-m, identifier-n, . . . , respectively.

(5) If format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-n, which is the resultant identifier.

(6) If format 3 is used, data items in identifier-1 are added to, and stored in, corresponding data items in identifier-2.

(7) For a description of the ROUNDED, SIZE ERROR, and CORRESPONDING options, see paragraph 6.6.1, Arithmetic Verbs.

6.7.3. ALTER

Function:

The ALTER statement modifies a predetermined sequence of operations.

Format:

ALTER *procedure-name-1* TO [PROCEED TO] *procedure-name-2*

[, *procedure-name-3* TO [PROCEED TO] *procedure-name-4*]. . .

Rules:

(1) Procedure-name-1, procedure-name-3, . . . is the name of a paragraph that contains only one sentence consisting of a GO TO statement without the DEPENDING ON option.

(2) Procedure-name-2, procedure-name-4, . . . is the name of a paragraph or section in the Procedure Division.

(3) During execution of the object program, the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, . . . replacing the object of the GO TO by procedure-name-2, procedure-name-4, . . . , respectively.

(4) A GO TO statement in a section whose priority is equal to or greater than 50 must not be referred to by an ALTER statement in a section with a different priority.

6.7.4. CALL

Function:

Used in conjunction with the ENTER verb in the main program to communicate with subprogram entry points.

Format:

CALL *entry-name* [USING { *file-name* / *data-name* / *procedure-name* / *sort-name* } ...]

Rule:

See the ENTER verb, paragraph 6.7.10 for information regarding use of CALL.

6.7.5. CLOSE

Function:

Terminates the processing of one or more input or output reels, units, or files with optional rewind with or without lock.

Format:

CLOSE *file-name-1* [REEL / UNIT] [WITH { NO REWIND / LOCK }]

[, *file-name-2* [REEL / UNIT] [WITH { NO REWIND / LOCK }]] ...

Rules:

(1) File-name must not be the name of a SORT file.

(2) After a CLOSE statement without a REEL/UNIT phrase has been executed for a file, an OPEN statement must be executed before any other reference can be made to the file.

(3) The REEL/UNIT option is used to effect reel or unit swapping in a sequential file process. If the reel/unit is to be dismounted from the device, the LOCK option should be employed.

(4) The UNIT option is only applicable for direct access files when ACCESS MODE IS SEQUENTIAL is specified.

(5) The REEL, NO REWIND, and LOCK options are only applicable to magnetic tape files and are meaningless when operating with any other device.

(6) When the LOCK option is specified for REEL, the current reel of the tape file is rewound and unloaded. When the LOCK option is specified for the entire file, the file may not be OPENed again in the object program.

(7) Each file-name refers to an FD name in the Data Division.

(8) If neither LOCK nor NO REWIND is specified, the current reel of the file is rewound and all other reels belonging to the file are rewound. However, this rule does not apply to those reels controlled by a prior CLOSE REEL entry.

(9) If the NO REWIND option is specified, the current reel of the file remains in whatever position it is in at the time the CLOSE is given.

6.7.6. COMPUTE

Function:

The COMPUTE statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression. COMPUTE is available only with the extended compiler.

Format:

$$\underline{\text{COMPUTE}}\ \textit{identifier-1}\ [\underline{\text{ROUNDED}}] = \left\{ \begin{matrix} \textit{identifier-2} \\ \textit{literal} \\ \textit{arithmetic-expression} \end{matrix} \right\}$$

[; ON SIZE ERROR *imperative-statement*]

Rules:

(1) Literal must be a numeric literal.

(2) Each identifier must refer to an elementary numeric item.

Only identifier-1 may contain editing symbols.

(3) The arithmetic-expression option permits the use of any meaningful combination of identifiers, numeric literals, and arithmetic operators, parenthesized as required.

(4) The maximum size of each operand is 18 decimal digits.

(5) The identifier-2 and literal options provide a method for setting the value of identifier-1 equal to the value of identifier-2 or literal.

(6) The final result of operations evaluated in the arithmetic-expression is placed in identifier-1.

(7) The arithmetic-expression option allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements.

(8) Intermediate results are possible in a COMPUTE statement containing multiple operands. The compiler treats a statement as a succession of operations, and reserves memory areas for required intermediate results. The compiler also determines the number of integer and decimal places reserved for intermediate results. The ON SIZE ERROR option applies only to final results. See Appendix I for a discussion of how the compiler handles intermediate results.

(a) Arithmetic operators character representation:

Addition	+
Subtraction	–
Multiplication	*
Division	/
Exponentiation	**

(b) Parentheses may be used to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first; within a nest of parentheses, evaluation proceeds from the least inclusive set to the most inclusive set.

(c) When parentheses are not used or parenthesized expressions are at the same level of inclusiveness, the following order of evaluation is implied:

unary + and – signs

**

* and /

+ and –

6.7.7. COPY

Function:

The COPY statement permits copying text from the COBOL library into the source program. The extended compiler provides the facility for copying text from the library into the source program with a capability for word substitution as text is copied. See Format 2.

Format 1:

COPY *library-name*

Format 2:

COPY *library-name* [REPLACING *word-1* BY { *word-2* / *identifier-1* / *literal-1* }
[, *word-3* BY { *word-4* / *identifier-2* / *literal-2* }] ...].

Rules:

(1) The COPY statement may appear as follows:

(a) in any of the paragraphs in the Environment Division;

(b) in any level indicator entry (FD, SD) or an 01 level number entry in the Data Division;

(c) in a section or a paragraph in the Procedure Division.

(2) The library-name may be composed of no more than eight alphanumeric characters and a hyphen; the name must contain at least one alphabetic character.

(3) No other statement or clause may appear in the same entry as the COPY statement.

(4) The copying process is terminated by the end of the library text.

(5) Both the COPY statement and the statements of the library text to which it refers, will appear on the output listing.

(6) The text contained on the library must not contain any COPY statements.

(7) If the REPLACING phrase is used, each occurrence of word-1, word-3, etc., in the text being copied from the library is replaced by the word, identifier, or literal associated with it in the REPLACING phrase.

(8) Use of the REPLACING option does not alter the material as it appears on the library.

(9) Word-1, word-2, etc., may be a data-name, procedure-name, condition-name, mnemonic-name, file-name or sort-name.

(10) The literal-n may be numeric or nonnumeric, or any figurative constant except the ALL literal.

6.7.8. DISPLAY

Function:

The DISPLAY statement causes low volume data to be written to an appropriate hardware device or system memory location. It can also be used to set the UPSI switches. (Refer to Appendix L for a detailed explanation of DISPLAY statement usage.)

Format:

DISPLAY { *literal-1* / *identifier-1* } [, *literal-2* / , *identifier-2*] . . . [UPON *mnemonic-name*]

Rules:

(1) When the UPON option is omitted, the data is written on the console typewriter.

(2) When the UPON option is specified, the mnemonic-name must be associated with an implementor-name in the SPECIAL-NAMES paragraph (4.2.3) in the Environment Division.

(3) Those SPECIAL-NAMES which may be associated with the DISPLAY statement via mnemonic-name are:

SYSCOM, SYSCONSOLE, SYSSWCH, SYSSWCH-n, SYSLST. See Table 4–1 for more detailed information.

(4) Console DISPLAYs are in 59-character multiples; each multiple is followed by a hyphen and preceded by CD10 which indicates a COBOL DISPLAY (see Appendix L). For signed numeric items, a separate sign character is displayed immediately following the operand.

(5) The number of printer characters displayed are in multiples of 132. An advance of one line precedes each line of output. Each operand displayed is limited to 4092 characters. For signed numeric items, a separate sign character is displayed immediately following the operand.

6.7.9. DIVIDE

Function:

The DIVIDE statement divides one numeric data item into another and sets the value of a data item equal to the results.

Formats:

Format 1:

DIVIDE { *identifier-1* / *literal* } INTO *identifier-2* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Format 2:

DIVIDE { *identifier-1* / *literal-1* } INTO { *identifier-2* / *literal-2* } GIVING *identifier-3* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Format 3:

DIVIDE { *identifier-1* / *literal-1* } BY { *identifier-2* / *literal-2* } GIVING *identifier-3* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Format 4:

DIVIDE { *identifier-1* / *literal-1* } INTO { *identifier-2* / *literal-2* } GIVING *identifier-3* [ROUNDED]

REMAINDER *identifier-4* [; ON SIZE ERROR *imperative-statement*]

Format 5:

DIVIDE { *identifier-1* / *literal-1* } BY { *identifier-2* / *literal-2* } GIVING *identifier-3* [ROUNDED]

REMAINDER *identifier-4* [; ON SIZE ERROR *imperative-statement*]

Rules:

(1) Each identifier must refer to a numeric elementary item except in formats 2 and 3, where identifiers to the right of the word GIVING may contain editing symbols.

(2) Each literal must be a numeric literal.

(3) The maximum size of each operand is 18 decimal digits.

(4) When format 1 is used, the resulting quotient replaces identifier-2.

(5) When either format 2 or 3 is used, the result is stored in identifier-3.

(6) For a description of the ROUNDED and SIZE ERROR options, see paragraph 6.6.1, Arithmetic Verbs.

(7) The composite of operands, which is the data item resulting from the superimposition of all receiving data items aligned on their decimal points, must not contain more than 18 digits.

(8) Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. A remainder in COBOL is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If the ROUNDED option is specified, the quotient is rounded after the remainder is determined.

6.7.10. ENTER

Function:

The ENTER statement is used in conjunction with the CALL or ENTRY statements to permit run-time communications between the main COBOL program and COBOL or any other language subprograms. ENTER may also be used with the EXIT PROGRAM or RETURN options.

Formats:

Format 1:

ENTER LINKAGE.

CALL *entry-name* [USING { *data-name* / *procedure-name* / *file-name* / *sort-name* } ...] .

ENTER COBOL.

Format 2:

ENTER LINKAGE.

ENTRY *entry-name* [USING { *unqualified data-name* } ...] .

ENTER COBOL.

Format 3:

ENTER LINKAGE.

{ EXIT PROGRAM. / RETURN. }

ENTER COBOL.

Rules:

(1) Format 1 causes control to be transferred from one object program to another within the run unit.

(a) Entry-name must be the external symbol of an entry point in the subprogram being CALLed.

(b) Each of the identifiers in the USING clause of the CALL statement must be a reference to a 77- or 01-level data item in the File, Working-Storage, or Linkage Section of the CALLing program.

(c) Procedure-name, file-name, and sort-name can be used only if the CALLed subprogram is written in a language other than COBOL.

(d) If the subprogram is written in COBOL, there are two ways to CALL the subprogram, depending on the entry point of the subprogram:

- If the entry point is the beginning of the Procedure Division (USING after the division header), then entry-name in format 1 must be the same as the PROGRAM-ID of the CALLed subprogram.
- If the entry point in the subprogram is designated by the ENTRY statement (format 2), then the entry-name in format 1 must be the same as the entry-name in format 2.

(e) If the CALLed program is written in assembler language, entry points are labels specified by assembler directive ENTRY or labels of START and CSECT assembler directives.

(2) Format 2 is used in the CALLed subprogram to designate an ENTRY point; it may not appear in the Declaratives portion.

(a) If the CALLing program is written in COBOL, entry-name in format 2 must be the same as entry-name in format 1.

(b) Data-name can be neither qualified nor subscripted.

(c) Data-names are the names of 01- or 77-level data items specified in the Linkage Section of this particular subprogram.

(d) The sequence of appearance of the operands in the two USING clauses is extremely significant since corresponding operands refer to a single common data item, i.e., correspondence is by position and not by name. Each reference to an operand in the CALLed program's USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the CALLing program.

(3) Format 3 is used in the CALLed subprogram to return control to the CALLing program.

(a) All COBOL subprograms must contain this clause.

(b) Control returns to the point in the CALLing program immediately following the CALL statement.

(c) The EXIT PROGRAM or RETURN options are equivalent. RETURN is included for compatibility with other COBOL implementations.

(4) See Appendix F for sample CALLing and CALLed programs.

6.7.11. ENTRY

Function:

The ENTRY statement is used in conjunction with the ENTER statement in a CALLED program to establish an entry point.

Format:

→ ENTRY *entry-name* [USING *unqualified data-name* . . .].

Rules:

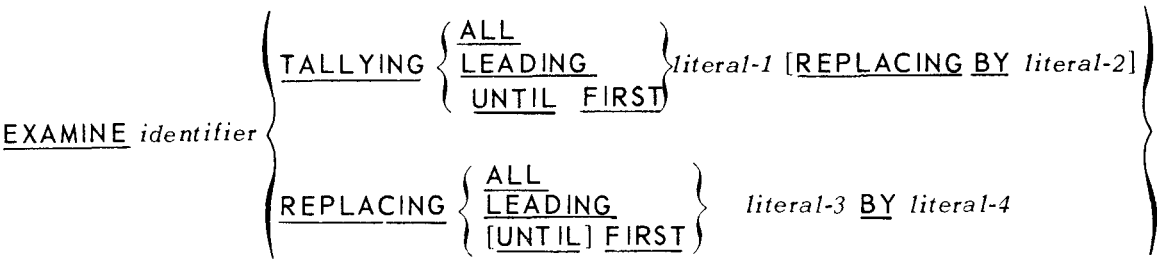
See 6.7.10.

6.7.12. EXAMINE

Function:

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

Format:



Rules:

(1) The description of the identifier must be such that USAGE IS DISPLAY (explicitly or implicitly).

(2) Each literal must consist of a single character belonging to a class consistent with that of identifier. A literal may be any figurative constant except ALL.

(3) Examination of identifier proceeds as follows:

(a) Nonnumeric: Examination starts at the leftmost character and proceeds to the right; each character is examined individually.

(b) Numeric: Examination starts at the leftmost character and proceeds to the right. Each character except the sign (which is ignored) is examined individually.

(4) The count derived as a result of the TALLYING option is placed in a special register called TALLY. Depending upon which option is selected, the count represents the following:

(a) ALL option: The number of occurrences of literal-1.

(b) LEADING option: The number of occurrences of literal-1 prior to encountering a character other than literal-1.

(c) UNTIL FIRST option: The number of occurrences of characters not equal to literal-1 encountered before the first occurrence of literal-1.

(5) When either of the REPLACING options is used, the replacement rules are as follows:

(a) ALL option: Literal-2 or literal-4 substituted for each occurrence of literal-1 or literal-3.

(b) LEADING option: The substitution of literal-2 or literal-4 terminates as soon as a character, other than literal-1 or literal-3, is encountered.

(c) UNTIL FIRST option: The substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 is encountered.

(d) FIRST option: The first occurrence of literal-1 or literal-3 is replaced by literal-2 or literal-4.

6.7.13. EXIT

Function:

The EXIT statement provides a common end point for a series of procedures, or marks the logical end of a called program.

Format:

EXIT [PROGRAM]

Rules:

(1) The EXIT statement must be preceded by a paragraph-name and be the only sentence in the paragraph. The EXIT statement must appear in a sentence by itself.

(2) The point to which control is transferred may be at the end of a range of procedures governed by a PERFORM or at the end of a Declarative Section. The EXIT statement is provided to enable a procedure-name to be associated with such a point.

(3) If control reaches an EXIT statement without the optional word PROGRAM, and no associated PERFORM or USE statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

(4) If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

6.7.14. GO TO

Function:

The GO TO statement causes control to be transferred from one part of the Procedure Division to another. GO TO (format 3) is used as a special exit from a USE procedure.

Formats:

Format 1:

GO TO [*procedure-name*]

Format 2:

GO TO *procedure-name-1* [, *procedure-name-2*] . . . , *procedure-name-n*

DEPENDING ON *identifier*

Format 3:

GO TO MORE-LABELS

Rules:

(1) Each procedure-name is the name of a paragraph or section in the Procedure Division of the program.

(2) Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

(3) When format 1 is used, control is transferred to procedure-name or to another procedure-name if the GO TO statement has been affected by an ALTER statement.

(4) If procedure-name is omitted in format 1, an ALTER statement referring to this GO TO statement must be executed prior to execution of this GO TO statement.

(5) In order for a GO TO statement to be ALTERable, it must be the only statement in a paragraph. Only format 1 may be ALTERed.

(6) When a GO TO statement is ALTERed, control is transferred to the new procedure-name each time the GO TO statement is executed, until the GO TO statement is ALTERed again with a different procedure-name.

(7) When format 2 is used, control is transferred to procedure-name-1, procedure-name-2, ...,procedure-name-n, DEPENDING ON the value of identifier being 1, 2, . . . , n. If the value of identifier is greater than n or equal to zero, control is passed to the sentence following this statement.

(8) Format 3 causes control to be transferred from a USE procedure to the I-O control system. The following rules apply to the GO TO MORE-LABELS option:

   (a) Format 3 can appear only within a label-processing section in the declarative portion.

   (b) When an input file is being processed, format 3 is a request to the I-O control routine to make the next standard user label record available, and transfer control to the beginning of the USE procedure. If there are no more labels to be processed, control is returned to the Procedure Division.

   (c) When an output file is being processed, format 3 requests the I-O control routine to write the label in the user label area and return control to the first statement in the USE procedure so as to permit another label record to be created in the user label area.

6.7.15. IF

Function:

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

Format:

IF *condition*; [THEN] { *statement-1* / NEXT SENTENCE } [; { ELSE / OTHERWISE* } { *statement-2* / NEXT SENTENCE }]

Rules:

(1) Statement-1 and statement-2 represent either a conditional statement or an imperative statement.

(2) The NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.

(3) Execution of an IF statement takes the following action:

(a) Condition TRUE: Statements immediately following the condition (statement-1) are executed; control then passes implicitly to the next sentence.

(b) Condition FALSE: Either statement-2 is executed or, if ELSE is omitted, the next sentence is executed.

(4) Statement-1 and statement-2 may contain an IF statement, and the IF is considered nested. IF statements within IF statements are considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

(5) When control is passed to the next sentence, it is transferred to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

(6) The condition in an IF statement causes the object program to select between alternate control paths, depending on the truth value of a test. There are five possible types of conditions:

- relation condition
- class condition
- condition-name (restricted in basic compiler, see 5.3.12)
- switch-status condition
- sign condition

These conditions are discussed under rules 7 through 11.

**OTHERWISE is an extension to American National Standard COBOL.*

The logical operators (extended compiler only) used in combination with these conditions are:

- OR
- AND
- NOT

Table 6–1 indicates the relationship between the logical operators and conditions A and B.

CONDITION		CONDITION AND VALUE		
A	B	IF A AND B	IF A OR B	IF NOT A
TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

Table 6–1. Logical Operator/Condition Relationships

The ways in which conditions and logical operators may be combined are shown in Table 6–2.

FIRST SYMBOL	SECOND SYMBOL					
	condition	OR	AND	NOT	(	)
condition	no	yes	yes	no	no	yes
OR	yes	no	no	yes	yes	no
AND	yes	no	no	yes	yes	no
NOT	yes	no	no	no	yes	no
(	yes	no	no	yes	yes	no
)	no	yes	yes	no	no	yes

Table 6–2. Logical Operator/Condition Combinations

(7) Relation Condition

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, or an arithmetic expression. General format for a relation condition is:

$$\left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\\ \textit{arithmetic-expression-1}\end{array}\right\} \textit{relational-operator} \left\{\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\\ \textit{arithmetic-expression-2}\end{array}\right\}$$

The first operand is called the subject of the condition; the second operand is called the object of the condition. The subject and object may not both be literals. The relational-operator specifies the type of comparison to be made in a relational condition. The relational-operators and the format in which they are used is:

```
IF { identifier-1 }
   { literal-1    }
   { arithmetic-expression-1 }

   { IS { [NOT] GREATER THAN }                    }
   {    { [NOT] >            }                    }
   { IS { [NOT] LESS THAN    }                    }
   {    { [NOT] <            }                    }
   { IS { [NOT] EQUAL TO     }                    }
   {    { [NOT] =            }                    }
   { EQUALS*                                      }
   { UNEQUAL*                                     }
   { EXCEEDS*                                     }

   { identifier-2 }
   { literal-2    }
   { arithmetic-expression-2 }
```

When relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

(a) The omission of the subject of the relation condition, or

(b) The omission of the subject and relational operator of the relation condition.

Within a sequence of relation conditions both forms of abbreviation may be used. The effect of using such abbreviations is as if the omitted subject was replaced by the last preceding stated subject, or the omitted relational operator was replaced by the last preceding stated relational operator.

Ambiguity may result from using NOT in conjunction with abbreviations. In this event, NOT will be interpreted as a logical operator rather than as part of a relational operator. Thus:

```
A > B AND NOT > C OR D
```

is equivalent to:

```
A > B AND NOT A > C OR A > D
```

or

```
A > B AND (NOT A > C) OR A > D
```

Comparison of the various types of operands is accomplished as follows:

(a) Numeric Operands

For numeric operands comparison is made with respect to the algebraic value of the operands. The number of digits in the operands is not significant. Zero is considered a unique value regardless of the sign.

*American National Standard COBOL extensions.

Comparison of these operands is permitted regardless of their usage. Unsigned numeric operands are considered positive for purposes of comparison.

(b) Nonnumeric Operands

For nonnumeric operands or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters.

The size of an operand is the total number of characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same. The two cases to be considered are operands of equal size and operands of unequal size.

- Operands of Equal Size

  Corresponding character positions are compared, starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the item is reached, whichever comes first. The items are determined to be equal if all pairs of characters are equal.

  The first encountered pair of unequal characters is compared for relative location in the UNIVAC 9400 collating sequence (see Appendix A for hexadecimal values of characters). The operand which contains that character which is positioned higher in the UNIVAC 9400 collating sequence is determined to be the greater operand.

- Operands of Unequal Size

  Comparison proceeds as though the shorter operand was extended on the right by sufficient spaces to make the operands of equal size.

(c) Index-Names and/or Index Data-Items

- Two Index-Names

  The result is the same as if the corresponding occurrence numbers are compared.

- Index-Name and Data-Item or Literal

  The occurrence number that corresponds to the value of the index-name is compared to the data-item or literal, both of which must be elementary unsigned integers.

- Index Data-Item and Index-Name or Two Index Data-Items

  The actual values are compared without conversion.

The result of the comparison of an index data-item with any data-item or literal not specified above is undefined.

(8) Class Condition

The class condition determines whether the operand is numeric or alphabetic. The general format for the class condition is:

IF *identifier* IS [NOT] { NUMERIC / ALPHABETIC }

The operand being tested must be described, implicitly or explicitly, as USAGE IS DISPLAY.

- Numeric Test

  Identifier can be described as alphanumeric or numeric. If the record description of the item being tested does not contain an operational sign, the item is considered numeric only if the contents are numeric and a sign is not present.

- Alphabetic Test

  Identifier must be described as alphabetic. The item being tested is considered alphabetic only if the contents consist of any combination of the characters A through Z and the space.

(9) Condition-Name Condition

A conditional variable is tested to determine whether its value is equal to one of the values associated with a condition-name. In the extended compiler a condition-name may be associated with a range(s) of values; the conditional variable is then tested to determine whether its value falls within this range of values.

The format for a condition-name condition is:

IF [NOT] *condition-name*

(10) Switch-Status Condition

Determines the ON or OFF status of a switch as described in 4.2.3 (10). The condition-name specified in the ON or OFF STATUS IS option is tested in the following format:

IF [NOT] *condition-name*

(11) Sign Condition

Determines whether the algebraic value of a numeric operand is less than, greater than, or equal to zero. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. The format for a sign condition is:

IF { *identifier* / *arithmetic-expression* } IS [NOT] { POSITIVE / NEGATIVE / ZERO }

6.7.16. INSERT

The INSERT statement is available in the extended compiler only and is an extension to American National Standard COBOL.

Function:

The INSERT statement is used to add a logical record to indexed and direct access files.

Format:

INSERT *record-name* [FROM *identifier-1*] INVALID KEY *imperative-statement*

Rules:

(1) The INSERT verb can be used only when ACCESS IS RANDOM and ORGANIZATION IS DIRECT or INDEXED.

(2) A file must be OPENed prior to execution of the first INSERT statement for that file.

(3) The record-name is the name of a logical record in the File Section of the Data Division and must not be part of a sort file.

(4) When the FROM option is used, data is moved from identifier-1 to record-name according to the rules specified for the MOVE verb without the CORRESPONDING option.

(5) After the INSERT is executed, information in record-name is no longer available, but identifier-1 information is available.

(6) The INVALID KEY is used for processing direct access files. In random access mode, the INSERT performs the function of a SEEK statement prior to writing. The imperative-statement is executed when the contents of the KEY(s) being used to locate the record is found to be invalid.

6.7.17. MOVE

Function:

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

Formats:

Format 1:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{literal-1} \end{matrix} \right\} \underline{\text{TO}}\ \textit{identifier-2}\ [, \textit{identifier-3}] \ldots$$

Format 2:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \textit{identifier-1}\ \underline{\text{TO}}\ \textit{identifier-2}$$

Rules:

(1) If the CORRESPONDING option is used, selected items within identifier-1 are moved to selected items within identifier-2 according to rule (8) in 6.6.1 except that identifiers need not be numeric and may be either

(a) both elementary items, or

(b) one elementary item and one group item.

Only one identifier may appear to the right of the word TO, and the results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

(2) When moving to more than one area, the data designated by literal-1 or identifier-1 is moved first to identifier-2, then to identifier-3, etc.

(3) Any MOVE in which both the sending and receiving items are elementary items is an elementary MOVE. Every elementary item belongs to one of the following categories:

- numeric
- alphabetic
- alphanumeric
- numeric edited
- alphanumeric edited

Table 6–3 shows legal categories of sending and receiving fields:

SENDING	RECEIVING				
	Numeric	Alphabetic	Alphanumeric	Numeric Edited	Alphanumeric Edited
Numeric	YES	NO	YES*	YES	YES*
Alphabetic	NO	YES	YES	NO	YES
Alphanumeric	YES	YES	YES	YES	YES
Numeric Edited	NO	NO	YES	NO	YES
Alphanumeric Edited	NO	YES	YES	NO	YES

** A numeric item whose implicit decimal point is not immediately to the right of the least significant digit must not be moved to an alphanumeric or alphanumeric edited data item.*

Table 6–3. Sending and Receiving Fields

(4) The following rules apply to legal elementary moves:

(a) When the receiving field is alphanumeric edited, alphanumeric, or alphabetic, justification and any necessary spacefilling takes place as defined under the JUSTIFIED clause. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.

(b) When the receiving field is a numeric or numeric edited item, alignment by decimal point and any necessary zerofilling takes place, except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. Truncation occurs if the sending item has more digits to the left or right of the decimal point than the receiving item can contain. The result at object time is undefined if the sending item contains any nonnumeric characters.

(c) Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in the receiving item.

(d) When the sending field is an edited item, it will be treated as an alphanumeric item.

(e) An index data item cannot appear as an operand in a MOVE statement.

(5) Any MOVE that is not an elementary MOVE is treated as if it were an alphanumeric to alphanumeric elementary MOVE, except that there is no conversion of data from one form of internal representation to another.

(6) The figurative constant ZERO (ZEROS, ZEROES) belongs to the numeric category. The figurative constant SPACE (SPACES) belongs to the alphabetic category. All other figurative constants belong to the alphanumeric category.

6.7.18. MULTIPLY

Function:

The MULTIPLY statement causes numeric data items to be multiplied and sets the value of a data item equal to the results.

Formats:

Format 1:

MULTIPLY { *identifier-1* / *literal-1* } BY *identifier-2* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Format 2:

MULTIPLY { *identifier-1* / *literal-1* } BY { *identifier-2* / *literal-2* }

GIVING *identifier-3* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Rules:

(1) Only identifier-3, in format 2 may refer to a data item that contains editing symbols. All other identifiers must refer to numeric elementary items.

(2) Each literal must be a numeric literal.

(3) When format 1 is used, the initial value of identifier-1 is multiplied by the initial value of identifier-2. The value of the multiplier (identifier-2) is replaced by the product resulting from operation on that identifier.

(4) When format 2 is used, the initial value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2, and the result is stored in identifier-3.

(5) The maximum size of each operand is 18 decimal digits.

(6) For a description of the ROUNDED and SIZE ERROR option, see 6.6.1.

6.7.19 NOTE

Function:

The NOTE sentence allows COBOL programmers to write commentary which will be produced on the listing, but not compiled.

Format:

NOTE *character-string*.

Rules:

(1) Any combination of the characters from the UNIVAC 9400 System character set may be included in the character-string.

(2) If a NOTE sentence is the first sentence of the paragraph, the entire paragraph is considered to be part of the character-string, whereas a comment line (see Table 2–3) is not.

(3) If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first occurrence of a period followed by a space.

6.7.20. OPEN

Function:

The OPEN statement initiates the processing of both input and output files. It initiates checking and/or writing of labels and other input-output operations.

Format:

```
OPEN { INPUT { file-name [ REVERSED ] }... }
     { OUTPUT { file-name [WITH NO REWIND] }... }...
     { I-O {file-name}... }
```

Rules:

(1) At least one of the options INPUT, OUTPUT, or I-O must be specified. These options may appear in any order.

(2) The I-O option pertains only to mass storage files.

(3) The REVERSED and NO REWIND options apply only to sequential single reel processing.

(4) The OPEN statement must be applied to all files except sort files.

(5) File-name refers to the FD name in the File Section of the Data Division.

(6) The OPEN statement for a file must be executed prior to the first READ, INSERT, REWRITE, SEEK, or WRITE statement for that file.

(7) A second OPEN statement for a file must not be executed prior to the execution of a CLOSE statement for that file.

(8) The OPEN statement does not obtain or release the first data record. When checking or writing labels, the user's beginning label subroutine will be executed if one has been specified by a USE statement.

6.7.21. PERFORM

Function:

This verb permits a temporary departure from the normal sequence of execution in order to execute one or more procedures, either a specified number of times or until a specified condition is satisified, after which control is automatically returned to the normal sequence. Formats 3 and 4 are available only in the extended compiler.

Formats:

Format 1:

PERFORM *procedure-name-1* [THRU *procedure-name-2*]

Format 2:

PERFORM *procedure-name-1* [THRU *procedure-name-2*] { *identifier-1* / *integer-1* } TIMES

Format 3:

PERFORM *procedure-name-1* [THRU *procedure-name-2*] UNTIL *condition-1*

Format 4:

PERFORM *procedure-name-1* [THRU *procedure-name-2*]

VARYING { *index-name-1* / *identifier-1* } FROM { *index-name-2* / *literal-2* / *identifier-2* }

BY { *literal-3* / *identifier-3* } UNTIL *condition-1*

[AFTER { *index-name-4* / *identifier-4* } FROM { *index-name-5* / *literal-5* / *identifier-5* }

BY { *literal-6* / *identifier-6* } UNTIL *condition-2*

[AFTER { *index-name-7* / *identifier-7* } FROM { *index-name-8* / *literal-8* / *identifier-8* }

BY { *literal-9* / *identifier-9* } UNTIL *condition-3*]]

Rules:

(1) Each procedure-name is the name of a section or paragraph in the Procedure Division.

(2) Each identifier represents a numeric elementary item described in the Data Division. In format 2, and format 4 with the AFTER option, each identifier represents a numeric item with no positions to the right of the assumed decimal point.

(3) Each literal represents a numeric literal.

(4) When the PERFORM statement is executed, control is transferred to the first statement after procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows:

(a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, return is after the last statement of procedure-name-1.

(b) If procedure-name-1 is a section name and procedure-name-2 is not specified, return is after the last statement of the last paragraph in procedure-name-1.

(c) If procedure-name-2 is a:

- paragraph-name, return is after the last statement of the paragraph,
- section-name, return is after the last sentence of the last paragraph in the section.

(5) If there are two or more direct paths to a return point in a group of procedures being PERFORMed, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all these paths must lead. If control passes to these procedures by means other than a PERFORM statement, regardless of use of EXIT, control passes through the last statement of the procedure to the following statement.

(6) Format 1 is the basic PERFORM statement. A procedure referred to by this type PERFORM is executed once after which control is passed to the statement following the PERFORM statement.

(7) Format 2 is the TIMES option. When the TIMES option is used, the procedures are PERFORMed the number of times specified by identifier-1 or integer-1. Control is then transferred to the statement following the PERFORM statement. The value of identifier-1 or integer-1 must not be negative, and if the value is zero, control passes immediately to the statement following the PERFORM statement. Once the PERFORM statement has been initiated, any reference to identifier-1 has no effect in varying the number of times the procedures are executed.

(8) Format 3 is the UNTIL option. The specified procedures are PERFORMed until the condition specified by the UNTIL option is true. Then, control is transferred to the statement following the PERFORM statement. Note that if the condition specified by the UNTIL option is true at the beginning of the execution of the PERFORM statement, control passes to the statement following the PERFORM statement.

(9) Format 4 is the VARYING option. This option is used to change the value of one or more identifiers or index-names during the execution of a PERFORM statement. When index-names are used, the FROM and BY clauses have the same effect as in a SET statement. In rules 10 through 12, reference to identifier as the object of VARYING and FROM phrases also refers to index-name.

(10) When one identifier is varied:

(a) Identifier-1 is set to its initial value, either identifier-2 or literal-2.

(b) If condition-1 is false, the sequence of procedures is executed once, and the value of identifier-1 is incremented or decremented by identifier-3 or literal-3, and condition-1 is evaluated again. This cycle continues until condition-1 is true, after which control is passed to the statement following the PERFORM statement.

(c) If condition-1 is true at the beginning of execution of the PERFORM statement, control passes directly to the statement following the PERFORM statement.

(11) When two identifiers are varied:

(a) Identifier-1 and identifier-4 are set to their initial values, identifier-2 and identifier-5, respectively. During execution, these initial values must be positive.

(b) Condition-1 is evaluated. If true, control is passed to the statement following the PERFORM statement. If false, condition-2 is evaluated.

(c) If condition-2 is false, the sequence of procedures is executed once, after which identifier-4 is changed by identifier-6, and condition-2 is evaluated again. This cycle continues until condition-2 is true.

(d) When condition-2 is true, identifier-4 is set to its initial value (identifier-5), identifier-1 is changed by identifier-3, and condition-1 is reevaluated.

(e) The PERFORM statement is completed when condition-1 is true; if false, the cycle continues until condition-1 is true.

(f) Figure 6–1 illustrates the logic of the PERFORM statement when two identifiers are varied:

I_n = identifier
L_n = literal
C_n = condition
P_n = procedure-name

(g) At the termination of this PERFORM statement, identifier-4 contains its initial value, while identifier-1 contains a value that differs from the last used setting by an increment or decrement depending on identifier-3. If condition-1 was true when the PERFORM statement was initiated, identifiers-1 and -4 contain their initial values.

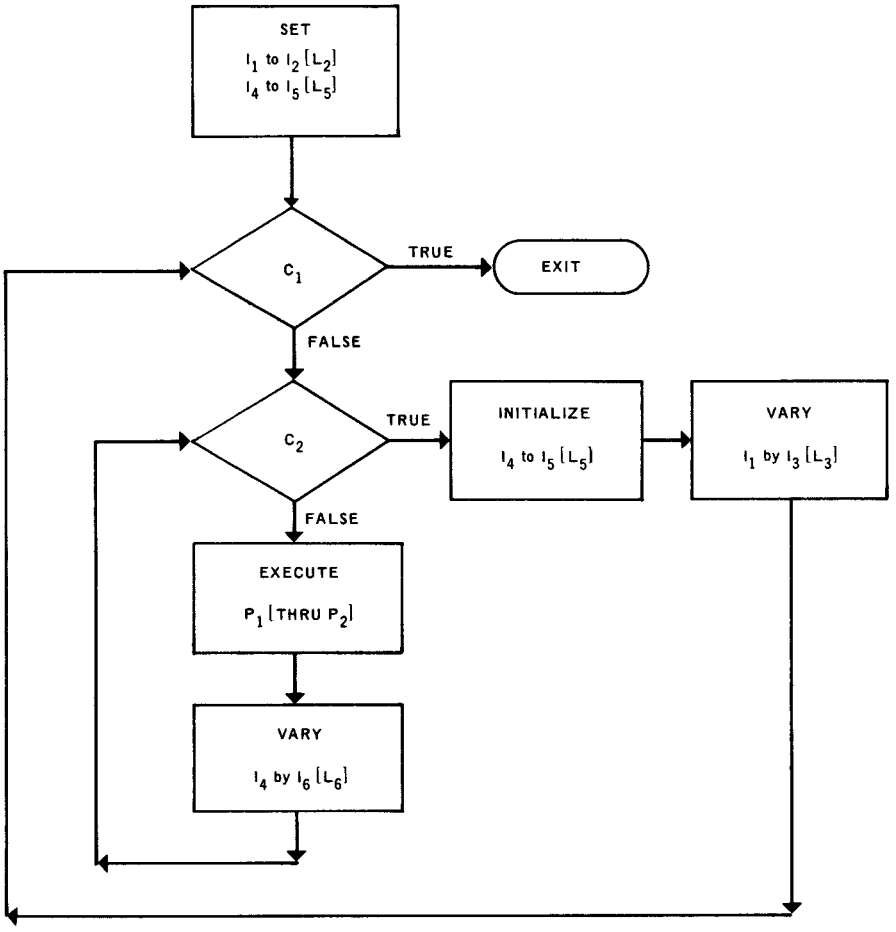


Figure 6–1. PERFORM Logic: Varying Two Identifiers

(12) When three identifiers are varied:

(a) Logic is the same as for two identifiers, except that identifier-7 goes through a complete cycle each time identifier-4 is changed by identifier-6 which, in turn, goes through a complete cycle each time identifier-1 is varied.

(b) Figure 6–2 illustrates the logic of the PERFORM statement when three identifiers are varied:

I_n = identifier
L_n = literal
C_n = condition
P_n = procedure-name

At the termination of this PERFORM statement, identifier-4 and identifier-7 contain their initial values, while identifier-1 contains a value that differs from the last used setting by an increment or decrement depending on identifier-3. If condition-1 was true when the PERFORM statement was initiated, identifier-1, identifier-4, and identifier-7 each contains its initial value.

(13) A PERFORM statement within a section which has a priority number less than the SEGMENT-LIMIT can have within its range only the following:

(a) Sections with priority numbers of less than 50.

(b) Sections wholly contained in a single segment whose priority number is greater than 49.

(14) A PERFORM statement that appears in a section which has a priority number greater than the SEGMENT-LIMIT can have within its range only the following:

(a) Sections with the same priority number as the section containing the PERFORM statement.

(b) Sections with a priority number less than the SEGMENT-LIMIT.

(15) Independent segments are made available in their initial state. Fixed overlayable segments are made available in their last used state.

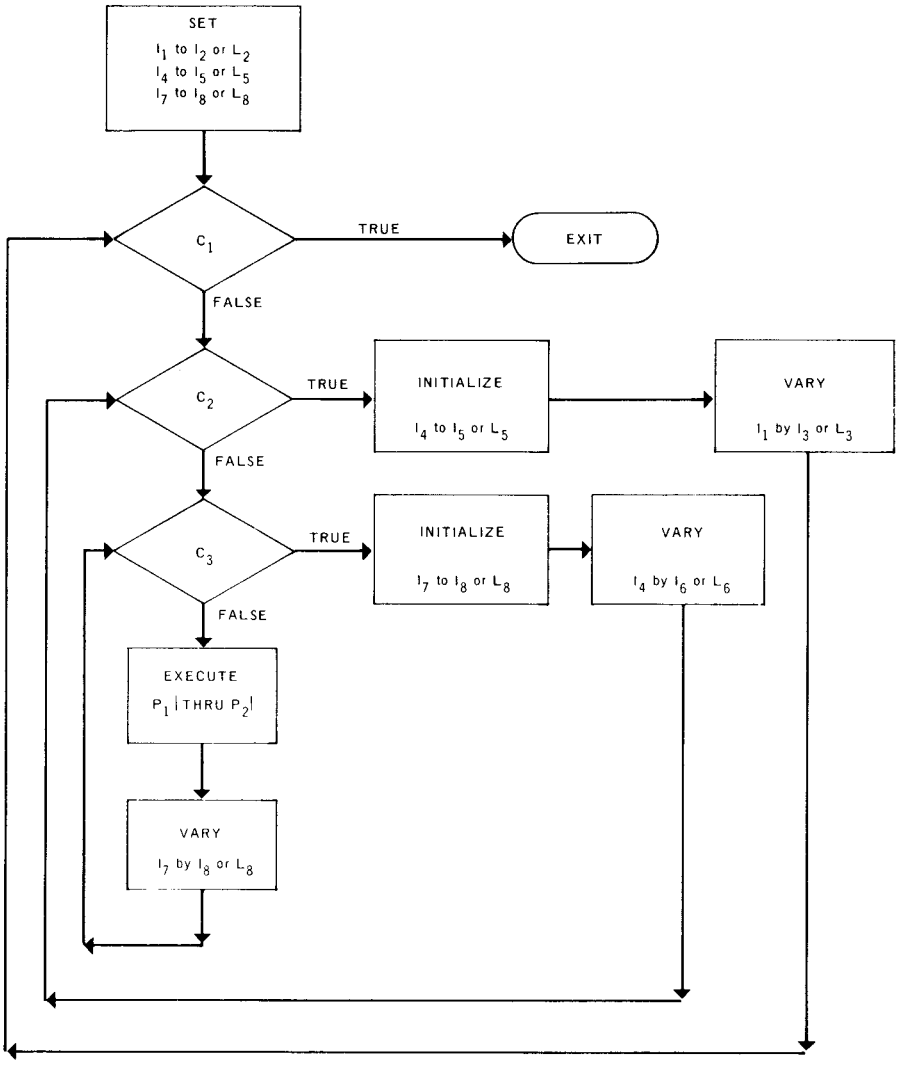


Figure 6–2. PERFORM Logic: Varying Three Identifiers

(16) If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit.

6.7.22. READ

Function:

For sequential file processing, the READ statement makes available the next logical record from an input file and allows performance of a specified imperative-statement when end of file is detected.

For mass storage files in the random access mode, the READ statement makes available a specific record from a direct access device, and allows performance of a specified imperative-statement if the contents of the associated KEYS are found to be invalid.

Format:

READ *file-name* RECORD [INTO *identifier*] { AT END / INVALID KEY } *imperative-statement*

Rules:

(1) An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file.

(2) When a file consists of more than one type of record, the records automatically share the same storage area.

(3) If, while processing a sequential file, the logical end of the file is reached during the execution of a READ statement, the statement specified in the AT END phrase is executed. After the execution of this phrase, a READ statement for that file must not be initiated without prior execution of a CLOSE and an OPEN statement for that file.

(4) If an OPTIONAL file is not present, the imperative-statement in the AT END phrase is executed on the first READ.

(5) The READ statement performs the functions of the SEEK statement implicitly for random access files.

(6) If a direct access file (INPUT or I-O) in the random access mode is contained on more than one physical unit, any end of unit procedures are the responsibility of the user.

(7) The INTO option may be used only when the input file contains just one size record, and file-name cannot be the name of a sort file. Moving INTO will be performed according to the rules specified for the MOVE verb, without the CORRESPONDING option.

(8) Data items of a logical record cannot be accessed prior to the READ of the associated record. (The record area may not be accessed prior to a READ.)

6.7.23. RELEASE

Function:

The RELEASE statement, available with the extended compiler only, transfers records to the initial phase of a sort operation.

Format:

RELEASE *record-name* [FROM *identifier*]

Rules:

(1) Record-name must be named in the DATA RECORDS clause of its associated sort file.

(2) Identifier in the FROM option must refer to a data item in Working-Storage or in an input record area.

(3) The identifier and record-name must name different data items.

(4) If the FROM option is used, the contents of identifier is moved to record-name, then the contents of record-name is released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING option. The information in the record area is no longer available, but the information in identifier is available.

6.7.24. RETURN

Function:

The RETURN statement, available with the extended compiler only, obtains sorted records from the final phase of the sort operation.

Format:

RETURN *file-name* RECORD [INTO *identifier*] ; AT END *imperative-statement*

Rules:

(1) File-name must be a sort file with an SD entry in the Data Division.

(2) A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name.

(3) The identifier in the INTO option must be the name of a Working-Storage area or output record area, and the output file must contain only one type of record. The data is available in both the output record area and the identifier area.

(4) The execution of a RETURN statement causes the next record to be made available in the order specified by the KEYs listed in the SORT statement for processing in the record area associated with the sort file.

(5) Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option.

(6) After execution of the AT END phrase, no RETURN statements may be executed within the current output procedure.

6.7.25. REWRITE

Function:

This verb is an extension to American National Standard COBOL. It performs the same function as the WRITE statement and is provided for compatibility only.

Formats:

Format 1:

REWRITE *record-name* [FROM *identifier*]

Format 2:

REWRITE *record-name* [FROM *identifier*] INVALID KEY *imperative-statement*

Rules:

(1) A file must be OPENed prior to execution of the first REWRITE statement for that file.

(2) The record-name is the name of a logical record in the File Section of the Data Division and must not be part of a sort file.

(3) When the FROM option is used, data is moved from identifier-1 to record-name according to the rules specified for the MOVE verb without the CORRESPONDING option.

(4) After the REWRITE is executed, information in record-name is no longer available, but identifier-1 information is available.

(5) The INVALID KEY option in format 2 is used for processing direct access files.

(a) In sequential access mode, the imperative-statement is executed when the end of the last segment of the file is reached and an attempt is made to execute a REWRITE for that file.

(b) In random access mode, the REWRITE performs the function of a SEEK statement prior to writing. The imperative-statement is executed when the contents of the KEY(s) being used to locate the record is found to be invalid.

6.7.26. SEARCH

Function:

The SEARCH statement is used to search a table for a table-element that satisfies the specified condition and to adjust the associated index-name to indicate that table-element.

Formats:

Format 1:

```
SEARCH identifier-1 [ VARYING { index-name-1 } ]
                              { identifier-2 }

[; AT END imperative-statement-1 ]

; WHEN condition-1 { imperative-statement-2 }
                   { NEXT SENTENCE         }

[; WHEN condition-2 { imperative-statement-3 } ] ...
                    { NEXT SENTENCE         }
```

Format 2:

```
SEARCH ALL identifier-1 [; AT END imperative-statement-1]

; WHEN condition-1 { imperative-statement-2 }
                   { NEXT SENTENCE         }
```

Rules:

(1) In both formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in format 2 must also contain the KEY IS phrase in its OCCURS clause.

(2) Identifier-2 , when specified, must be described as USAGE IS INDEX or as a numeric elementary data item without any positions to the right of the assumed decimal point.

(3) In format 1, condition-1, condition-2, etc., may be any condition as described in 6.7.15.

(4) In format 2, condition-1 may consist of a relation condition incorporating the relation EQUALS or EQUAL TO or equal sign, or a condition-name condition, where the VALUE clause that describes the condition-name contains only a single literal. Alternatively, condition-1 may be a compound condition formed from simple conditions of the type just mentioned, with AND as the only connective. Any data-name that appears in the KEY clause of identifier-1 may appear as the subject or object of a test or be the name of the conditional variable with which the tested condition-name is associated; however, all preceding data-names in the KEY clause must also be included within condition-1. No other tests may appear within condition-1.

(5) If format 1 of the SEARCH is used, a serial type of search operation takes place, starting with the current index setting.

(a) If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next sentence.

(b) If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisified, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element which exceeds the last element of the table by one or more occurrences, in which case the search terminates as indicated in 5a. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

(6) In a format 2 SEARCH, the results of the SEARCH ALL operation are predictable only when the data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of identifier-1.

(7) If format 2 of the SEARCH is used, a nonserial type of search operation takes place, in which case the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation using a binary search technique.

If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END phrase appears, or to the next sentence when this phrase does not appear; in either case the final setting of the index is set to the first occurrence. If condition-1 can be satisfied, index indicates an occurrence that allows condition-1 to be satisfied and control passes to imperative-statement-2.

(8) After execution of an imperative-statement-1, imperative-statement-2, or imperative-statement-3 that does not terminate with a GO TO statement, control passes to the next sentence.

(9) In format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY clause of identifier-1. Any other index-names for identifier-1 remain unchanged.

(10) In format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY clause of identifier-1. Any other index-names for identifier-1 remain unchanged.

(11) In format 1, if the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY clause of identifier-1 , that index-name is used for this search. If this is not the case, or if the VARYING identifier-1 phrase is specified, the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used for the search. In addition, the following operations will occur:

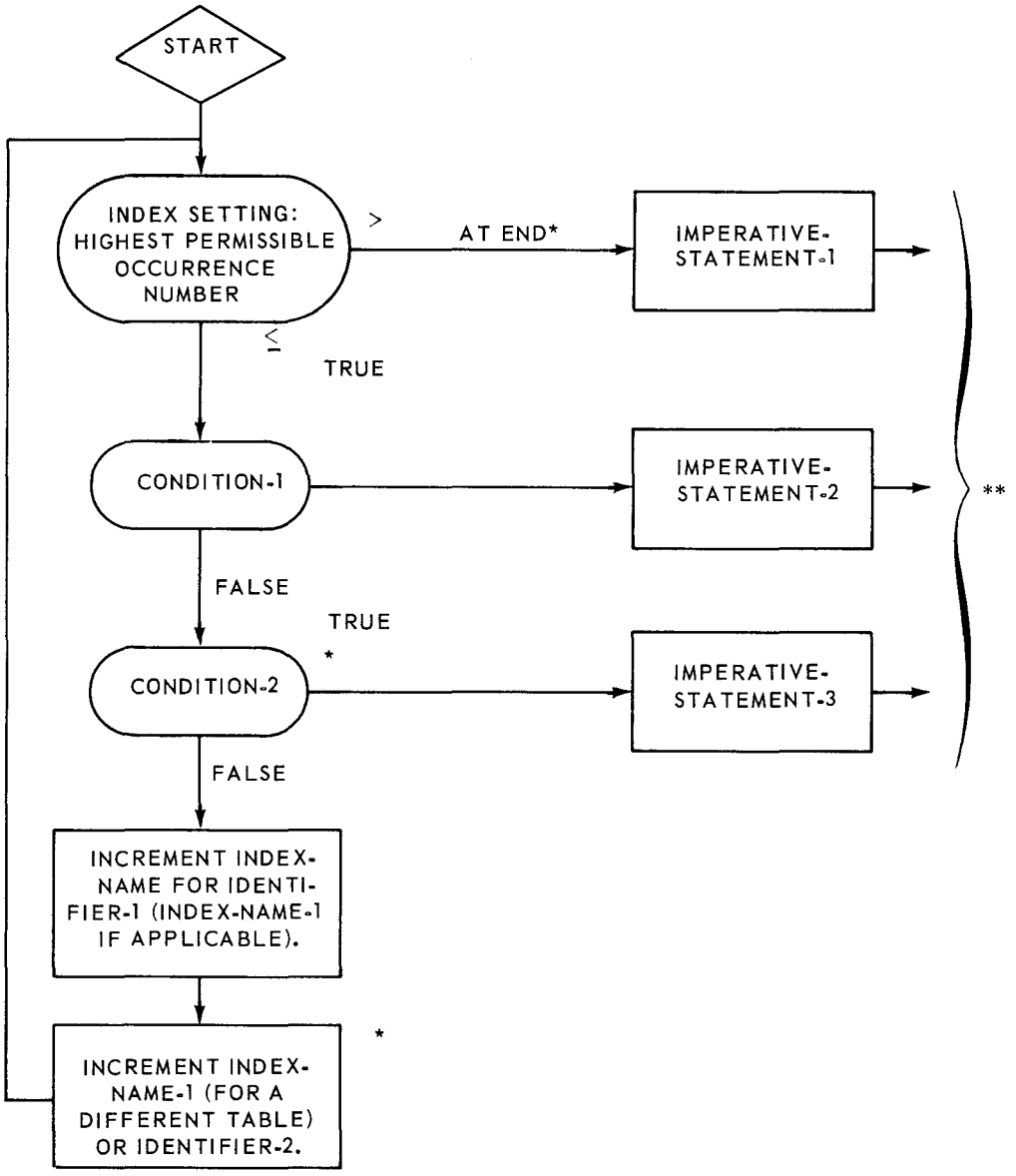
(a) If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY clause of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

(b) If the VARYING identifier-2 phrase is specified, identifier-2 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented. If identifier-2 has a USAGE IS INDEX clause, it is assumed to contain a value appropriate as an index setting for identifier-1.

(12) If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two- or three-dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two- or three-dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

(13) Format 2 is available only in the extended compiler.

A diagram of the Format 1 SEARCH operation containing two WHEN phrases follows.



* These operations are options included only when specified in the SEARCH statement.

** Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

Figure 6–3. SEARCH Logic

6.7.27. SEEK

Function:

The SEEK statement initiates access of a mass storage data record for subsequent reading or writing and is available only in the extended compiler.

Format:

SEEK *file-name* RECORD

Rules:

(1) A SEEK statement pertains only to the disc files as specified in the following chart.

ORGANIZATION TYPE	ACCESS METHOD	SEEK ALLOWED
SEQUENTIAL	SEQUENTIAL	NO
RELATIVE	SEQUENTIAL	YES
	RANDOM	
DIRECT	SEQUENTIAL	
	RANDOM	
INDEXED	SEQUENTIAL	
	RANDOM	NO

(2) The value of the identifier in the ACTUAL KEY clause is used by SEEK to determine the location of the record to be accessed when ORGANIZATION is RELATIVE or DIRECT. When ORGANIZATION IS INDEXED, the value of the identifier in the ACTUAL or SYMBOLIC key clause is used.

(3) Two SEEK statements for the same file may logically follow each other. Any validity check associated with the first SEEK is negated by the execution of a second SEEK statement.

→ 6.7.28. SET

Function:

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

Formats:

Format 1:

$$\underline{\text{SET}} \left\{ \begin{matrix} \textit{index-name-1} \\ \textit{identifier-1} \\ \textit{index-data-item-1} \end{matrix} \right\} \left[, \begin{matrix} \textit{index-name-2} \\ \textit{identifier-2} \\ \textit{index-data-item-2} \end{matrix} \right] \ldots \underline{\text{TO}} \left\{ \begin{matrix} \textit{literal-1} \\ \textit{index-name-3} \\ \textit{identifier-3} \\ \textit{index-data-item-3} \end{matrix} \right\}$$

Format 2:

$$\underline{\text{SET}}\ \textit{index-name-1} \left[, \textit{index-name-2} \right] \ldots \left\{ \begin{matrix} \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{matrix} \right\} \left\{ \begin{matrix} \textit{literal-1} \\ \textit{identifier-1} \end{matrix} \right\}$$

Rules:

(1) All identifiers must be either index data items or numeric elementary items described as unsigned without any positions to the right of the assumed decimal point, except that identifier-1 in format 2 must not be an index-data-item.

(2) All literals must be positive integers.

(3) All index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

(4) In format 2, the contents of index-name-1, index-name-2 ... are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of identifier-1 or literal-1.

(5) The following paragraphs explain the allowable combinations of choices in the SET statement:

(a) SET *index-name-1* TO *index-name-2*

The occurrence number value of index-name-2 is used to compute a new displacement value for index-name-1. Also, the occurrence number value of index-name-2 replaces that of index-name-1. If "length of one occurrence" is the same for both, no computation is necessary.

(b) SET *index-name* TO *index-data-item*

Same as (a), except that no computation takes place. If the value contained in the index-data-item does not correspond to an occurrence number of an element in the table indexed by index-name, the result is undefined.

(c) SET *index-name* TO { *identifier* / *literal* }

When identifier or literal is a numeric data item and USAGE is not INDEX. The value of identifier or literal is treated as an occurrence number and is used to compute a new displacement value for index-name. Identifier or literal must be elementary unsigned integer. Also, the value of identifier or literal replaces the occurrence number value of index-name.

(d) SET *index-data-item-1* TO { *index-name* / *index-data-item-2* }

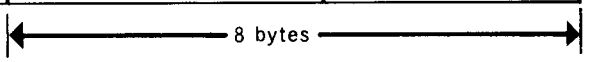
MOVE with no conversion is executed. Index-data-item-1 has no associated table element length; therefore, there is no unique displacement value for a given occurrence number value.

(e) SET *identifier* TO *index-name*

The value of the occurrence number of index-name replaces the value of identifier with appropriate conversion to the data type of identifier; i.e., conversion of binary occurrence number to packed decimal. Rules for MOVE statement with integer numeric sending field apply. Identifier must be a numeric data item, an alphanumeric data item, or a group item.

(6) Internal format of index-name and index-data-item:

DESCRIPTION OF CONTENTS	OCCURRENCE NUMBER IN BINARY	DISPLACEMENT IN BINARY
format	32 bits	32 bits
range	0 to 65,535	0 to 65,535



Index-name items are word aligned, but index-data-items are not aligned.

(7) Formula for calculating displacements for index-name:

Displacement = (occurrence-number-1) X (length of one occurrence)

6.7.29. SORT

Function:

The SORT statement creates a sort file by executing input procedures or by transferring records from another file. It sorts the records in the sort file on a set of specified keys, and makes each record from the sort file (in sorted order) available to some output procedures or to an output file. SORT is available only with the extended compiler.

Format:

```
SORT file-name-1 ON { DESCENDING } KEY { data-name-1 } ...
                    { ASCENDING  }

[; ON { DESCENDING } KEY { data-name-2 } ... ] ...
      { ASCENDING  }

{ INPUT PROCEDURE IS section-name-1 [THRU section-name-2] }
{ USING file-name-2                                       }

{ OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4] }
{ GIVING file-name-3                                       }
```

Rules:

(1) File-name-1 must be described in an SD entry in the Data Division.

(2) Each data-name must represent data items described in records associated with file-name-1. Numeric display key items may not exceed 16 digits; other key items must not exceed 256 characters. Key data-names may not be described with an OCCURS clause nor may they be subordinate to an entry which contains an OCCURS clause.

(3) Section-name-1 and section-name-3 are names of an input and output procedure, respectively.

(4) File-name-2 and file-name-3 must be described in an FD entry in the Data Division. They may not be described in an SD entry. However, the record format of file-name-2 and/or file-name-3 must be that specified for the sort file. The size of the logical record(s) described for file-name-2 and file-name-3 must be equal to the size of the logical record(s) described for file-name-1. File-name-2 and file-name-3 may not be described as containing undefined format records (RECORDING MODE IS U).

(5) A SORT statement may appear more than once in the Procedure Division of a program, but may not appear in the Declarative Section or in the input and output procedures associated with a SORT statement.

(6) INPUT PROCEDURE

(a) Must consist of one or more consecutive sections that do not form a part of any output procedure.

(b) Must include at least one RELEASE statement in order to transfer records to the sort file.

(c) RELEASE statements in the input procedure have no meaning unless they are controlled by a SORT statement; therefore, control must not be passed to the input procedure except when a related SORT statement is being executed.

(d) The input procedure can include any procedures needed to select, create, or modify records.

(e) The input procedure must not contain any SORT statements.

(f) ALTER, GO TO, and PERFORM statements are not permitted to refer to procedure-names outside the input procedure.

(g) ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

(7) OUTPUT PROCEDURE

(a) Must consist of one or more consecutive sections that do not form a part of any input procedure.

(b) Must include at least one RETURN statement in order to make sorted records available for processing.

(c) RETURN statements in the output procedure have no meaning unless they are controlled by a SORT statement; therefore, control must not be passed to the output procedure except when a related SORT statement is being executed.

(d) The output procedures can include any procedures needed to select, modify, or copy the records that are being returned one at a time in sorted order, from the sort file.

(e) Rules 6 (e), 6 (f), and 6 (g) also apply to the output procedures.

(8) When the ASCENDING clause is used, the sorted sequence is from lowest value of KEY to highest value according to the UNIVAC 9400 System character collating sequence shown in Appendix A. The sorted sequence is reversed when the DESCENDING clause is used. In the format, data-name-1 is the most significant key, data-name-2 is the next most significant key, and so on.

(9) The Record Description for every sort file must contain the KEY items data-name-1, data-name-2, and so on. When the KEY item appears in more than one record, the data descriptions must be equivalent and their starting position must always be the same number of character positions from the beginning of each record. Numeric display key items must not exceed 16 digits. Other key items must not exceed 256 characters.

(10) If INPUT PROCEDURE is specified, control is passed to section-name-1 before file-name-1 is sequenced by the SORT statement. When control passes the last statement of the input procedure, the records that have been released to file-name-1 are sorted.

(11) If the USING option is specified, all the records in the file-name-2 are transferred to the file-name-1. The SORT statement automatically performs the function of the OPEN, READ, USE, and CLOSE statements for file-name-2. File-name-2 must be a sequential access file.

(12) If OUTPUT PROCEDURE is specified, control passes to section-name-3 after file-name-1 has been sequenced by the SORT statement. When control passes the last statement of the output procedure, the sort is terminated and control is passed to the next statement after the SORT statement. The RETURN statements in the output procedure are the requests for the next sorted record.

(13) If the GIVING option is specified, all the sorted records in file-name-1 are transferred to file-name-3 as the implied output procedure for this SORT statement. File-name-3 is automatically opened before transferring the records and closed after the last record in the sort file is returned. File-name-3 must be a sequential access file.

6.7.30. STOP

Function:

To halt the object program permanently or temporarily, with or without a display of a literal.

Format:

$$\underline{\text{STOP}} \left\{ \begin{matrix} \textit{literal} \\ \underline{\text{RUN}} \end{matrix} \right\}$$

Rules:

(1) Literal may be numeric or nonnumeric, or any figurative constant except ALL.

(2) Literal is communicated to the operator through the typewriter console, and continuation of the program begins with the execution of the next statement after the STOP statement. The literal option is equivalent to a DISPLAY statement, but requires a reply from the operator to continue the program.

For example, the error routine

```
SEQ-ERROR.
STOP 'CARDS OUT OF SEQUENCE, CORRECT SEQUENCE, REPLACE
CARDS IN READER, ANSWER R WHEN READY'.
```

will cause the literal to be DISPLAYed as follows:

```
CD10 CARDS OUT OF SEQUENCE, CORRECT SEQUENCE, REPLACE
CARDS IN - CD10 READER, ANSWER R WHEN READY.
```

This will be followed by

CA10 ACCEPT READY

and program operation is suspended pending operator reply.

(3) When the RUN option is used, the object program is halted permanently; therefore, when this option appears in an imperative statement, it should appear as the only, or the last, statement in a sequence of imperative statements.

6.7.31. SUBTRACT

Function:

The SUBTRACT statement allows the programmer to subtract one, or the sum of two or more, numeric data items from one or more items, and to set the values of one or more items equal to the results.

Formats:

Format 1:

SUBTRACT { *literal-1* / *identifier-1* } [, *literal-2* / , *identifier-2*] ...

FROM *identifier-m* [ROUNDED] [, *identifier-n* [ROUNDED]] ...

[; ON SIZE ERROR *imperative-statement*]

Format 2:

SUBTRACT { *literal-1* / *identifier-1* } [, *literal-2* / , *identifier-2*] ...

FROM { *literal-m* / *identifier-m* } GIVING *identifier-n* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Format 3:

SUBTRACT { CORR / CORRESPONDING } *identifier-1*

FROM *identifier-2* [ROUNDED]

[; ON SIZE ERROR *imperative-statement*]

Rules:

(1) When format 1 is used, all literals and identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifier-n, etc., and the result of the subtraction is stored as the new value in identifier-m, identifier-n, etc.

(2) The maximum size of each operand is 18 decimal digits. The composite of operand, which is that data item resulting from the superimposition of all operands, excluding the data item that follows the word GIVING, aligned on their decimal points, must not contain more than 18 digits.

(3) In format 2, identifier-n may refer to a data item that contains editing symbols. All other identifiers must refer to numeric elementary items.

(4) When format 2 is used, all literals or identifiers preceding the word FROM are added together, and this total is subtracted from literal-m or identifier-m; then the result of the subtraction is stored as the new value in identifier-n.

(5) If format 3 is used, data items under identifier-1 are subtracted FROM and stored in CORRESPONDING data items under identifier-2.

(6) For a description of the ROUNDED, SIZE ERROR, and CORRESPONDING options, see 6.6.1.

6.7.32. TRANSFORM

Function:

The TRANSFORM statement may be used to alter characters of an identifier according to a user-defined transformation rule or table. It may also be used to effect code base translation between EBCDIC and ASCII via compiler supplied tables.

Formats:

Format 1:

TRANSFORM *identifier-3* [, *identifier-4*] . . . CHARACTERS

FROM { *identifier-1* / *nonnumeric-literal-1* / *figurative-constant-1* } TO { *identifier-2* / *nonnumeric-literal-2* / *figurative-constant-2* }

Format 2:

TRANSFORM *identifier-3* [, *identifier-4*] . . . CHARACTERS

FROM { EBCDIC TO ASCII / ASCII TO EBCDIC }

Format 3:

TRANSFORM *identifier-3* [, *identifier-4*] . . . CHARACTERS

{ ON / BY } *identifier-5*

Rules:

(1) All identifiers used in this statement must be described either explicitly or implicitly as USAGE IS DISPLAY. Identifier-1, identifier-2, or identifier-5 may not be variable length operands.

(2) The least significant digit position of a signed numeric DISPLAY item without a SEPARATE SIGN clause is treated as a single character, not a signed digit.

(3) In format 1, identifier-1 and identifier-2 must not exceed 256 characters in length. The length of identifier-2 must be either equal to the length of identifier-1, or have a length of 1 character.

(4) In format 1, nonnumeric-literals must be enclosed in quotation marks (or apostrophes).

(5) In format 1, all figurative-constants are permitted except ALL nonnumeric-literal.

(6) In format 1, a character must not be duplicated in identifier-1 or in nonnumeric-literal-1.

(7) In format 3, identifier-5 must have a length of 256 characters.

(8) The following paragraphs explain the allowable combinations of choices in the TRANSFORM statement.

(a) The following rules apply to these combinations in format 1:

identifier-1 TO identifier-2

identifier-1 TO nonnumeric-literal-2

nonnumeric-literal-1 TO identifier-2

nonnumeric-literal-1 TO nonnumeric-literal-2

nonnumeric-literal-1 TO figurative-constant-2

identifier-1 TO figurative-constant-2

- If the FROM and the TO operands have the same length, any occurrence in identifier-3, identifier-4, and so on, of a character (or the single character) in operand-1 is replaced by the character in the corresponding position (or the single character) of operand-2.

- If the FROM operand exceeds one character and the TO operand is only one character, any occurrence in identifier-3, identifier-4, and so on, of any character in operand-1 is replaced by the single character in operand-2.

(b) The following rule applies to these combinations in format 1:

figurative-constant-1 TO identifier-2

figurative-constant-1 TO nonnumeric-literal-2

figurative-constant-1 TO figurative-constant-2

- Length of operand-1 and operand-2 is one character. Any occurrence in identifier-3 of the single character in operand-1 is replaced by the single character in operand-2.

(c) The following applies to format 2:

- Identifier-3 is transformed from ASCII to EBCDIC or from EBCDIC to ASCII depending on the FROM and TO operands according to the table contained in Appendix N.

(d) The following rules apply to format 3:

- Identifier-3 may be described as any length up to a maximum of 65,535 characters.
- Identifier-5 is a 0–255 binary value positional translate table, i.e., any character in identifier-3 with a binary value of 0, will be transformed to the character in the first position of identifier-5; any character in identifier-3 with a binary value of 1, will be transformed to the character in the second position of identifier-5, etc.

(9) The TRANSFORM verb is an extension to ANS COBOL.

6.7.33. USE

Function:

The USE statement specifies procedures for input-output label and error handling that are in addition to the standard procedures specified by the input-output system. The three format options available are:

- Label writing and checking
- Error checking
- Printer form-overflow

Format 3 is an extension to American National Standard COBOL.

Formats:

Format 1:

```
USE {BEFORE} STANDARD [BEGINNING] [REEL]
    {AFTER }          [ENDING   ] [FILE]
                                  [UNIT]

LABEL PROCEDURE ON {file-name-1 [, file-name-2]...}
                   {INPUT                         }
                   {OUTPUT                        }
                   {I-O                           }.
```

Format 2:

```
USE AFTER STANDARD ERROR PROCEDURE ON {file-name-1 [, file-name-2]...}
                                      {INPUT                         }
                                      {OUTPUT                        }
                                      {I-O                           }.
```

Format 3:

```
USE FOR FORM-OVERFLOW ON file-name-1.
```

Rules:

(1) A USE statement must immediately follow a section header in the Declaratives portion of the Procedure Division, and must be followed by a period. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

(2) The USE statement defines the conditions calling for the execution of the USE procedures, and the USE statement itself is never executed.

(3) When format 1 is used:

(a) If the file-name option is present, the FD entry for file-name-1 must contain a LABEL RECORDS ARE data-name clause.

(b) If BEGINNING or ENDING is omitted, the designated procedures are executed for both beginning and ending labels. Ending is not applicable for direct access files whose organization is other than sequential.

(c) If the REEL/UNIT clause is used, the designated procedures are executed for each new REEL or UNIT of a file but not for the start or end of the file itself. If UNIT, REEL, FILE clause is omitted, the designated procedures are executed for the REEL or UNIT, whichever is applicable, and the FILE. The REEL option is not applicable to mass storage files and the UNIT option is not applicable to files in the random access mode.

(d) When the INPUT, OUTPUT, or I-O option is specified, the USE procedure refers to all appropriate files except those described with the LABEL RECORDS OMITTED or STANDARD clause.

(e) The BEFORE option is not applicable to the UNIVAC 9400 System, but is accepted for compatibility. The BEFORE option is processed as if AFTER were specified.

(4) When format 2 is used, the USE procedure will be initiated when system standard I-O error recovery procedures have been exhausted.

(5) When format 3 is used, control will be transferred to the USE procedure when a printer carriage overflow condition is detected. The carriage tape channel 9 is used to indicate overflow. Format 3 is an extension to the COBOL standard.

Overflow is detected during the print and space functions of the printer. If form positioning by means of the paper tape loop is specified (advancing mnemonic name), a form overflow condition does not occur.

(6) File-name must not represent a sort file in any format.

(7) Input-output statements are not allowed inside USE procedures except for the following verbs:

(a) ACCEPT

(b) DISPLAY

(c) WRITE to a printer within a FORM-OVERFLOW procedure.

NOTE: At least one DISPLAY to SYSLST must be performed in the non-declarative portion of the PROCEDURE DIVISION before any are performed with the DECLARATIVE portion. ACCEPTs from SYSCONSOLE or the job stream are not permitted inside a USE for LABEL PROCEDURE.

(8) CALL and ENTRY statements are not allowed within USE procedures.

(9) Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the non-declarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative or to the procedures associated with such a USE declarative.

6.7.34. WRITE

Function:

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of the printer. The WRITE statement permits performance of a specified imperative statement if the contents of the associated KEYS are found to be invalid.

Formats:

Format 1:

WRITE *record-name* [FROM *identifier-1*]

$$\left[\left\{\begin{matrix}\text{BEFORE}\\\text{AFTER}\end{matrix}\right\}\text{ADVANCING}\left\{\begin{matrix}\textit{identifier-2}\ \text{LINES}\\\textit{integer}\ \text{LINES}\\\textit{mnemonic-name}\end{matrix}\right\}\right]$$

Format 2:

WRITE *record-name* [FROM *identifier-1*] ; INVALID KEY *imperative-statement*

Rules:

(1) A file must be OPENed prior to execution of the first WRITE statement for that file.

(2) The record-name is the name of a logical record in the File Section of the Data Division and must not be part of a sort file.

(3) When the FROM option is used, data is moved from identifier-1 to record-name according to the rules specified for the MOVE verb without the CORRESPONDING option.

(4) After the WRITE is executed, information in record-name is no longer available, but identifier-1 information is available.

The record area associated with an output file may not be accessed prior to the open for that file. (After a WRITE is executed the record is no longer available.)

(5) The INVALID KEY option in format 2 is used for processing direct access files.

(a) In sequential access mode, the imperative-statement is executed when the end of the last segment of the file is reached and an attempt is made to execute a WRITE for that file.

(b) In random access mode, the WRITE performs the function of a SEEK statement prior to writing. The imperative-statement is executed when the contents of the KEY(s) being used to locate the record is found to be invalid.

(6) The ADVANCING option allows control of the vertical positioning of each record on the printed page. If this option is omitted for a printer file, the printer will automatically advance one line before printing. Any form of the ADVANCING option overrides this automatic advance.

(a) The content of identifier-2 and the value of integer must not exceed 127. A value of 0 is permissible (where overprinting is desired).

(b) Mnemonic-name specifies a channel in the forms control paper tape loop. This channel is identified in the SPECIAL-NAMES paragraph of the Environment Division, using SYSCHAN-t IS mnemonic-name, where t is the channel.

(7) The USE FOR FORM-OVERFLOW clause in the Declaratives portion of the Procedure Division permits the programmer to perform special procedures when a form overflow condition exists. Form overflow is detected during the print and space functions of the printer. If form positioning by means of paper tape loop is specified (ADVANCING mnemonic-name), form overflow condition does not occur.

(8) Printer speed can be increased by as much as 100% in printer-bound COBOL object programs if the WRITE statement to the printer is:

WRITE RECORD-NAME [FROM *identifier*]

BEFORE ADVANCING *integer* LINES

This statement allows printing and advancing to be done with one command to the printer.

Because of the nature of the commands which the printer hardware can accept, if a WRITE AFTER ADVANCING is used, object code will be generated causing two commands to the printer: the first to space the printer form, and the second to perform the printing. Note that the COBOL WRITE statement without an ADVANCING option specified is defined to be a WRITE AFTER ADVANCING 1 LINE. Therefore, the WRITE statement without an ADVANCING option results in the slower printer speed.

7. SEGMENTATION

7.1. GENERAL

Segmentation provides a method of communication with the compiler to specify object program overlay requirements. Since UNIVAC 9400 COBOL deals just with segmentation of procedures, only the Procedure Division is considered in determining segmentation requirements for an object program.

7.2. PROGRAM SEGMENTS

When segmentation is used, it is mandatory that the Procedure Division be written in sections. Each section must be classified as belonging to either the fixed portion or to one of the independent segments of the object program. Segmentation does not negate the need to qualify procedure-names in order to ensure uniqueness.

7.2.1. Fixed Portion

In the basic compiler, the fixed portion is that part of the object program which is always in memory. This portion of the program cannot be overlaid by any other part of the program.

In the extended compiler, the fixed portion is that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments, the fixed permanent segment and the fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program. A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause (see 4.2.2). Such a segment, if called for by the program, is always made available in its last used state.

In the basic compiler, an implicit SEGMENT-LIMIT of 50 is always in effect. If the SEGMENT-LIMIT clause is not specified in the extended compiler, an implicit SEGMENT-LIMIT of 50 is also in effect.

7.2.2. Independent Segments

In the basic compiler, an independent segment is a part of the object program which can overlay, and can be overlaid by another independent segment. An independent segment is in its initial state whenever control is transferred to that segment from a segment with a different priority-number.

In the extended compiler, an independent segment is part of the object program which can overlay, and can be overlaid by either a fixed overlayable segment or another independent segment. An independent segment is in its initial state whenever control is transferred to that segment from a segment with a different priority-number.

7.3. SECTION

Function:

Segment classification is accomplished by means of a system of priority-numbers which are included in the section header.

Format:

section-name SECTION [*priority-number*].

Rules:

(1) The priority-number must be an integer ranging in value from 0 through 99.

(2) If priority-number is omitted from the section header, the priority is assumed to be 0.

(3) Sections in the Declaratives must not contain priority-numbers in their section headers.

(4) The logical sequence of the object program execution is the same as the physical sequence of the source program except for specific user supplied transfers of control. Sections with the same explicit or implicit priority-number, however, physically comprise a single object program segment.

(5) In the basic compiler, sections with priority-number 0 through 49 constitute the fixed permanent segment of the object program. Sections with priority-number 50 through 99 constitute independent segments. All sections with the same priority-number must appear together in the source program.

In the extended compiler, sections with priority-number 0 up to, but not including, the SEGMENT-LIMIT priority-number constitute the fixed permanent segment of the object program. Sections with priority-numbers ranging from the SEGMENT-LIMIT to 49 are fixed overlayable segments. Sections with priority-number 50 through 99 constitute independent segments. Sections with the same priority-number need not be grouped together in the source program.

7.4. RESTRICTIONS

When Segmentation is used, the following restrictions are placed on the ALTER and PERFORM statements.

7.4.1. The ALTER Statement

Any GO TO statement in a fixed segment (priority-number 49 or less) can be ALTERed by an ALTER statement located in any other segment of the program. A GO TO statement in an independent segment (priority-number 50 or greater) can only be ALTERed by an ALTER statement located in the same segment as the GO TO statement.

7.4.2. The PERFORM Statement

(1) A PERFORM statement that appears in a section with a priority-number less than the implicit or explicit SEGMENT-LIMIT priority-number can have, within its range only the following:

(a) Sections with a priority of less than 50.

(b) Sections wholly contained in a single segment whose priority-number is greater than 49.

(2) A PERFORM statement that appears in a section with a priority-number equal to or greater than the implicit or explicit SEGMENT-LIMIT priority-number can have, within its range, only the following:

(a) Sections with the same priority-number as that containing the PERFORM statement.

(b) Sections with a priority-number less than the implicit or explicit SEGMENT-LIMIT priority-number.

8. TABLE HANDLING

8.1. GENERAL

The table handling module provides a means of defining contiguous data items in a tabular form, thereby permitting easy access to any item regardless of its position in the table.

This section contains the methods of table definition and referencing that are available to the UNIVAC 9400 COBOL user. For a complete discussion of table handling see *Fundamentals of COBOL – Table Handling, UP-7503.2* (current version).

8.2. DEFINING A TABLE

Each data item in a table (called a table element) must be the subject of an OCCURS clause in the data description. This clause specifies the number of times that the table element appears in the table.

To define a one-dimensional table, an OCCURS clause is written as a part of the data description for the repeated item. Any practical number of occurrences may be specified (see 5.3.3 rule 4).

Defining a one-dimensional table within each occurrence of a table element gives rise to a two-dimensional table. This is done by writing an OCCURS clause for a data item subordinate (i.e., having numerically larger level number) to another item for which an OCCURS clause was written. Tables with up to three dimensions can be defined in this manner in UNIVAC 9400 COBOL. Each dimension must be defined by an OCCURS clause, and must be defined on a different hierarchal level.

The OCCURS clause is fully explained in 5.3.3 of this manual.

8.3. TABLE REFERENCE

To reference a table element, it is necessary to specify which occurrence of the table element is intended.

Occurrence numbers are specified by one of two methods: subscripting or indexing. In either method, the reference is made by immediately following the data-name with a set of occurrence specifications (subscripts or index-names) enclosed in parentheses.

Up to three subscript or index levels may appear in the reference, depending upon the number of dimensions involved. There must be one subscript or index-level for each OCCURS clause in the defined hierarchy which contains the element name, including the one for the element name. Multiple subscripts and index-names are written left to right in descending order of inclusiveness.

8.4. SUBSCRIPTING

Definition:

An integer value which specifies the occurrence number of a table element is a subscript.

Format:

data-name (subscript-1 [, *subscript-2* [, *subscript-3*]])

Rules:

(1) The subscript value must be a positive or unsigned integer and may be represented as a numeric literal or as a data-name defined elsewhere as an elementary numeric data item with no character positions to the right of the assumed decimal point. Data-name subscripts may be mixed with numeric literal subscripts within a reference.

(2) The lowest valid subscript is 1; the highest valid subscript is the number of item occurrences specified in the OCCURS clause. The area allocated, multiplied by the number of occurrences cannot exceed 65,535.

(3) References are made to individual items within a table of homogeneous elements by specifying the name of the table, followed by one or more spaces, followed by its related subscript(s) in parentheses. A left parenthesis may not be followed by a space, nor may a right parenthesis be preceded by a space.

8.5. INDEXING

Definition:

A technique used to refer to individual table elements within a table of like elements that have not been assigned individual data-names. An index-name contains the occurrence number of a table element which can be used for:

- Direct indexing by using the index-name as a subscript.
- Relative indexing by appending to the index-name the + or – operator followed by an unsigned integer.

Format:

data-name (index-name-1 [{±} *integer-1*]

[, *index-name-2* [{±} *integer-2*]]

[, *index-name-3* [{±} *integer-3*]])

Rules:

(1) Index-names are defined by use of the INDEXED BY option in the OCCURS clause. Further data description is not used because allocation and format are hardware-dependent. The index-name may be used only in reference to the table element described by the OCCURS clause or to one of its subordinate items.

(2) Index-names are initialized and modified in the object program by means of the SET statement.

(3) References are made to individual items within a table of homogeneous elements by specifying the name of the table element, followed by its related index-name(s) in parentheses.

(4) A data item in a file can be described by a USAGE IS INDEX clause. This data item value can then be transferred to an index-name without conversion, by means of the SET statement.

8.6. SEARCHING

Definition:

Data that has been arranged in the form of a table is often searched. In COBOL the SEARCH statement provides facilities, through its two options, for producing serial and nonserial (binary) searches. In using the SEARCH statement the programmer may vary an associated index-name or an associated data-name. This statement also provides facilities for execution of imperative statements when certain conditions are true and includes an AT END phrase (see 6.7.26).

9. SORTING

9.1. GENERAL

The UNIVAC 9400 COBOL SORT feature offers the user an efficient means of sorting records against a set of specified keys in addition to a variety of processing considerations, such as adding or deleting records, or modification of records within the file. Sorting is included in the extended compiler only.

9.2. ORGANIZATION OF A SORT PROGRAM

A sort file, like any other file, is a set of records. It is described in the Data Division by a special type of File Description called a Sort File Description. The sort file may be thought of as an internally contained intermediate representation of the file, following the initial input of unsorted records and preceding the final output of sorted records.

A COBOL program may contain any number of sort operations. In general, a sort operation proceeds as follows:

(1) Control passes to a SORT statement. The SORT statement specifies the sort file to be created and the data keys that guide the sort operation. It either identifies the input procedure and output procedure or names the source of the unsorted input records and those files which are to receive the sorted output records.

(2) The input procedure, if named in the SORT statement, is executed. This input procedure must contain at least one RELEASE statement. If no INPUT PROCEDURE is specified, the input file is named in the USING option of the SORT statement. The effect of either option is to make input records available to the sort operation.

(3) The records made available to the sort operation are sorted on a set of specified keys as shown in the KEY clause.

(4) The SORT statement passes control to the output procedure, if one is named. The output procedure must contain at least one RETURN statement, the effect of which is to move sorted records from the sort file to the output file. If no output procedure is used, the GIVING option must specify the output file.

(5) The operation of the SORT statement is terminated and control passes to the next statement in sequence.

When the input procedure or an output procedure is in control, all transfers of control must refer to procedures contained within that input procedure or output procedure. Conversely, control cannot be transferred into an input or output procedure from points in the Procedure Division outside the physical limits of that given input or output procedure. Neither an input nor an output procedure may contain a SORT statement.

For a detailed discussion of COBOL sorting consult *Fundamentals of COBOL – Sorting, UP-7503.3* (current version).

9.3. SORT STATEMENT FORMATS

The following paragraphs summarize the statements used in UNIVAC 9400 COBOL sorts.

9.3.1. Sort File Select Statement

Function:

The SELECT statement is used to name the sort file and to identify the hardware storage medium used during the sorting process.

Format:

SELECT *filename* ASSIGN TO [*external-name*] [*integer-1*] *implementor-name-1* [OR *implementor-name-2*]

Rules:

(1) The SELECT statement is discussed in detail on 4.3.1.

(2) The external name is not required in the sort file SELECT statement, as fixed external names are used.

(3) Tape or disc are the only applicable devices for a sort file. Note that regardless of which device is specified, the temporary storage medium used is determined at execution time using the external name (SM01, SM02, SM03,..., SM06 for tape; DM01,..., DM08 for disc).

(4) The optional OR clause serves only as documentation since the actual temporary medium is determined at execution time through the job control stream.

9.3.2. Sort File Description

Function:

The Sort File Description (SD) defines the structure of the file to be sorted.

Format:

SD *file-name*

[; RECORD CONTAINS [*integer-5* TO] *integer-6* CHARACTERS]

$$\left[; \text{RECORDING MODE IS} \left\{ \begin{matrix} \text{F} \\ \text{V} \\ \text{D} \end{matrix} \right\} \right]$$

$$\left[; \text{DATA} \left\{ \begin{matrix} \text{RECORD IS} \\ \text{RECORDS ARE} \end{matrix} \right\} \textit{data-name-1} \; [, \textit{data-name-2}] \ldots \right]$$

Rules:

Paragraph 5.2.2 lists the rules applicable to this statement.

9.3.3. RELEASE

Function:

RELEASE is used in the input procedure of a SORT statement to transfer records to the initial phase of a sort operation.

Format:

RELEASE *record-name* [FROM *identifier*]

Rules:

This statement is discussed in detail in 6.7.23.

9.3.4. RETURN

Function:

RETURN is used in the output procedure of a SORT statement to obtain sorted records from the final phase of a sort operation.

Format:

RETURN *file-name* RECORD [INTO *identifier*] AT END *imperative-statement*

Rules:

This statement is discussed in detail in paragraph 6.7.24.

9.3.5. SORT

Function:

The SORT statement controls the creation of the sort file by specifying the means of input, the sorting keys, and the means of output.

Format:

SORT *file-name-1* ON { DESCENDING / ASCENDING } KEY *data-name-1* [, *data-name-2*] . . .

[; ON { DESCENDING / ASCENDING } KEY [*data-name-3*]. . .] . . .

{ INPUT PROCEDURE IS *section-name-1* [THRU *section-name-2*] / USING *file-name-2* }

{ OUTPUT PROCEDURE IS *section-name-3* [THRU *section-name-4*] / GIVING *file-name-3* }

Rules:

The rules governing this format are discussed in detail in 6.7.29.

9.3.6. Use of Sort Feature in the Extended Compiler

The extended compiler provides the COBOL user with the following optional sort features:

- The 9400 COBOL compiler generates linkage code to the standard system sort/merge facilities for all SORT statements. A detailed description of the sort/merge facility is available in *UNIVAC 9400 System Sort/Merge Programmers Reference, UP-7664* (current version).

- The following information is provided to clarify the various options of the sort/merge available to the COBOL programmer. COBOL object programs normally utilize the small volume sort facilities of the sort/merge program; therefore, no operator intervention (except console typeins) is required. The programmer may, however, request the use of the large volume sort facilities. Internal, tape-only, disc-only, and tape/disc sorts are all possible, depending on record volumes and environment.

  (1) Record Size

  The maximum COBOL record size of 4092 characters may be sorted.

  (2) Record Format

  Record format may be fixed (F) or variable (V). For variable format, the BIN size (subrecord size used for internal sort purposes, see *UNIVAC 9400 System Sort/Merge Programmers Reference, UP-7664* (current version)) specified by the program will be the size of the smallest record described in the sort description (SD).

  NOTE: If the USING/GIVING option of the SORT statement is used, the record format of the USING/GIVING files must agree with the SD record format.

  (3) Storage Allocation

  The compiler ensures that the object program contains the minimum storage required for sorting by including in the generated output module a RESERVE linker control statement. Linking the compiler output then produces an object program which includes an area reserved to satisfy the minimum sort needs. This area is referenced within the object program modules by an external reference (EXTRN) to label KE$ALP.

  NOTE: If the programmer inhibits the compiler generation of linker control statements (Option OUT=L), he must satisfy this area requirement in constructing linker control specifications.

  The method employed by the compiler ensures that the sort area is the last storage associated with the object program. If the programmer allocates additional storage for program execution (RUN command parameter), all storage from KE$ALP to the end of the program storage is used by the sort/merge for internal processing. Additional storage can greatly increase the efficiency of the sort operation and should be allocated when possible.

(4) Device Allocation

If the storage allocated for sorting is not adequate to allow an internal sort, external devices must be allocated for intermediate storage. Magnetic tape and/or disc devices may be assigned for this purpose through job control statements. Tapes are assigned using fixed sort filenames of SM01, SM02, ..., SM06. If tapes are assigned, a minimum of three are required, and a maximum of six may be used. Disc devices (maximum of 32, four per filename), which must contain SYSPOOL area, are assigned usign fixed sort filenames of DM01, DM02,..., DM08.

(5) Control Stream Parameters

When the sort is executed by the object program, the presence of control stream parameters are questioned using a console typeout. This feature allows the programmer to override the parameters specified in the object program (i.e., sharing of tape devices with small volume sorts, request for automatic execution of large volume sort, etc.). For details, see *UNIVAC 9400 System Sort/Merge Programmers Reference, UP-7664* (current version).

(6) Multiple Sorts

The 9400 COBOL compiler does not restrict the programmer from using multiple SORT statements which refer to the same SD (sort file description), or using multiple SORT statements which refer to different sort descriptions. Only one SORT statement may be active at any one time.

NOTE: A SORT statement may not appear in an input or output procedure of another SORT statement. If an object program attempts to execute a sort during a previous sort operation, a console message is displayed.

10. LIBRARY

10.1. GENERAL

The Library module provides the capability of specifying text that is to be copied from the COBOL Library which contains text that is available to a source program at compile time. The effect of the compilation of library text is the same as if the text were actually written as part of the source program. COBOL Library text is placed in the COBOL Library as a function independent of the COBOL program.

This section contains library information as applicable to the UNIVAC 9400 COBOL user. For a complete discussion of the COBOL Library module see *Fundamentals of COBOL – Language UP-7503.1* (current version).

10.2. USING THE COPY STATEMENT

The COBOL Library may contain text for the Environment Division, the Data Division and the Procedure Division available through the use of the COPY statement. The rules for the COPY statement are given in 6.7.7 of this manual.

The COPY statement may be written in any of the following forms:

In the Environment Division:

SOURCE-COMPUTER. COPY statement.
OBJECT-COMPUTER. COPY statement.
SPECIAL-NAMES. COPY statement.
FILE-CONTROL. COPY statement.
I-O-CONTROL. COPY statement.

In the File Section:

FD file-name COPY statement.
SD sort-file-name COPY statement.
01 data-name COPY statement.

In the Working-Storage Section:

01 data-name COPY statement.

In the Procedure Division:

{paragraph-name. / section-name SECTION [priority-number].} COPY statement.

In addition to referencing the library module through the COPY statement, the programmer must define the device and file which contain the library module in his job control stream. The LFD name given to this file must also be present on a param card with keyword LIN.

The compiler performs no editing of library modules. Whatever is contained in the library under the specified library-name is copied into the program. Lines of code taken from the library are marked with a C to the right of the line number on the source listing if the text is copied without replacement. Lines of code which have one or more words of text replaced are marked with an R.

Example:

If a COBOL program contains the following lines of code:

```
FILE SECTION.
FD FILE01 COPY LIB-FD01 REPLACING DN-1 BY TAX-A.
01  TAX-A.
      .
      .
      .
```

and the assigned library file contains a module named LIB-FD01 with the lines:

```
LABEL RECORDS ARE STANDARD
BLOCK CONTAINS 1 RECORD
DATA RECORD IS DN-1.
```

at compile time the source listing would be as follows:

```
LINE NO.              SOURCE STATEMENT

   .                    .          .
   .                    .          .
   .                    .          .
00033              FILE SECTION.
00034              FD  FILE01 COPY LIB-FD01 REPLACING DN-1 BY TAX-A
00035 C                LABEL RECORDS ARE STANDARD
00036 C                BLOCK CONTAINS 1 RECORD
00037 R                DATA RECORD IS DN-1
00038              01  TAX-A.
                       .          .
                       .          .
                       .          .
```

The effect on the program is the same as if the programmer had written:

```
FILE SECTION.
FD  FILE01
    LABEL RECORDS ARE STANDARD
    BLOCK CONTAINS 1 RECORD.
    DATA-RECORD IS TAX-A.
01  TAX-A.
      .      .      .
      .      .      .
      .      .      .
```

PARAM cards for use with the COPY statement are defined in Appendix C.

APPENDIX A. UNIVAC 9400 SYSTEM CHARACTER SET

A.1. GENERAL

Table A–1 shows the character set of the UNIVAC 9400 System in collating sequence.

CHARACTER CODES

DECIMAL	HEXA-DECIMAL	EBCDIC	80-COLUMN CARD CODE	CONSOLE KEYBOARD CODE (EBCDIC)	7-TRACK TAPE (BCDIC)	COMPRESSED CARD CODE ①
0	00	NUL	12-0-9-8-1			NO PUNCH
1	01		12-9-1			3 ¦ 12
2	02		12-9-2			4 ¦ 11
3	03		12-9-3			1 ¦ 12,11
4	04	PF	12-9-4			5 ¦ 0
5	05	HT	12-9-5			2 ¦ 12,0
6	06	LC	12-9-6			7 ¦ 11,0
7	07	DEL	12-9-7			6 ¦ 12,11,0
8	08		12-9-8			9 ¦ 8
9	09		12-9-8-1			9,3 ¦ 8,12
10	0A		12-9-8-2			9,4 ¦ 8,11
11	0B		12-9-8-3			9,1 ¦ 8,12,11
12	0C		12-9-8-4			9,5 ¦ 8,0
13	0D		12-9-8-5			9,2 ¦ 8,12,0
14	0E		12-9-8-6			9,7 ¦ 8,11,0
15	0F		12-9-8-7			9,6 ¦ 8,12,11,0
16	10		12-11-9-8-1			↑ ↑
17	11		11-9-1			B B
18	12		11-9-2			I I
19	13		11-9-3			T T
20	14	RES	11-9-4			
21	15	NL	11-9-5	CARR. RET(CR)		
22	16	BS	11-9-6			P P
23	17	IL	11-9-7			O O
24	18		11-9-8			S S
25	19		11-9-8-1			I I
26	1A		11-9-8-2			T T
27	1B		11-9-8-3			I I
28	1C		11-9-8-4			O O
29	1D		11-9-8-5			N N
30	1E		11-9-8-6			S S
31	1F		11-9-8-7			
32	20	DS	11-0-9-8-1			0, 4,
33	21	SOS	0-9-1			1, 5,
34	22	FS	0-9-2			2, 6,
35	23		0-9-3			3. 7
36	24	BYP	0-9-4			
37	25	LF	0-9-5	LINE FEED(LF)		
38	26	EOB	0-9-6			
39	27	PRE	0-9-7			
40	28		0-9-8			
41	29		0-9-8-1			
42	2A	SM	0-9-8-2			
43	2B		0-9-8-3			
44	2C		0-9-8-4			
45	2D		0-9-8-5			
46	2E		0-9-8-6			
47	2F		0-9-8-7			
48	30		12-11-0-9-8-1			
49	31		9-1			
50	32		9-2			
51	33		9-3			
52	34	PN	9-4			
53	35	RS	9-5			
54	36	UC	9-6			
55	37	EOT	9-7	⑤ (EOM)		
56	38		9-8			
57	39		9-8-1			
58	3A		9-8-2			
59	3B		9-8-3			
60	3C		9-8-4			
61	3D		9-8-5			
62	3E		9-8-6			
63	3F		9-8-7			
64	40	SP	NO PUNCHES	SPACE (SP)		
65	41		12-0-9-1			
66	42		12-0-9-2			
67	43		12-0-9-3			
68	44		12-0-9-4			

① Punch patterns used to store the corresponding hexadecimal representation in the indicated bit positions of a byte.

Table A–1. UNIVAC 9400 System Character Set (Part 1 of 4)

DECIMAL	HEXA-DECIMAL	EBCDIC②	80-COLUMN CARD CODE	CONSOLE KEYBOARD CODE (EBCDIC)	7-TRACK TAPE (BCDIC)
69	45		12-0-9-5		
70	46		12-0-9-6		
71	47		12-0-9-7		
72	48		12-0-9-8		
73	49		12-8-1		
74	4A	¢	12-8-2	¢	
75	4B	.	12-8-3	.	B A 8 2 1
76	4C	<	12-8-4	<	B A 8 4
77	4D	(	12-8-5	(	B A 8 4 1
78	4E	+	12-8-6	+	B A 8 4 2
79	4F	| (Vert. Bar)	12-8-7	¦ (Vert. Bar)	B A 8 4 2 1
80	50	&	12	&	B A
81	51		12-11-9-1		
82	52		12-11-9-2		
83	53		12-11-9-3		
84	54		12-11-9-4		
85	55		12-11-9-5		
86	56		12-11-9-6		
87	57		12-11-9-7		
88	58		12-11-9-8		
89	59		11-8-1		
90	5A	!	11-8-2	! or OR	
91	5B	$	11-8-3	$	B 8 2 1
92	5C	*	11-8-4	*	B 8 4
93	5D	)	11-8-5	)	B 8 4 1
94	5E	;	11-8-6	;	B 8 4 2
95	5F	¬ (Not)	11-8-7	¬ (Not)	B 8 4 2 1
96	60	–	11	–	B
97	61	/	0-1	/	A 1
98	62		11-0-9-2		
99	63		11-0-9-3		
100	64		11-0-9-4		
101	65		11-0-9-5		
102	66		11-0-9-6		
103	67		11-0-9-7		
104	68		11-0-9-8		
105	69		0-8-1		
106	6A	\	12-11		
107	6B	, (Comma)	0-8-3	, (Comma)	A 8 2 1
108	6C	%	0-8-4	%	A 8 4
109	6D	_ (Underscore)	0-8-5	– (Underscore)	A 8 4 1
110	6E	>	0-8-6	>	A 8 4 2
111	6F	?	0-8-7	?	A 8 4 2 1
112	70		12-11-0		
113	71		12-11-0-9-1		
114	72		12-11-0-9-2		
115	73		12-11-0-9-3		
116	74		12-11-0-9-4		
117	75		12-11-0-9-5		
118	76		12-11-0-9-6		
119	77		12-11-0-9-7		
120	78		12-11-0-9-8		
121	79		8-1		
122	7A	:	8-2	:	A
123	7B	#	8-3	#	8 2 1
124	7C	@	8-4	@	8 4
125	7D	' (Prime or Apos.)	8-5	' (Prime or Apos.)	8 4 1
126	7E	=	8-6	=	8 4 2
127	7F	" (Quotes)	8-7	" (Quotes)	8 4 2 1
128	80		12-0-8-1		
129	81	a	12-0-1		

② Lower case letters are an industry standard and are not printable on the UNIVAC 9400 Printer.

Table A–1. UNIVAC 9400 System Character Set (Part 2 of 4)

DECIMAL	HEXA-DECIMAL	EBCDIC②	80-COLUMN CARD CODE	CONSOLE KEYBOARD SET (EBCDIC)	7-TRACK TAPE (BCDIC)
130	82	b	12-0-2		
131	83	c	12-0-3		
132	84	d	12-0-4		
133	85	e	12-0-5		
134	86	f	12-0-6		
135	87	g	12-0-7		
136	88	h	12-0-8		
137	89	i	12-0-9		
138	8A		12-0-8-2		
139	8B		12-0-8-3		
140	8C		12-0-8-4		
141	8D		12-0-8-5		
142	8E		12-0-8-6		
143	8F		12-0-8-7		
144	90		12-11-8-1		
145	91	j	12-11-1		
146	92	k	12-11-2		
147	93	l	12-11-3		
148	94	m	12-11-4		
149	95	n	12-11-5		
150	96	o	12-11-6		
151	97	p	12-11-7		
152	98	q	12-11-8		
153	99	r	12-11-9		
154	9A		12-11-8-2		
155	9B		12-11-8-3		
156	9C		12-11-8-4		
157	9D		12-11-8-5		
158	9E		12-11-8-6		
159	9F		12-11-8-7		
160	A0		11-0-8-1		
161	A1		11-0-1		
162	A2	s	11-0-2		
163	A3	t	11-0-3		
164	A4	u	11-0-4		
165	A5	r	11-0-5		
166	A6	w	11-0-6		
167	A7	x	11-0-7		
168	A8	y	11-0-8		
169	A9	z	11-0-9		
170	AA		11-0-8-2		
171	AB		11-0-8-3		
172	AC		11-0-8-4		
173	AD		11-0-8-5		
174	AE		11-0-8-6		
175	AF		11-0-8-7		
176	B0		12-11-0-8-1		
177	B1		12-11-0-1		
178	B2		12-11-0-2		
179	B3		12-11-0-3		
180	B4		12-11-0-4		
181	B5		12-11-0-5		
182	B6		12-11-0-6		
183	B7		12-11-0-7		
184	B8		12-11-0-8		
185	B9		12-11-0-9		
186	BA		12-11-0-8-2		
187	BB		12-11-0-8-3		
188	BC		12-11-0-8-4		
189	BD		12-11-0-8-5		
190	BE		12-11-0-8-6		
191	BF		12-11-0-8-7		

② Lower case letters are an industry standard and are not printable on the UNIVAC 9400 Printer.

Table A–1. UNIVAC 9400 System Character Set (Part 3 of 4)

DECIMAL	HEXA-DECIMAL	EBCDIC	80-COLUMN CARD CODE	CONSOLE KEYBOARD SET (EBCDIC)	7-TRACK TAPE (BCDIC)
192	C0	PZ	12-0		B A 8 2
193	C1	A	12-1	A	B A 1
194	C2	B	12-2	B	B A 2
195	C3	C	12-3	C	B A 2 1
196	C4	D	12-4	D	B A 4
197	C5	E	12-5	E	B A 4 1
198	C6	F	12-6	F	B A 4 2
199	C7	G	12-7	G	B A 4 2 1
200	C8	H	12-8	H	B A 8
201	C9	I	12-9	I	B A 8 1
202	CA		12-0-9-8-2		
203	CB		12-0-9-8-3		
204	CC		12-0-9-8-4		
205	CD		12-0-9-8-5		
206	CE		12-0-9-8-6		
207	CF		12-0-9-8-7		
208	D0	MZ	11-0		B 8 2
209	D1	J	11-1	J	B 1
210	D2	K	11-2	K	B 2
211	D3	L	11-3	L	B 2 1
212	D4	M	11-4	M	B 4
213	D5	N	11-5	N	B 4 1
214	D6	O	11-6	O	B 4 2
215	D7	P	11-7	P	B 4 2 1
216	D8	Q	11-8	Q	B 8
217	D9	R	11-9	R	B 8 1
218	DA		12-11-9-8-2		
219	DB		12-11-9-8-3		
220	DC		12-11-9-8-4		
221	DD		12-11-9-8-5		
222	DE		12-11-9-8-6		
223	DF		12-11-9-8-7		
224	E0		0-8-2		A 8 2
225	E1		11-0-9-1		
226	E2	S	0-2	S	A 2
227	E3	T	0-3	T	A 2 1
228	E4	U	0-4	U	A 4
229	E5	V	0-5	V	A 4 1
230	E6	W	0-6	W	A 4 2
231	E7	X	0-7	X	A 4 2 1
232	E8	Y	0-8	Y	A 8
233	E9	Z	0-9	Z	A 8 1
234	EA		11-0-9-8-2		
235	EB		11-0-9-8-3		
236	EC		11-0-9-8-4		
237	ED		11-0-9-8-5		
238	EE		11-0-9-8-6		
239	EF		11-0-9-8-7		
240	F0	0	0	0	8 2
241	F1	1	1	1	1
242	F2	2	2	2	2
243	F3	3	3	3	2 1
244	F4	4	4	4	4
245	F5	5	5	5	4 1
246	F6	6	6	6	4 2
247	F7	7	7	7	4 2 1
248	F8	8	8	8	8
249	F9	9	9	9	8 1
250	FA		12-11-0-9-8-2		
251	FB		12-11-0-9-8-3		
252	FC		12-11-0-9-8-4		
253	FD		12-11-0-9-8-5		
254	FE		12-11-0-9-8-6		
255	FF		12-11-0-9-8-7		

Table A–1. UNIVAC 9400 System Character Set (Part 4 of 4)

APPENDIX B. UNIVAC 9400 SYSTEM COBOL RESERVED WORDS

B.1. GENERAL

Reserved words are part of the COBOL language structure and cannot be used for data or procedure names.

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASCII
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BLANK
BLOCK
BLOCK-COUNT
BLOCK-LENGTH-CHECK
BUFFER-OFFSET
BY
CALL
CARD-PUNCH
CARD-READER
CARD-READER-51
CARD-READER-66
CHARACTERS
CHANGED
CLOSE
COBOL
COMMA
COMP
COMP-3
COMPUTATIONAL
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONTAINS
COPY
CORR
CORRESPONDING
CURRENCY
CYLINDER-INDEX
CYLINDER-OVERFLOW
DATA
DATE-COMPILED
DATE-WRITTEN
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DIRECT
DISC-8411
DISC-8414
DISC
DISPLAY
DIVIDE
DIVISION
DOWN
EBCDIC
ELSE
END
ENDING
ENTER
ENTRY
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXHIBIT
EXIT
EXTENDED-INSERTION
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILE-PREPARATION
FILLER
FIRST
FOR
FORM-OVERFLOW
FROM
GENERATE
GIVING
GO
GREATER
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICES
INITIATE
INPUT
INPUT-OUTPUT
INSERT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
LABEL
LEADING
LEFT
LESS
LINE
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
MAP
MASTER-INDEX
MEMORY
MODE
MODULES
MONITOR
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY
NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OTHERWISE
OUTPUT
PERCENT
PERFORM
PIC
PICTURE
POSITION
POSITIVE
PRINTER
PROCEDURE
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID
QUOTE
QUOTES
RANDOM
READ
READY
RECORD
RECORDING

RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMAINDER
REMARKS
RENAMES
REPLACING
RERUN
RESET
RESERVE
RESTRICTED
RETURN
REVERSED
REWIND
REWRITE
RIGHT
ROUNDED
RUN
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SEPARATE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
SYMBOLIC
SYNC
SYNCHRONIZED
SYSCHAN-4
SYSCHAN-5
SYSCHAN-6
SYSCHAN-7
SYSCHAN-8
SYSCHAN-9
SYSCHAN-10
SYSCHAN-11
SYSCHAN-12
SYSCHAN-13
SYSCHAN-14
SYSCHAN-15
SYSCOM
SYSCONSOLE
SYSDATE
SYSERR
SYSERR-0
SYSERR-1
SYSERR-2
SYSERR-3
SYSERR-4
SYSERR-5
SYSERR-6
SYSERR-7
SYSERR-8
SYSERR-9
SYSERR-10
SYSERR-11
SYSERR-12
SYSERR-13
SYSERR-14
SYSERR-15
SYSLST
SYSSWCH
SYSSWCH-0
SYSSWCH-1
SYSSWCH-2
SYSSWCH-3
SYSSWCH-4
SYSSWCH-5
SYSSWCH-6
SYSSWCH-7
SYSTIME
TALLY
TALLYING
TAPE
TAPE-6
TAPES
TERMINATE
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TRACE
TRACKS
TRAILING
TRANSFORM
UNEQUAL
UNIT
UNIVAC-9400
UNTIL
UP
UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
VERIFY
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

APPENDIX C. SOURCE AND COPY LIBRARY INPUT SPECIFICATION

C.1. GENERAL

This appendix describes the method of reading a source program from either the control stream or from a tape or a disc library.

The formats for the source and copy library PARAM statements are presented in the following paragraphs.

Source Library Input:

Format:

```
// PARAM IN=program-name/file-name
```

program-name — 1- to 8-character name of source program to be compiled.

file-name — 1- to 8-character name used to identify the file on which the source program resides. This name must appear on the LFD control card used to define the device to the Job Control program.

If the file-name is omitted, the following name will automatically be supplied:

Name when using basic system – SYSRES
Name when using extended system – SORS$

NOTE: When only four tapes are available for compilation, source input should be mounted on SCR2. The compiler console message:

DISMOUNT SCR2. MOUNT A SCRATCH TAPE ON SCR2.

will appear when tape is to be changed.

Copy Library Input:

Format:

```
// PARAM LIN=file-name
```

file-name — 1- to 8-character name used to identify the file on which the COPY library resides. This name must appear on the LFD control card used to define the device to the Job Control program.

If the file-name is omitted, the following name will automatically be supplied:

Name when using basic system – SYSRES
Name when using extended system – COPY$

The COPY element-name is supplied in the source program via the COPY clause.

C.2. BASIC COMPILER SOURCE LIBRARY INPUT AND COPY LIBRARY INPUT

The source program may be read from the job stream or from a tape library. Any copy library modules referenced in the source program may be read from a tape library. Any library structure to be accessed by the compiler must be created by the UNIVAC 9400 tape librarian.

Any library tapes to be accessed by the compiler must be defined in the job stream and on PARAM cards (keyword IN for the source library; LIN for the copy library). If the source program does not reference any copy library modules, the copy library file need not be defined.

Examples:

(1) Source library and copy library:

```
// DVC 10    // VOL SPxxxx    // LFD FILE-NAME-1
// DVC 11    // VOL SPxxxx    // LFD FILE-NAME-2
```

with PARAM cards:

```
// PARAM IN=PROGRAM-NAME/FILE-NAME-1
// PARAM LIN=FILE-NAME-2
```

(2) Source library and copy library on the same tape:

```
// DVC 10    // VOL SPxxxx    // LFD FILE-NAME-1
```

with PARAM cards:

```
// PARAM IN=PROGRAM-NAME/FILE-NAME-1
// PARAM LIN=FILE-NAME-1
```

(3) Source from the job stream and copy library:

```
// DVC 10    // VOL SPxxxx    // LFD FILE-NAME-1
```

with PARAM card:

```
// PARAM LIN=FILE-NAME-1
```

In the above examples, FILE-NAME-1 and FILE-NAME-2 are programmer-supplied names. PROGRAM-NAME is the name of the source library module which contains the source program.

Either the source library, the copy library, or both libraries on the same tape may be defined as SCR2. If so, SCR2 must be assigned in the job stream and on the PARAM cards. When the source program and any referenced copy library modules have been read, a console message is printed instructing the operator to demount the library tape on SCR2 and mount a scratch tape.

C.3. EXTENDED COMPILER SOURCE LIBRARY INPUT AND COPY LIBRARY INPUT

The source program may be read from the job stream, a tape library, or a disc library. Any copy library modules which are referenced by the source program may be read from a tape library or a disc library. Any library structures to be accessed by the compiler must have been created by the UNIVAC 9400 tape or disc librarian.

Any library structures to be accessed by the compiler must be defined in the job stream and their LFD names must appear on PARAM cards (keyword IN for the source library; LIN for the copy library). If there are no copy library modules referenced by the source program, the copy library need not be defined.

Examples:

(1) Source and copy library on disc:

```
// DVC 20   // VOL DSPxxx
// LBL FILE-ID-1,DSPxxx     // LFD FILE-NAME-1
// DVC 20   // VOL DSPxxx
// LBL FILE-ID-2,DSPxxx     // LFD FILE-NAME-2
```

with PARAM cards:

```
// PARAM IN=PROGRAM-NAME/FILE-NAME-1
// PARAM LIN=FILE-NAME-2
```

(2) Source library tape and copy library disc:

```
// DVC 10   // VOL SPxxxx     // LFD FILE-NAME-1
// DVC 20   // VOL DSPxxx
// LBL FILE-ID-1,DSPxxx   // LFD FILE-NAME-2
```

with PARAM cards:

```
// PARAM IN=PROGRAM-NAME/FILE-NAME-1
// PARAM LIN=FILE-NAME-2
```

(3) Source from the job stream and copy library tape:

```
// DVC 10  // VOL SPxxxx   // LFD FILE-NAME-1
```

with PARAM card:

```
// PARAM LIN=FILE-NAME-1
```

(4) Source and copy from the same library tape:

```
// DVC 10   // VOL SPxxxx   // LFD FILE-NAME-1
```

with PARAM cards:

```
// PARAM IN=PROGRAM-NAME/FILE-NAME-1
// PARAM LIN=FILE-NAME-1
```

In the above examples, FILE-NAME-1 and FILE-NAME-2 are programmer-supplied names. FILE-ID-1 and FILE-ID-2 are file-id names that were used at the time the disc library was created. PROGRAM-NAME is the name of the source library module that contains the source program.

APPENDIX D. 9400 COBOL PROCESSING TECHNIQUES FOR DIRECT ACCESS DEVICES

D.1. INTRODUCTION

This appendix describes the different processing techniques available to the 9400 COBOL programmer for processing files assigned to direct access devices. The technique chosen to process a particular file is dependant upon the file organization and the manner in which the file is to be accessed. Each processing technique has its particular advantages and disadvantages. In selecting a processing technique, one must consider such factors as device characteristics, file size, file activity, file growth potential, etc. For more information, see *UNIVAC 9200 II/9300 II/9400 Systems P.I.E. 8411 Disc File Direct Access Subsystem Concepts, UP-7704.1* (current version).

D.2. FILE ORGANIZATION

A file organization is specified by the ORGANIZATION clause in the SELECT statement for the file. UNIVAC 9400 COBOL provides four classes of file organization:

(1) sequential

(2) relative

(3) direct

(4) indexed

Each file organization requires a specific number and kind of control field (KEY) to locate records within the file. Due to the use of keys, no two file organizations are compatible. For example, a sequentially organized file cannot be read if described as a file with organization indexed.

D.2.1. ORGANIZATION IS SEQUENTIAL

When sequential file organization is used, the logical record of a file will be recorded sequentially in order of their creation. Records in a sequential file are read in the order they were written.

D.2.2. ORGANIZATION IS RELATIVE

Relative file organization is characterized by the use of relative record addressing to locate logical records within the file. Prior to the actual creation of a file with relative organization, the entire file must be initialized so that it contains a physical record in every possible record position (see D.5.2). The address of any logical record in the file is determined by its position relative to the first record of the file. A unique data item, defined by the ACTUAL or RELATIVE KEY clause, is used to specify the RELATIVE record number being processed. The RELATIVE record number contained in the key area is converted by Data Management programs into the physical record address.

D.2.3. ORGANIZATION IS DIRECT

Direct file organization is characterized by the use of relative track addressing and record identification keys to locate logical records within the file. Prior to the actual creation of a file with direct organization, the entire file area must be initialized such that every track is erased and its capacity record is normalized (see D.5.2). The address of any logical track in the file is determined by its position relative to the first track of the file. A unique data item, defined by the ACTUAL KEY clause, is used to specify the logical track number being processed. The RELATIVE track number contained in the key area is converted by Data Management programs into the physical track address. Another data item, defined by the SYMBOLIC KEY clause, is used to identify the desired record within the RELATIVE track.

D.2.4. ORGANIZATION IS INDEXED

When indexed file organization is used, the logical records of the file are initially recorded (loaded) sequentially into the prime data area of the file. Each logical record must contain an embedded record identification field, defined by the RECORD KEY clause. All records presented for loading must have their RECORD KEY value in presorted ascending sequence. During file load, a hierarchy of indexes, based on the value of each RECORD KEY and the record's physical address, is created in the index areas of the file. These indexes are used, in subsequent file processing, to physically locate the referenced record. When ACCESS IS RANDOM, a unique data item, defined by the SYMBOLIC KEY clause, is used to specify the RECORD KEY value of the record being processed.

D.3. FILE ACCESS MODES

The ACCESS MODE clause defines the order in which data may be transferred to or from the direct access storage device. The two modes of accessing data are:

ACCESS IS SEQUENTIAL

or

ACCESS IS RANDOM

- Sequential Access

  The reading and writing of records in a file is done in a sequential (serial) manner. The order of referencing records is implicitly determined by the physical position of the records in the file.

- Random Access

  Permits the reading and writing of records in a file in the order specified by the COBOL programmer through keys associated with the file.

D.4. FILE PROCESSING TYPES

The combination of the ORGANIZATION and ACCESS MODE clauses defines a file processing technique. The following chart outlines the seven types of file processing techniques supported by 9400 COBOL and their relationship to file organization and record access mode.

ACCESS MODE	FILE ORGANIZATION			
	SEQUENTIAL	RELATIVE	DIRECT	INDEXED
SEQUENTIAL	type 1	type 2	type 3	type 6
RANDOM		type 4	type 5	type 7

Processing type 1 is assumed when the ORGANIZATION clause is omitted and the ACCESS MODE clause is either SEQUENTIAL or omitted.

Processing type 4 is assumed when the ORGANIZATION clause is omitted and the ACCESS MODE clause is RANDOM.

D.5. KEY CLAUSE AND USAGE

The following paragraphs describe the key clauses and their use.

D.5.1. ORGANIZATION Clause

(1) ORGANIZATION IS SEQUENTIAL

No keys are required for this file organization.

(2) ORGANIZATION IS RELATIVE

The ACTUAL KEY or RELATIVE KEY clause is required and is used to contain a relative record number. The logical records of a relative file are addressed by their logical position relative to the start of the file. The relative record number is reported in ACTUAL or RELATIVE KEY clauses after a READ or WRITE when ACCESS IS SEQUENTIAL. If ACCESS IS RANDOM, the value specified in the key by the programmer is used to determine the physical address of that logical record. ACTUAL KEY and RELATIVE KEY are synonymous. Whichever key is used must be defined as an unsigned integer according to the rules for numeric items. However, no more than seven digits are used to represent the relative record number. The ACTUAL or RELATIVE KEY may not be defined as part of the data description of the file to which it is associated.

(3) ORGANIZATION IS DIRECT

Both ACTUAL KEY and SYMBOLIC KEY clauses are required to address records. The ACTUAL KEY clause contains a relative track number. The relative track number is reported in the ACTUAL KEY clause after a READ or WRITE when ACCESS IS SEQUENTIAL. When ACCESS IS RANDOM, the value specified in the ACTUAL KEY clause by the programmer is used to determine the physical address of the logical track number. ACTUAL KEY must be defined as an unsigned integer according to the rules for numeric items. However, no more than seven digits are used to represent the relative track number. The SYMBOLIC KEY clause contains a record identifier used to distinguish each record on a track or series of tracks. from all other records on a track or series of tracks. The record identifier is re reported in SYMBOLIC KEY clause after a READ when ACCESS IS SEQUENTIAL.

When ACCESS IS RANDOM, the record identifier specified in the SYMBOLIC KEY clause by the programmer is used as a search argument to locate the record. The SYMBOLIC KEY must be defined as an alphabetic, alphanumeric, or numeric item with a length not greater than 255 characters. The ACTUAL or SYMBOLIC KEY may not be defined as part of the data description of the file to which it is associated.

(4) ORGANIZATION IS INDEXED

The RECORD KEY clause is required and is used as a record identifier to distinguish each record in the file from all other records in the file. The data area defined by the RECORD KEY clause must exist within the record area described for the file.

The RECORD KEY clause must appear in each record description and its displacement in each, relative to the first character of the record, must be the same.

The record identifier is delivered in the RECORD KEY clause, as part of the record, whenever a record is read. The RECORD KEY must be described as an alphabetic, alphanumeric, or numeric item with a length of 3 to 255 character positions. The use of SYMBOLIC KEY is optional when ACCESS IS SEQUENTIAL, and required when ACCESS IS RANDOM. When ACCESS IS SEQUENTIAL, the SYMBOLIC KEY may be used to specify a RECORD KEY value to be used as the starting point of a processing sequence. When ACCESS IS RANDOM, the SYMBOLIC KEY value specified by the programmer is used as a search argument to locate the record with the corresponding RECORD KEY value. When the SYMBOLIC KEY clause is used, it must not be defined as part of the record, but its description must be identical to the description of the file's RECORD KEY.

D.5.2. APPLY Clause

UNIVAC 9400 COBOL offers the user various APPLY clauses which permit specified input-output techniques to be applied to selected files. Since the APPLY clauses are related to file organization, they are discussed on that basis.

(1) ORGANIZATION IS SEQUENTIAL

APPLY VERIFY ON file-name ...

The APPLY VERIFY clause directs that each record written to the file is to be check-read. If specified for a file that is only OPEN INPUT, the clause has no effect.

(2) ORGANIZATION IS RELATIVE

APPLY VERIFY ON file-name ...

Same as for sequential organization.

APPLY FILE-PREPARATION ON file-name ...

Before a record can be physically written in a relative organized file, a pre-recorded record, of the same length as the record being written, must already exist on the device. The APPLY FILE-PREPARATION clause is used to create a dummy record in each relative record area in the file prior to writing the first real record. A dummy record contains HIGH-VALUE in the first character position, followed by LOW-VALUE in each remaining character position, for a maximum of 256 character positions. File-preparation may be omitted if the file presently exists in prepared form, having either been previously created or prepared by other means.

(3) ORGANIZATION IS DIRECT

APPLY VERIFY ON file-name ...

Same as for sequential organization.

APPLY FILE-PREPARATION ON file-name ...

Before a direct organization file can be created, obsolete data previously recorded in the physical file area must be erased. The APPLY FILE-PREPARATION clause is used to erase each track and write a "record-zero," or capacity record, on each track of the file. Record zero reflects space, in bytes, remaining on the track for additional records. File preparation takes place when the file is OPENed for OUTPUT. Once a direct organized file has been created, and file preparation is not employed, any and all logical records written to the file will be placed after the current records already on the track specified. If the file is not OPENed for OUTPUT, this clause has no effect.

APPLY RESTRICTED SEARCH OF integer TRACKS ON file-name ...

This clause is permitted only if ACCESS MODE IS RANDOM. It defines the number of tracks to be searched in order to locate a requested record. If this clause is not applied to the file, ALL TRACKS is assumed to be the search integer.

(4) ORGANIZATION IS INDEXED

APPLY VERIFY ON file-name ...

Same as for sequential organization.

APPLY MASTER-INDEX ON file-name ...

This clause is meaningful only when the file is being initially created (loaded). Its use results in the generation of a master cylinder index for the file.

APPLY CYLINDER-OVERFLOW AREA OF integer PERCENT ON file-name ...

This clause is meaningful only when the file is being initially created (loaded). Its use reserves, in each cylinder of the prime data area, the specified percentage of tracks to be used for cylinder overflow area. If the clause is not applied to the file, 20 percent of each cylinder will be reserved for cylinder overflow area. This represents two tracks on a UNIVAC 8411 Disc Subsystem, and four tracks on a UNIVAC 8414 Disc Subsystem. If no cylinder overflow area is desired, specify 0 PERCENT in the APPLY clause.

APPLY CYLINDER-INDEX AREA OF integer INDICES ON file-name ...

This clause is meaningful only if ACCESS MODE IS RANDOM. Its use reserves, in object time memory, an area sufficient to contain the specified number of cylinder index entries. One cylinder index entry exists for each cylinder in the prime data area. (The specified integer is incremented by three to allow for control fields required by indexed sequential Data Management).

APPLY EXTENDED-INSERTION AREA ON file-name ...

This clause is meaningful only if ACCESS MODE IS RANDOM and the INSERT verb is being used. This clause reserves, in object time memory, an I-O area large enough to contain all data recorded on a prime data track.

D.6. FILE PROCESSING TECHNIQUES

D.6.1. ORGANIZATION IS SEQUENTIAL

File processing type 1 is the only processing technique available when ORGANIZATION IS SEQUENTIAL. Keys can not be associated with type 1 files.

A file that is created with sequential organization must be referred to only with sequential organization.

Type 1 allows three record formats: RECORDING MODE IS U, V, or F. With record format F or V, records may be blocked or unblocked.

File preparation is not necessary with sequential organization.

- Output file

  An OPEN OUTPUT statement prepares the file for output operation. Standard labels are written. The USE for BEGINNING LABEL procedure is executed if user labels are specified.

  A WRITE statement causes the specified record to be written into the next sequential area of the file after blocking, as required, is effected.

  A CLOSE statement causes orderly termination of file processing.

- Input file

  An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked and user labels, if specified, are made available to the USE for BEGINNING LABEL procedure.

  A READ statement causes the next sequential record in the file to be made available after deblocking, as required, is effected.

  A CLOSE statement causes orderly termination of file processing.

- I/O file

  An OPEN for I-O statement prepares the file for input and output operation. Label processing is the same as when OPEN for INPUT. User labels may not be updated.

  The READ statement's operation is the same as when OPENed for INPUT.

  Each WRITE statement must be preceded by a READ statement. The WRITE statement causes the updated record to be rewritten onto the same physical area from which the record was read. Alteration of record length, insertion of new records, or deletion of existing records is not permitted.

The CLOSE statement causes orderly termination of file processing.

D.6.2. ORGANIZATION IS RELATIVE

Two file processing types are available with relative organization: type 2 when ACCESS IS SEQUENTIAL, and type 4 when ACCESS IS RANDOM.

The logical records of a file with relative organization are identified by relative record numbers. The contents of the item specified in the ACTUAL KEY clause indicates the position of the logical record relative to the beginning of the file, starting with the value of 1 for the first record.

A file that is created with relative organization must be referred to only with relative organization.

Relative organization allows only RECORDING MODE IS F. Records cannot be blocked nor can an ALTERNATE AREA be assigned to the file.

Before processing a newly allocated (or newly extended) relative file, each track of the file must be completely filled with physical records that have the same length as the data records to be written on the file. The APPLY FILE-PREPARATION clause may be used to write dummy records on each uninitialized track of the file. File preparation takes place during OPEN processing when the COBOL programmer:

(1) Writes an APPLY FILE-PREPARATION clause for the file in the I-O CONTROL paragraph of the Environment Division.

(2) Sets the value of the ACTUAL KEY data item to one, if the file is new, or to the record number of the first record of the new extent(s) when the file is being extended.

(3) Executes an OPEN OUTPUT statement after the ACTUAL KEY value has been set. File preparation occurs each time an OPEN OUTPUT statement is executed.

D.6.2.1. Type 2 – ACCESS IS SEQUENTIAL

- Output file

  An OPEN for OUTPUT statement prepares the file for output operation. The file is formatted if the APPLY FILE-PREPARATION clause is specified. Standard labels are written. The USE for BEGINNING LABEL procedure is executed if user labels are specified. The content of ACTUAL KEY is effectively initialized to 1, indicating the first record of the file.

  The WRITE statement records the logical record sequentially on the file, implicitly associating a record number with each of the logical records written. The record number is reported in the ACTUAL KEY item after the record is written. Record numbers start at 1 or, when a SEEK statement follows OPEN, with the programmer-supplied value in ACTUAL KEY, and are incremented sequentially with each WRITE statement.

The SEEK statement positions the direct access device to the track containing the relative record number indicated by the programmer-supplied ACTUAL KEY value. The key value will be used as the relative record number for the next WRITE. Following each WRITE statement, the record number is incremented sequentially until either another SEEK statement is executed or processing is terminated.

The CLOSE statement causes orderly termination of file processing.

- Input file

An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked and user labels, if specified, are made available to the USE for BEGINNING LABEL procedure. The contents of ACTUAL KEY is effectively initialized to 1, indicating the first record of the file.

The READ statement retrieves the logical records sequentially from the file, reporting the relative record number associated with the record in the ACTUAL KEY item. Record numbers start with 1 or, when a SEEK statement follows OPEN, with the value in the ACTUAL KEY item, and are incremented sequentially with each READ statement.

The SEEK statement's operation is the same as when the file is OPENed for OUTPUT except that READ replaces WRITE.

The CLOSE statement causes orderly termination of file processing.

- I-O file

An OPEN for I-O statement prepares the file for input and output operation. Label processing and key initialization is the same as when the file is OPENed for INPUT.

The READ statement's operation is the same as when the file is OPENed for INPUT.

The WRITE statement's operation is the same as when the file is OPENed for OUTPUT except that the ACTUAL KEY item is not incremented.

The operation of the SEEK statement is the same when the file is OPENed for INPUT.

The CLOSE statement causes orderly termination of file processing.

D.6.2.2. Type 4 – ACCESS IS RANDOM

- Output file

An OPEN for OUTPUT statement prepares the file for output operation. The file is formatted if the APPLY FILE-PREPARATION clause is specified. Standard labels are written. The USE for BEGINNING LABEL procedure is executed if user labels are specified.

The WRITE statement causes the logical record to be written onto the file based on the user-supplied relative record number in the ACTUAL KEY item.

The SEEK statement positions the direct access device to the track containing the relative record number indicated by the programmer-supplied ACTUAL KEY value.

The CLOSE statement causes orderly termination of file processing.

- Input file

  An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked and user labels, if specified, are made available to the USE for BEGINNING LABEL procedure.

  The READ statement retrieves the logical record from the file based on the user-supplied relative record number in the ACTUAL KEY item.

  The SEEK statement's operation is the same as when the file is OPENED for OUTPUT.

  The CLOSE statement causes orderly termination of file processing.

- I-O file

  An OPEN for I-O statement prepares the file for input and output operation. Label processing is the same as when the file is OPENed for INPUT.

  The READ statement's operation is the same as when the file is OPENed for INPUT.

  The WRITE and SEEK statement's operation is the same as when the file is OPENed for OUTPUT.

  The CLOSE statement causes orderly termination of file processing.

D.6.3. ORGANIZATION IS DIRECT

Two file processing types are available with direct organization: type 3 when ACCESS IS SEQUENTIAL, and type 5 when ACCESS IS RANDOM.

The logical records of a file with direct organization are identified by a relative track number and by symbolic record identifying information. The contents of the item specified in the ACTUAL KEY clause indicates the position of the logical track relative to the beginning of the file, starting with the value of 1 for the first track. In type 5, the APPLY RESTRICTED SEARCH clause defines the number of tracks to be searched in order to READ or WRITE a requested record. A search of the complete file is performed when the RESTRICTED SEARCH clause is omitted for a type 5 file. The contents of the item specified in the SYMBOLIC KEY clause defines the record identification value of the requested record.

A file that is created with direct organization must be referred to only with direct organization.

Direct organization allows RECORDING MODE IS F or U. Records cannot be blocked nor can an ALTERNATE AREA be assigned to the file.

Before processing a newly allocated (or newly extended) direct file, a new capacity record must be written on each uninitialized track of the file and the remainder of the track must be erased. The FILE-PREPARATION clause may be used to perform the above function. File preparation takes place during OPEN processing when the COBOL programmer:

(1) Writes an APPLY FILE-PREPARATION clause for the file in the I-O-CONTROL paragraph of the Environment Division.

(2) Sets the value of the ACTUAL KEY data item to 1, if the file is new, or to the number of the first track in the new extent(s) when the file is being extended.

(3) Executes an OPEN OUTPUT statement after the ACTUAL KEY value has been set. File preparation occurs each time an OPEN OUTPUT statement is executed.

D.6.3.1. Type 3 – ACCESS IS SEQUENTIAL

- Output File

  An OPEN for OUTPUT statement prepares the file for output operation. The file is formatted if the APPLY FILE-PREPARATION clause is specified. Standard labels are written. The USE for BEGINNING LABEL procedure is executed if user labels are specified. The contents of ACTUAL KEY is effectively initialized to 1, indicating the first track of the file.

  The WRITE statement records the logical record in the next available sequential position in the file. The contents of the SYMBOLIC KEY item is recorded with the data for the purpose of specific record identification during subsequent retrieval or update functions. The relative track number of the record is reported in ACTUAL KEY after the record is written. Track numbers start at 1 or, when a SEEK statement follows OPEN, with the programmer-supplied value in ACTUAL KEY, and are incremented sequentially as each track is filled.

  The SEEK statement positions the direct access device to the track specified by the programmer-supplied ACTUAL KEY value. This track number is used by each WRITE statement until the track is filled, at which time the track number is incremented sequentially. The track number is advanced in this manner until either another SEEK statement is executed or file processing is terminated.

  The CLOSE statement causes orderly termination of file processing.

- Input File

  An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked, and user labels, if specified, are made available to the USE for BEGINNING LABEL procedure. The contents of ACTUAL KEY is effectively initialized to 1, indicating the first track of the file.

  The READ statement retrieves the next available sequential record from the file. After the READ, the relative track number of the retrieved record is reported in the ACTUAL KEY item and the record identification information is reported in the SYMBOLIC KEY item. Track numbers start with 1 or, when a SEEK statement follows OPEN, with the value in the ACTUAL KEY item, and are incremented sequentially as each track is emptied.

The SEEK statement's operation is the same as when the file is OPENed for OUTPUT except that READ replaces WRITE.

The CLOSE statement causes orderly termination of file processing.

- I-O file

An OPEN for I-O prepares the file for input and output operation. Label processing and key initialization is the same as when the file is OPENed for INPUT.

The READ statement's operation is the same as when the file is OPENed for INPUT.

Each WRITE statement must be preceded by a READ statement. The WRITE statement causes the updated logical record to be rewritten onto the same physical area that the record was read from. The SYMBOLIC KEY value is not rewritten. The ACTUAL KEY value for the WRITE is defined by the previous READ. Consecutive WRITE's to the file alter the same physical record position.

The operation of the SEEK statement is the same as when the file is OPENed for INPUT.

The CLOSE statement causes orderly termination of file processing.

D.6.3.2. Type 5 – ACCESS IS RANDOM

- Output file

An OPEN for OUTPUT statement prepares the file for output operation. The file is formatted if the APPLY FILE-PREPARATION clause is specified. Standard labels are written. The USE for BEGINNING LABEL procedure is executed if user labels are specified.

The WRITE statement records the logical record in the next available position of the restricted track extent specified by the programmer-supplied ACTUAL KEY value (relative track number) and the APPLY RESTRICTED SEARCH clause (number of tracks in extent.) The contents of the SYMBOLIC KEY item is recorded with the record for the purpose or specific record identification during subsequent retrieval or update functions.

The SEEK statement positions the direct access device to the track specified by the programmer-supplied ACTUAL KEY value.

The CLOSE statement causes orderly termination of file processing.

- Input file

An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked, and user labels, if specified, are made available to the USE for beginning LABEL procedure.

The READ statement retrieves from the restricted track extent the logical record whose recorded record identification information is equal to the programmer-supplied value in the SYMBOLIC KEY item. The restricted track extent to be searched is specified by the ACTUAL KEY value (relative track number) and the APPLY RESTRICTED SEARCH clause (number of tracks in extent).

The SEEK statement's operation is the same as when the file is OPENed for OUTPUT.

The CLOSE statement causes orderly termination of file processing.

- I-O file

  An OPEN for I-O statement prepares the file for input and output operation. Label processing is the same as when the file is OPENed for INPUT.

  The READ statement's operation is the same as when the file is OPENed for INPUT.

  Each WRITE statement must be preceded by a READ statement. The WRITE statement causes the updated record to be rewritten onto the same physical area that the record was read from. The SYMBOLIC KEY value is not rewritten. The ACTUAL KEY value for the WRITE is defined by the previous READ. Consecutive WRITE's to the file alter the same physical record position.

  The operation of the SEEK statement is the same as when the file is OPENed for OUTPUT.

  The INSERT statement allows new records to be added to the file. The INSERT statement's operation is the same as a WRITE when the file is OPENed for OUTPUT.

  The CLOSE statement causes orderly termination of file processing.

D.6.4. ORGANIZATION IS INDEXED

Two file processing techniques are available with indexed organization: type 6 when ACCESS IS SEQUENTIAL, and type 7 when ACCESS IS RANDOM. Type 6 must be used to initially create a file or extend an existing file. Type 7 must be used to add records into (INSERT) an existing file.

The logical records of a file with indexed organization are identified by the value contained in an embedded identification field specified by the RECORD KEY clause.

A file that is created with indexed organization must be referred to only with indexed organization.

Indexed organization allows only RECORDING MODE IS F. An ALTERNATE AREA cannot be assigned to an indexed file. Records may be blocked or unblocked. The recorded format of unblocked records varies depending on the position of the RECORD KEY within the record. If the RECORD KEY is the first item described within the record, the physical record format is:

If the RECORD KEY is not the first item described within the record, the physical record format is:

In either case, the RECORD KEY value is delivered on a READ statement as part of the record.

Filepreparation is not necessary with indexed organization.

D.6.4.1. Type 6 – ACCESS IS SEQUENTIAL

- Output file

  An OPEN for OUTPUT statement prepares the file for output operation. File creation (load) is assumed unless the file already exists, in which case file extension is implied. Standard labels are written. User labels are not permitted.

  The WRITE statement records the logical record sequentially in the prime data area of the file after blocking, as required, is effected. The RECORD KEY value of the previous logical record. If the file is being extended, the RECORD KEY value of the first logical record written must be higher than the RECORD KEY value of the last logical record written during the previous file load or extension process.

  The CLOSE statement causes orderly termination of file processing. Standard labels are written.

- Input file

  An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked.

  A READ statement causes the next sequential record in the file to be made available after deblocking, as required, is effected. The RECORD KEY value is delivered as part of the logical record.

  The SEEK statement causes the programmer-supplied value in the SYMBOLIC KEY item to specify the RECORD KEY value of the next record to be read. If no record is found with that KEY, positioning is made to the record with the next higher KEY. Following the READ, records are again retrieved sequentially until another SEEK statement is executed or file processing is terminated.

  The CLOSE statement causes orderly termination of file processing. Standard labels are written.

- I-O file

  An OPEN for I-O statement prepares the file for input and output operation. Label processing is the same as when the file is OPENed for INPUT.

  The READ statement's operation is the same as when the file is OPENed for INPUT.

  Each WRITE statement must be preceded by a READ statement. The WRITE statement causes the updated record to be rewritten onto the same physical area that the record was read from. The RECORD KEY value must not be changed.

  The operation of the SEEK and CLOSE statements is the same as when the file is OPENed for INPUT.

D.6.4.2. Type 7 – ACCESS IS RANDOM

- Input file

  An OPEN for INPUT statement prepares the file for input operation. Standard labels are checked.

  The READ statement retrieves the logical record from the file whose RECORD KEY value is the same as the programmer-supplied value in the SYMBOLIC KEY item after deblocking, as required, if effected.

  The CLOSE statement causes orderly termination of file processing. Standard labels are written.

- I-O file

  An OPEN for I-O statement prepares the file for input and output operation. Label processing is the same as when the file is OPENed for INPUT.

  The READ statement's operation is the same as when the file is OPENed for INPUT.

  Each WRITE statement must be preceded by a READ statement. The WRITE statement causes the updated record to be rewritten onto the same physical area from which record was read. The RECORD KEY must not be changed.

  An INSERT statement causes the new logical record to be added into the file. The position of the inserted record is determined by its RECORD KEY value. Since all prime data area tracks are fully loaded (dense) during file load or file extension, an overflow area must be provided for INSERT's. The overflow area may be a cylinder overflow area (APPLY CYLINDER-OVERFLOW clause) and/or an independent overflow area (specified via the job control stream when the file is allocated). If both overflow areas exist, the cylinder overflow area is used first.

  The CLOSE statement's operation is the same as when the file is OPEN for INPUT.

D.7. SUMMARY OF AT END/INVALID KEY/ERROR CONDITIONS

(1) ORGANIZATION IS SEQUENTIAL

The AT END imperative statement is executed when an end-of-file record is detected.

The INVALID KEY imperative statement is executed when there is no space left on the file for the record to be written.

(2) ORGANIZATION IS RELATIVE or DIRECT

For type 2 and type 3 files, the AT END imperative statement is executed when an access to a record which is beyond the file is attempted.

For type 2 through type 5 files, the INVALID KEY imperative statement is executed when the relative record number or relative track number is beyond the file extents. The INVALID KEY imperative statement is also executed for a type 5 file if the record being processed is not located, or cannot be placed into, the current restricted track extent.

The following error conditions are reported in SYSERR for type 2 through type 5 files:

SYSERR-1	Wrong Length Record
SYSERR-4	No Room Found
SYSERR-8	Data Check Count Area
SYSERR-9	Track Overrun
SYSERR-10	End of Cylinder
SYSERR-11	Data Check in Key or Data
SYSERR-12	Record Not Found
SYSERR-13	End of File Record detected
SYSERR-14	End of Volume
SYSERR-15	Record not in specified File Extents

Additional information can be found in Table 4–1 of *UNIVAC 9400 Data Management System Programmers Reference, UP-7629* (current version).

The USE FOR ERROR procedure, if specified, is executed if any file error exists that did not cause an AT END or INVALID KEY condition.

When executing a USE FOR ERROR procedure, if SYSERR is OFF, a major hardware failure or major logic error is indicated.

(3) ORGANIZATION IS INDEXED

For type 6 files, the AT END imperative statement is executed when the logical end of file is reached.

For type 6 or type 7 files, the INVALID KEY imperative statement is executed if:

(a) during file creation or extension, a RECORD KEY value is found to be out of sequence;
(b) a duplicate RECORD KEY value is detected; or
(c) the RECORD KEY value cannot be found.

The following error conditions are reported in SYSERR for indexed files:

SYSERR-0 Unrecoverable device error
SYSERR-2 Prime data area full
SYSERR-3 Index area too small/record not found
SYSERR-5 Duplicate KEY
SYSERR-6 Records out of sequence/overflow area full
SYSERR-7 Record retrieved from overflow area

Additional information can be found in Table F–1 of *UNIVAC 9400 Data Management System Programmers Reference, UP-7629* (current version).

The USE for ERROR procedure, if specified, is executed if any file error exists that did not cause an AT END or INVALID condition.

When executing a USE for ERROR procedure, if SYSERR is OFF, a major hardware failure or major logic error is indicated.

Figure D–1 summarizes available SYSERR-n settings for ORGANIZATION IS INDEXED. The SYSERR-n column states the condition causing the setting of the corresponding SYSERR-n. The order column headings are self explanatory. The following is an example of the use of the figure.

ACCESS IS RANDOM, ORGANIZATION IS INDEXED, and a file that has been opened for INPUT is being READ. The programmer wishes to know which SYSERR number(s) is meaningful during INVALID KEY processing. He must find in the figure the paths which satisfy all conditions of the problem to obtain the applicable SYSERR number(s). In this example, SYSERR-3 is the only path that contains INVALID KEY, RANDOM, INPUT, and READ. SYSERR-3 will be set if no record is found when using SYMBOLIC KEY.

Table D–1 summarizes the COBOL disc processing techniques.

SYSERR-n	MEANINGFUL DURING	ACCESS TYPE	OPEN FOR	VERBS	SYSERR DEFINITION
0	NORMAL	SEQUENTIAL	INPUT, OUTPUT, I-O	SEEK	UNRECOVERABLE DEVICE ERROR
	USE FOR ERROR	SEQUENTIAL, RANDOM	INPUT, OUTPUT, I-O	READ, WRITE, INSERT	
2	NORMAL, USE FOR ERROR	SEQUENTIAL	OUTPUT	WRITE	PRIME DATA AREA FULL*
3	NORMAL	SEQUENTIAL	OUTPUT	OPEN	INDEX AREA TOO SMALL
	USE FOR ERROR	SEQUENTIAL	OUTPUT	WRITE	
	INVALID KEY	RANDOM	INPUT, I-O	READ, WRITE	
	NORMAL	SEQUENTIAL	INPUT, I-O	SEEK	NO RECORD FOUND USING SYMBOLIC KEY
	USE FOR ERROR	SEQUENTIAL	INPUT, I-O	READ, WRITE	
5	INVALID KEY	SEQUENTIAL, RANDOM	OUTPUT, I-O	WRITE, INSERT	DUPLICATE KEYED RECORD
6	INVALID KEY	SEQUENTIAL	OUTPUT	WRITE	OUT OF SEQUENCE
	USE FOR ERROR	RANDOM	I-O	INSERT	OVERFLOW AREA FULL
7	NORMAL	SEQUENTIAL, RANDOM	INPUT, I-O	READ	RECORD RETRIEVED FROM OVERFLOW AREA

*If prime data area is full on first WRITE, SYSERR-2 should be tested during NORMAL processing. If prime data area is full for any subsequent WRITE, SYSERR-2 should be tested in USE ERROR PROCEDURE processing.

Figure D-1. SYSERR-n Settings for ORGANIZATION IS INDEXED

PROCESSING TECHNIQUE		TYPE	ADDRESSING TECHNIQUE	REQUIRED KEY CLAUSES	RECORD FORMAT (4)	OPEN VERB	ALLOWABLE I-O STATEMENTS	REQUIRED CLAUSES	OPTIONAL CLAUSES	RESTRICTED CLAUSES
ORGANIZATION	ACCESS									
SEQUENTIAL OR OMITTED	SEQUENTIAL OR OMITTED	1		NONE ALLOWED	F U V	INPUT	READ AT END	SELECT ASSIGN LABEL RECORDS ARE {STANDARD / DATA-NAME} CLOSE	SELECT OPTIONAL, MULTIPLE UNIT, RESERVE, SAME (RECORD) AREA, BLOCK CONTAINS, RECORD CONTAINS, DATA RECORDS, APPLY VERIFY, USE LABEL, USE ERROR, CLOSE UNIT, READ INTO, WRITE FROM	APPLY RESTRICTED SEARCH, APPLY FILE-PREPARATION, APPLY MASTER-INDEX, APPLY CYLINDER-OVERFLOW, APPLY EXTENDED-INSERTION, APPLY CYLINDER-INDEX AREA
						OUTPUT	WRITE INVALID KEY			
						I-O	READ AT END WRITE INVALID KEY			
RELATIVE	SEQUENTIAL OR OMITTED	2 (1) (2)	RELATIVE RECORD (3)	ACTUAL OR RELATIVE	F	INPUT	READ AT END, SEEK		SAME (RECORD) AREA, RECORD CONTAINS, BLOCK CONTAINS 1 RECORD, DATA RECORD, APPLY VERIFY, APPLY FILE-PREPARATION USE LABEL, USE ERROR, RESERVE NO ALTERNATE AREA, READ INTO, WRITE FROM, INSERT FROM	APPLY RESTRICTED SEARCH, RESERVE INTEGER, OPTIONAL BLOCK CONTAINS 1 RECORD, USE ENDING LABEL, APPLY MASTER-INDEX, APPLY CYLINDER-OVERFLOW, APPLY EXTENDED-INSERTION, APPLY CYLINDER-INDEX AREA
						OUTPUT	WRITE INVALID KEY, SEEK			
						I-O	READ AT END, WRITE (5) INVALID KEY, SEEK (6)			
DIRECT	SEQUENTIAL OR OMITTED	3 (1) (2)	RELATIVE TRACK (3)	ACTUAL AND SYMBOLIC	F U	INPUT	READ AT END, SEEK			
						OUTPUT	WRITE INVALID KEY, SEEK			
						I-O	READ AT END, WRITE (5) INVALID KEY, SEEK (6)			
RELATIVE OR OMITTED	RANDOM	4 (2)	RELATIVE RECORD (3)	ACTUAL OR RELATIVE	F	INPUT	READ INVALID KEY, SEEK			
						OUTPUT	WRITE INVALID KEY, SEEK			
						I-O	READ INVALID KEY, WRITE (5) INVALID KEY, SEEK (6)			
DIRECT	RANDOM	5 (1) (2)	RELATIVE TRACK (3)	ACTUAL AND SYMBOLIC	F U	INPUT	READ INVALID KEY, SEEK			
						OUTPUT	WRITE INVALID KEY, SEEK		(SAME AS ABOVE) AND APPLY RESTRICTED SEARCH	(SAME AS ABOVE) EXCEPT APPLY RESTRICTED SEARCH IS ALLOWED
						I-O	READ INVALID KEY, WRITE (5) INVALID KEY, INSERT INVALID KEY, SEEK (6)			
INDEXED	SEQUENTIAL	6 (1) (2)		RECORD (7) {SYMBOLIC}	F	INPUT	READ AT END, SEEK	SELECT/ASSIGN LABEL RECORDS ARE STANDARD CLOSE	SAME (RECORD) AREA, RECORD CONTAINS, BLOCK CONTAINS, DATA RECORDS, RESERVE NO, APPLY VERIFY, APPLY MASTER-INDEX, APPLY CYLINDER-OVERFLOW, USE ERROR, INTO, FROM	APPLY RESTRICTED SEARCH, APPLY FILE-PREPARATION, RESERVE INTEGER, OPTIONAL, USE LABELS, LABEL RECORDS OMITTED OR DATA-NAME USE ENDING LABEL
						OUTPUT	WRITE INVALID KEY			
						I-O	READ AT END, WRITE (5) INVALID KEY, SEEK			
INDEXED	RANDOM	7 (1) (2)		RECORD AND SYMBOLIC	F	INPUT	READ INVALID KEY		(SAME AS ABOVE) AND APPLY EXTENDED-INSERTION, APPLY CYLINDER-INDEX AREA	
						I-O	READ INVALID KEY, WRITE (5) INVALID KEY, INSERT INVALID KEY			

(1) ANSI language element extension

(2) Extended compiler only

(3) Requires preformatting of entire file prior to writing

(4) Default RECORD FORMAT is underlined

(5) REWRITE accepted as synonym for WRITE

(6) SEEK not permitted between READ and WRITE

(7) ACTUAL key may be used in place of SYMBOLIC key for UNIVAC 9300 System compatibility.

Table D–1. Summary of COBOL Disc Processing Techniques

APPENDIX E. EXTENDED COMPILER FEATURES

E.1. GENERAL

As previously stated in 1.3 of this manual, the basic compiler is a true subset of the extended compiler. The features that are contained only in the extended compiler are summarized in Table E–1, with appropriate references to the text.

EXTENDED COMPILER FEATURE	REFERENCE
SD	5.2.2
compound conditions	6.5.2
COMPUTE verb	6.6, 6.6.1, 6.7.6
condition-name, with multiple values or THRU option only	5.3.12
COPY REPLACING	6.7.7
IF statement condition associated with range of value	5.3.12
INSERT	6.7.16
OCCURS DEPENDING	5.3.3
PERFORM statement, formats 3 and 4	6.7.21
RELEASE verb	6.7.23
RETURN verb	6.7.24
REWRITE	6.7.25
SEARCH	6.7.26
SEEK verb	6.7.27
SEGMENT-LIMIT	4.2.2
SORT verb	6.6, 6.7.29
VALUE IS statement, format 2	5.3.8
CORRESPONDING option for ADD, SUBTRACT, and MOVE verbs	6.7.2, 6.7.31, 6.7.17
Random Access Features	
ACCESS IS RANDOM	4.3.1
ORGANIZATION	4.3.1
KEYS	4.3.1
APPLY VERIFY	4.3.2
APPLY MASTER-INDEX	4.3.2
APPLY CYLINDER-INDEX AREA	4.3.2
APPLY CYLINDER-OVERFLOW AREA	4.3.2
APPLY EXTENDED-INSERTION	4.3.2
APPLY FILE-PREPARATION	4.3.2
SYSERR, SYSERR[-n]	4.2.3

Table E–1. Extended Compiler Features

APPENDIX F. CALLING AND CALLED PROGRAMS

F.1. GENERAL

Run-time communication between a main COBOL program and any other separately compiled or assembled subprogram is accomplished by use of the ENTER statement together with its associated statements:

- CALL
- ENTRY
- EXIT PROGRAM or RETURN
- USING clause with PROCEDURE DIVISION header

Actual transfer of control from a CALLing program to a CALLed program is via a CALL statement whose entry-name is identical to the entry-name in the ENTRY statement of the CALLed program. Return of control to the CALLing program is effected by the execution of an EXIT PROGRAM statement in the CALLed program. Control is returned to the statement following the CALL statement in the CALLing program.

Note that a CALLed program need not be a COBOL program. In such cases, the COBOL CALLing program may include procedure-names in its USING argument list. All lines in the examples containing an asterisk (*) in the continuation area (column 7) are commentary lines.

F.2. TREATMENT OF DATA ITEMS

Data items which are declared in the CALLing program and are referenced in the CALLed program are described in the File or Working Storage Section in the Data Division of the CALLing program. In the CALLed program, these data items are described, once again, but in the LINKAGE SECTION. Items described in the LINKAGE SECTION are not allocated computer storage by the compiler since these items already occupy storage in the CALLing program which furnishes their addresses to the CALLed program at object time.

Data items common to both programs are shared by means of corresponding USING clauses in each program. The operands in the USING clause of the CALLing program name the data items contained in the Data Division which are to be shared with the CALLed program. The USING clause in the CALLed program can either follow the PROCEDURE DIVISION header or be contained in an ENTRY statement. The operands must name data items described by 01 or 77 level entries in the LINKAGE SECTION.

The sequence of appearance of the operands in the two USING clauses is extremely significant since corresponding operands refer to a single common data item, i.e., correspondence is by position and not by name. Each reference to an operand in the CALLed program's USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the CALLing program. The CALLing program is responsible for ensuring physical data alignment if the description of a LINKAGE SECTION data item implies a hardware alignment requirement.

It should be noted that a CALLed program may also be a CALLing program sharing common data items in its Data Division (including LINKAGE SECTION items) with still another CALLed program.

F.3. LINKING

A sample linker job stream for CALLing and CALLed programs follows:

```
/$

        LOADM       CALLXX
        INCLUDE     CALLER,*
        INCLUDE     CALLED,*
        INCLUDE     ADDROUT,*

/*
```

F.4. UNIVAC 9400 COBOL CALL/ENTRY INTERFACE

The following example is provided to illustrate the use of CALL and ENTRY statements. The example consists of a COBOL program (CALLER, see Figure F–1) which shares data-items and calls upon a CALL subprogram (CALLED, see Figure F–2) and an assembly language subprogram (ADDROUT, see Figure F–3), for operations upon the shared data-items. Table F–1 shows the relationship between these programs.

ROUTINE	TYPE	LANGUAGE	INTERFACE	FUNCTION	COMMENT
CALLER	Program	COBOL	CALLs COBOLADD in CALLED. CALLs ASMBLRAD in ADDROUT.	Sets values in data-items and CALLs on subprograms to add values and provide result. Results are displayed on console.	Note that any 01 or 77 level data-item can be used as operand in CALL statement (shared with subprogram).
CALLED	Subprogram	COBOL	ENTRY point is COBOLADD. EXIT accomplished via exit program.	Adds values in several shared data-items and leaves result in a shared data-item.	Items to be shared with a CALLing program are described as 01 or 77 level data-items in LINKAGE SECTION.
ADDROUT	Subprogram	ASM	ENTRY point is ASMBLRAD. Exit accomplished via BR RE$.	Same as CALLED above.	Items to be shared with a CALLing program may be described within a DSECT. The arguments passed represent the address of each item in the CALLing program storage.

Table F–1. Program/Subprogram Relationships

```
LINE NO.    SOURCE STATEMENT
00001       IDENTIFICATION DIVISION.
00002       PROGRAM-ID.  CALLER.
00003       ENVIRONMENT DIVISION.
00004       CONFIGURATION SECTION.
00005       SOURCE-COMPUTER.  UNIVAC-9400.
00006       OBJECT-COMPUTER.  UNIVAC-9400.
00007       DATA DIVISION.
00008       WORKING-STORAGE SECTION.
00009       77  DATA1      PIC 9999.
00010       77  DATA2      PIC 99.
00011       77  CTR        PIC 99    VALUE 01.
00012       01  DATAX.
00013           02  DATA3  PIC 99.
00014           02  DATA4  PIC 99.
00015       PROCEDURE DIVISION.
00016       P0.
00017           MOVE CTR TO DATA2, DATA3, DATA4.
00018       P0D.
00019          ENTER LINKAGE.
00020          CALL ASMBLRAD   USING DATA2, DATAX, DATA1.
00021          ENTER COBOL.
00022          DISPLAY ' CALLER RECVD ' DATA2 ' + ' DATA3 ' + ' DATA4 ' = '
00023          DATA1 ' FROM ASMBLRAD '.
00024          ADD 1 TO DATA4.
00025       P1.
00026          ENTER LINKAGE.
00027          CALL COBOLADD USING DATA2, DATAX, DATA1.
00028          ENTER COBOL.
00029       P3.
00030          DISPLAY ' CALLER RCVD ' DATA2 ' + ' DATA3 ' + ' DATA4 ' = '
00031          DATA1 ' FROM COBOLADD'.
00032       P4.    IF CTR LESS THAN 12 ADD 1 TO CTR GO TO P0 ELSE
00033              DISPLAY 'END OF RUN' STOP RUN.
```

Figure F-1. Example of CALLing Program

```
LINE NO.                    SOURCE STATEMENT
00001                       IDENTIFICATION DIVISION.
00002                       PROGRAM-ID.  CALLED.
00003                       ENVIRONMENT DIVISION.
00004                       CONFIGURATION SECTION.
00005                       SOURCE-COMPUTER.  UNIVAC-9400.
00006                       OBJECT-COMPUTER.  UNIVAC-9400.
00007                       DATA DIVISION.
00008                       LINKAGE SECTION.
00009                       01  DATAX.
00010                           02  DATA3  PIC 99.
00011                           02  DATA4  PIC 99.
00012                       77 DATA1      PIC 9999.
00013                       77 DATA2      PIC 99.
00014                       PROCEDURE DIVISION.
00015                       PO.  ENTER LINKAGE.  ENTRY COBOLADD USING DATA2 DATAX DATA1.
00016                            ENTER COBOL.
00017                       P1.  ADD DATA2 DATA3 DATA4 GIVING DATA1.
00018                       P9.  ENTER LINKAGE.  EXIT PROGRAM.  ENTER COBOL.
```

Figure F-2. Example of CALLed Program

```
ADDROUT  START 0
         PRINT NOGEN
         STDEQU
         PRINT  GEN
DUMMY    DSECT
DATA2ASM DS    CL2              A DSECT IS A DESCRIPTION NOT TO
DATAXASM DS    OCL4               BE MAPPED SINCE IT WILL RESIDE
DATA3ASM DS    CL2                ELSEWHERE AT OBJECT TIME
DATA4ASM DS    CL2
DATA1ASM DS    CL4
ADDROUT  CSECT
         USING DATA2ASM,R2$       R2 WILL BE USED TO COVER DATA2
         USING DATAXASM,R3$       R3 WILL BE USED TO COVER DATA3/4
         USING DATA1ASM,R4$       R4 WILL BE USED TO COVER DATA1
         USING *,RF$              COVER FOR THIS ROUTINE
ASMBLRAD STM   RE$,RC$,12(RD$)    SAVE CALLERS REGS IN HIS SAVEAREA
         ENTRY ASMBLRAD           DECLARES ENTRY POINT LABEL
         LR    R2$,RD$            SAVE ADR OF CALLERS SAVEAREA
         LA    RD$,SAVEAREA       LOAD RD$ WITH ADDR OF THIS ROUT S-A
         STM   R2$,R2$,4(RD$)     SAVE CALLER S-A ADR IN THIS ROUT SA
         STM   RD$,RD$,8(R2$)     SAVE THIS ROUT SA ADR IN CALLER SA
         LM    R2$,R4$,0(R1$)     LOAD COVER REGS WITH ARG'S
         PACK  HOLD2(2),DATA2ASM(2)
         ZAP   ACCUM(3),HOLD2(2)
         PACK  HOLD2(2),DATA3ASM(2)
         AP    ACCUM(3),HOLD2(2)
         PACK  HOLD2(2),DATA4ASM(2)
         AP    ACCUM(3),HOLD2(2)
         UNPK  DATA1ASM(4),ACCUM(3)
         L     RD$,4(,RD$)        ADDR OF CALLERS SA
         LM    RE$,RC$,12(RD$)    RESTORES CALLERS REGS
         MVI   12(RD$),X'FF'      SET CALLED TO RETURNED STATUS
         BR    RE$
SAVEAREA DS    18F
ACCUM    DS    CL3
HOLD2    DS    CL2
         END
```

Figure F–3. Example of CALLed Assembly Subprogram

Example console typeout at execution time of the three programs.

```
 13:42 RE 15
I13:42 Ø1 ST22 AC2 RL        AP
I13:42 Ø1 JCØ6 15 COBOL      CALLXX          :    :    :Ø94
I13:42 15 ST18 // ALTER ,.L
I13:42 15 CD1Ø  CALLER RCVD Ø1 + Ø1 + Ø1 = ØØØC FROM ASMBLRAD
I13:42 15 CD1Ø  CALLER RCVD Ø1 + Ø1 + Ø2 = ØØØ4 FROM COBOLADD
I13:43 15 CD1Ø  CALLER RCVD Ø2 + Ø2 + Ø2 = ØØØF FROM ASMBLRAD
I13:43 15 CD1Ø  CALLER RCVD Ø2 + Ø2 + Ø3 = ØØØ7 FROM COBOLADD
I13:43 15 CD1Ø  CALLER RCVD Ø3 + Ø3 + Ø3 = ØØØI FROM ASMBLRAD
I13:43 15 CD1Ø  CALLER RCVD Ø3 + Ø3 + Ø4 = ØØ1Ø FROM COBOLADD
I13:43 15 CD1Ø  CALLER RCVD Ø4 + Ø4 + Ø4 = ØØ1B FROM ASMBLRAD
I13:43 15 CD1Ø  CALLER RCVD Ø4 + Ø4 + Ø5 = ØØ13 FROM COBOLADD
I13:43 15 CD1Ø  CALLER RCVD Ø5 + Ø5 + Ø5 = ØØ1E FROM ASMBLRAD
I13:43 15 CD1Ø  CALLER RCVD Ø5 + Ø5 + Ø6 = ØØ16 FROM COBOLADD
I13:43 15 CD1Ø  CALLER RCVD 06 + 06 + 06 = 001H FROM ASMBLRAD
I13:44 15 CD1Ø  CALLER RCVD 06 + 06 + 07 = 0019 FROM COBOLADD
I13:44 15 CD1Ø  CALLER RCVD Ø7 + Ø7 + Ø7 = ØØ2A FROM ASMBLRAD
I13:44 15 CD1Ø  CALLER RCVD Ø7 + Ø7 + Ø8 = ØØ22 FROM COBOLADD
I13:44 15 CD1Ø  CALLER RCVD Ø8 + Ø8 + Ø8 = ØØ2D FROM ASMBLRAD
I13:44 15 CD1Ø  CALLER RCVD Ø8 + Ø8 + Ø9 = ØØ25 FROM COBOLADD
I13:44 15 CD1Ø  CALLER RCVD Ø9 + Ø9 + Ø9 = ØØ2G FROM ASMBLRAD
I13:44 15 CD1Ø  CALLER RCVD Ø9 + Ø9 + 1Ø = ØØ28 FROM COBOLADD
I13:44 15 CD1Ø  CALLER RCVD 1Ø + 1Ø + 1Ø = ØØ3  FROM ASMBLRAD
I13:44 15 CD1Ø  CALLER RCVD 1Ø + 1Ø + 11 = ØØ31 FROM COBOLADD
I13:45 15 CD1Ø  CALLER RCVD 11 + 11 + 11 = ØØ3C FROM ASMBLRAD
I13:45 15 CD1Ø  CALLER RCVD 11 + 11 + 12 = ØØ34 FROM COBOLADD
I13:45 15 CD1Ø  CALLER RCVD 12 + 12 + 12 = ØØ3F FROM ASMBLRAD
I13:45 15 CD1Ø  CALLER RCVD 12 + 12 + 13 = ØØ37 FROM COBOLADD
I13:45 15 CD1Ø END OF RUN
I13:45 Ø1 JTØ1 15 COBOL    RUN TIME     : 1:14:476
```

APPENDIX G. COMPILER OPTIONS

G.1. GENERAL

The optional // PARAM card provides a method of presenting parameters to the compiler to exercise specific COBOL options.

NOTE: Only one blank may precede the P of the word PARAM.

When PARAM cards are used, they must be positioned immediately following the // EXEC EXEC card in the compilation job stream. The // PARAM cards will be printed on the first page of the compiler output listing.

If a PARAM card format error or an illegal parameter is encountered, a console message is produced and the compilation is terminated.

If there are no PARAM cards supplied, the compiler will produce a source program listing and a source program diagnostic report, and generate an object module.

Absence of PARAM cards implies:

```
// PARAM LST=(S)
```

G.2. LIST OPTIONS

Format:

```
// PARAM LST=(spec 1,...,spec n)
```

where spec 1,..., spec n is one or more of the following:

A – Activate ambiguity mode of reference resolution. Normally, references are resolved by the first appropriate definition encountered for the referenced name. The definition search process begins with the first entry in the appropriate division and continues through to the last entry in that division.

In the ambiguity mode, the definition search process is not terminated when the reference has been resolved, but continues in an attempt to uncover and report duplicate definitions. When the search of the division that corresponds to the reference type has been completed, the other divisions are also searched to determine if the highest possible qualifier rule has been violated. Diagnostic messages 151 through 154 report the presence of ambiguous references/definitions.

C – Produce storage map and cross reference listing for the Data Division and Procedure Division.

E – Ignore printer mismatch errors during compilation.

K – Inhibit source item sequence number checking (columns 1 through 6 of the source item)

L – Single space source and diagnostic listing.

M – Produce Data Division storage map listing.

N – Inhibit all listable output except PARAM card listing and compilation identification heading.

O – Produce object code listing.

P – Produce Procedure Division storage map listing.

S – Produce source program listing.

W – Inhibit listing of all precationary diagnostics. These errors are identified by a severity code of P.

G.3. OUTPUT OPTIONS

Formats:

```
// PARAM OUT=(spec 1,...,spec n)
```

where spec 1,...,spec n is one or more of the following:

A – Produce ASCII sensitive object program.

L – Inhibit generation of linker control cards in object module.

N – Inhibit generation of object module.

P – Disregard mis-match errors for all object program print files.

S – Disable object program SORT PARAM request console message.

T – Inhibit compiler generation of a transfer address in the object mode. When invoked, the program cannot be executed unless it is CALLed.

G.4. SOURCE LIBRARY INPUT

Format:

```
// PARAM IN=program-name/file-name
```

where:

program-name – 1- to 8-character name of source program to be compiled.

file-name – 1- to 8-character name used to identify the file on which the source program resides. This name must appear on the LFD control card used to define the device to the Job Control program.

If the file-name is omitted, the following name will automatically be supplied:

Name when using basic system – SYSRES
Name when using extended system – SORS$

NOTE: When only four tapes are available for compilation, source input should be mounted on SCR2. The compiler console message:

DISMOUNT SCR2. MOUNT A SCRATCH TAPE ON SCR2.

appears when tape is to be changed.

G.5. COPY LIBRARY INPUT

Format:

```
// PARAM LIN=file-name
```

file-name – 1- to 8-character name used to identify the file on which the COPY library resides. This name must appear on the LFD control card used to define the device to the Job Control program.

If the file-name is omitted, the following name will automatically be supplied:

Name when using basic system – SYSRES
Name when using extended system – COPY$

The COPY element-name is supplied in the source program via the COPY clause.

G.6. OBJECT MODULE VERSION/REVISION NUMBER

Format:

```
// PARAM VER=vv/rr
```

where:

vv – version number

rr – revision number

These numbers are applied to compiler output module.

APPENDIX H. INTERMEDIATE RESULTS IN ARITHMETIC OPERATIONS

H.1. GENERAL

For certain arithmetic statements, the compiler generates code that uses internal work areas for storage of intermediate results. Intermediate results may be required in the following types of statements:

(1) ADD, where more than one operand preceeds TO or GIVING.

(2) SUBTRACT, where more than one operand preceeds FROM or GIVING.

(3) Any statement containing an arithmetic expression which specifies more than one operation.

Arithmetic expressions are simplified by the compiler to become a series of simple arithmetic operations that store partial results in intermediate result areas, which may then be used as operands in succeeding operations.

The compiler provides a description for an intermediate result which is appropriate for its use in the operation or series of operations for which it is required. This description can be expressed as a numeric PICTURE; however, it should be noted that an intermediate result, used in the evaluation of an expression, may contain as many as 30 digits.

H.2. ADD AND SUBTRACT STATEMENTS

The description of the intermediate result area is determined by forming the composite of operands (see 6.6.1) and appending one additional digit in the most significant position to contain overflow when there are 10 or fewer operands immediately following the verb, or two digits when there are more than 10 operands.

H.3. EXPRESSIONS

The following abbreviations are used:

l — length in mappable digits.

pl — point location which is the number of places that the decimal point is displaced from the position it would occupy if the mappable digits were considered to be an integer. For example, for the picture 99V9, pl = 1, because the decimal point has been displaced one position; for the picture PP999, pl = 5. A negative value in pl indicates trailing P's in the associated picture, e.g., for the picture 99PP, pl = −2.

OP1 — first operand

OP2 — second operand

ir — intermediate result

comp — composite of operands

mag — magnitude = l − pl
The maximum value that a variable can assume is $10^{mag} - 10^{-pl}$.

When expressions are evaluated, a composite of all operands except those immediately to the right of the exponentiation operator is formed. The receiving data item, when present, is considered in determining the composite. The following rules apply:

Operator	Description
+,−	$pl_{ir} = \max(pl_{OP1}, pl_{OP2})$ $L_{ir} = \max(mag_{OP1}, mag_{OP2}) + pl_{ir}$
*	$pl_{ir} = pl_{OP1} + pl_{OP2}$ $L_{ir} = mag_{OP1} + mag_{OP2} + pl_{ir}$
/	$pl_{ir} = pl_{comp}$ $L_{ir} = pl_{OP2} - pl_{OP1} + L_{OP1} + pl_{ir}$
**	$pl_{ir} = 12$ $L_{ir} = 30$

NOTE: When an expression appears in a COMPUTE statement and the ROUNDED option is specified, one digit is added in the least significant position of the receiver description before the composite is formed.

When application of the preceding rules produces an intermediate result length that is greater than 30, the description must be re-adjusted. Three cases must be considered:

Case	Condition	pl_{ir}	l_{ir}
1	$pl_{ir} \leq pl_{comp}$	unchanged	30
2	$pl_{ir} > pl_{comp}$ AND $L_{ir} - pl_{comp} > 30$	pl_{comp}	30
3	$pl_{ir} > pl_{comp}$ AND $L_{ir} - pl_{comp} \leq 30$	$30 + pl_{ir} - l_{ir}$	30

APPENDIX I. JOB CONTROL STREAM

I.1. GENERAL

The load modules for the basic compiler are named COBOLB00 through COBOLB24; and for the extended compiler, COBOL000 through COBOL024.

I.2. SAMPLE JOB STREAMS

Listed below are sample job streams used for compilation:

BASIC (T.O.S.)

```
// JOB jobname
// OPTION NOVOL
// DVC  3
// LFD PRNTR
// DVC  4
// LFD SYSRES
// DVC  5
// LFD SCR1
// DVC  6
// LFD OBJFIL
// DVC  7
// LFD SCR 2
// EXEC COBOLB00,SYSRES
// PARAM operand,operand,...
/$
   COBOL Source Program
/*
/&
```

EXTENDED (D.O.S.)

```
// JOB jobname
// OPTION NOVOL
// DVC  3
// LFD PRNTR
// DVC  20
// VOL xxxx
// LFD SYSRES
// DVC  20
// VOL xxxx
// DVC  21
// VOL xxxx
// LFD SYSPOOL

// EXEC COBOL000,LOAD$LIB,,REL
// PARAM operand,operand,...
/$
   COBOL Source Program
/*
/&
```

BASIC (D.O.S.)

```
// JOB jobname
// OPTION NOVOL
// DVC  3
// LFD PRNTR
// DVC  4
// LFD SCR1
// DVC  10
// LFD SCR2
// DVC  11
// LFD OBJFIL
// EXEC COBOLB,LOAD$LIB,,REL
// PARAM operand,operand,...
/$
   COBOL Source Program
/*
/&
```

The following operating instructions pertain to compiling under DOS, with either the basic or extended compiler:

(1) Boot from disc (the system will load the compiler from tape for basic COBOL only).

(2) File job stream on disc via console: FILE (EOM) (All cards in reader will be filed, including the COBOL source deck).

(3) Run job via console:

RUN jobname,,GO,A000 (A000 allocates remainder of memory for compilation when running in 65K environment).

(4) Normal system console messages may appear during compilation.

(5) Compilation is terminated when the following message appears on console:

9400 COBOL COMPILATION TIME FOR program-id START xxxx END xxxx

Operating instructions for running basic COBOL under TOS are the same as above, with the following exceptions:

(1) Boot system from tape instead of disc.

(2) Job stream may not be filed.

(3) Console run command is:

RUN jobname,,GO

If the source program to be compiled exists on a library tape or disc file, follow the procedures described under PARAM=IN, in Appendix G.

The job stream statements used for compile-link-go on the extended disc compiler are as follows:

(1) Compiling a source program from cards

```
COMPILE
        // JOB name
        // DVC  3 // LFD PRNTR
        // DVC  20 // VOL DSP155 // LFD Load
        // DVC  20 // VOL DSP155 // DVC 21 // VOL DSP185
        // LFD SYSPOOL
        // EXEC COBOL,LOAD$LIB,,REL
        // PARAM (your options)
        /$
           Source Program
        /*
LINK    // DVC 21 // VOL DSP185 // LBL RESV$LIB,DSP185
        // LFD RESV$
        // EXEC DLINK,LOAD$LIB,,REL
        // DVC 21 // VOL DSP185 // LBL USERLIB*,DSP185
        // LFD LIB1
LIB     // EXEC LIBUPS,LOAD$LIB,,REL
        // PARAM LIN=(1,NALT1,NALT2)
        /$
        FILL   LIB1,I,1
           ADDL PROGRAM NAME** (MCL)
```

EXECUTION

```
/*
// DVC XX } WHATEVER COMPILED PROGRAM
// DVC XX } HAS ASSIGNED
// EXEC PROGRAM NAME***,USERLIB,LIB1,REL
/&
```

(2) Compiling a source program from a disc source library

```
// JOB name
// DVC 3 // LFD PRNTR
// DVC 20 // VOL DSP155 // DVC 21 // VOL DSP185
// LFD SYSPOOL
// DVC 23 // VOL DSPXXX // LBL NAME,DSPXXX
// LFD LIBIN
// EXEC COBOL,LOADLIB,,REL
// PARAM IN=PROGRAM NAME/LIBIN
// PARAM (your options)
// DVC 21 // VOL DSP185 // LBL RESV$LIB,DSP185

// LFD RESV$
// EXEC DLINK,LOAD$LIB,,REL
// DVC 21 // VOL DSP185 // LBL USERLIB*,DSP185
// LFD LIB1
// EXEC LIBUPS,LOAD$LIB,,REL
// PARAM LIN=(1,NALT1,NALT2)
/$
FILL LIB1,I,1
   ADDL PROGRAM NAME** (MCL)
/*
// DVC XX } WHATEVER COMPILED PROGRAM
// DVC XX } HAS ASSIGNED
// EXEC PROGRAM NAME***,USERLIB*,LIB1,REL
/&
```

*USERLIB may be a temporary disc area from which the program may be executed. If the object program is to be saved, it must be transferred to a private library.

** This is the name assigned by the linker, i.e., PROGRAM ID. IFTEST. ADDL IFTEST00(MCL)

When the problem program is segmented, each segment must be added via ADDL, i.e., IFTEST contains 4 segments:

```
ADDL IFTEST00 }
ADDL IFTEST01 }
ADDL IFTEST02 } (MCL) or ADDL IFTEST.ALL (MCL)
ADDL IFTEST03 }
```

***This is the name assigned the problem program by the PROGRAM-ID clause, i.e., // EXEC IFTEST,USERLIB,LIB1,REL.

I.3. DATA MANAGEMENT INTERFACE AND JOB CONTROL INFORMATION

The following is offered to clarify the use of Data Management by the COBOL compiler. A distinction must be made, however, between the compiler itself, and the code produced by it.

The compiler uses its own input-output routines which are designed solely to meet the needs of the compiler. Data Management is not utilized at compile time.

The code produced by the compiler from source statements makes use of Data Management. This enables compatibility with other languages and a normal amount of flexibility when defining files.

The compiler does a great deal of work for the user on his I-O functions, and, for a better understanding of the requirements and options, the user should be familiar with *UNIVAC 9400 System Data Management System Programmers Reference, UP-7629* (current version).

To supply label information for files on tape volumes, the LBL statement is used. This statement normally follows a VOL statement in the control stream. The LBL statement is required for all standard tape labels.

// VOL XXXXXX (tape number supplied by user)
// LBL file-id, file-serial-number, volume-sequence-number, expiration-date, creation-date, file-sequence-number, generation-number, version-number.

PARAMETER	LENGTH IN CHARACTERS	DESCRIPTION
file-id	1 to 17	Cannot contain imbedded blanks. Name of file as specified in SELECT statement, i.e., SELECT TAPEIN ASSIGN TO TAPE.
file-serial-number	1 to 6	Identical to the first volume serial number appearing on the VOL card.
volume-sequence-number	4	Used with multireel files and is the position of the current reel with respect to the first reel on which the file begins.
expiration-date (b = blank)	6	Expiration date of the file in the form byyddd.
creation-date	6	Creation date of the file in the form byyddd.
file-sequence-number	4	Used with multireel files to assign a numeric sequence for a file within a multifile set.
generation-number	4	Uniquely identifies the edition of a file.
version-number (of generation)	2	Indicates the version of a generation of a file.

I.4. LINKER CONSIDERATIONS

The object module produced by the basic compiler is always written to the tape transport whose LFD name is OBJFIL.

If the transport is at load point at the start of the compilation, the object module will become the first entry in OBJFIL. If the transport is not at load point at the start of the compilation, stacking of object modules is implied, and the compiler output will be placed behind the existing modules already on OBJFIL. The compiler does not rewind OBJFIL at the end of a compilation.

If the single object module produced by the compiler represents the complete user program, OBJFIL should be rewound prior to the start of compilation. The compiler provides the necessary communication for the linkage editor via embedded control cards within the object module output, so that the linking is done without requiring linker control cards in the job stream. All that is necessary, besides device allocation, is an EXEC control card for the linkage editor (see *UNIVAC 9400 System Linkage Editor Programmers Reference, UP-7703* (current version). The linkage editor then processes the first module on OBJFIL and places the executable program back on OBJFIL or, if specified, LDMFIL.

Therefore, for a segmented or nonsegmented COBOL program, the following job stream can be used:

```
// DVC 3    // LFD PRNTR
// OPTION NOVOL
// DVC 5    // VOL NNNNNN    // LFD SYSRES
// DVC 6                     // LFD OBJFIL
// DVC 7                     // LFD SCR1
// EXEC LINK
```

If the COBOL program being compiled represents only a part of a complete user program, the user must define, using the control stream, the structure and position of all modules in the program. The control cards required are defined in *UNIVAC 9400 System Linkage Editor Programmers Reference, UP-7703* (current version).

Nonsegmented COBOL programs may be linked in any fashion, providing the user has ensured compliance with VCON and segment loading requirements. COBOL modules contain ENTRY points for all entry points specified within the program and VCON references for all CALLed program-names. It should be noted that if the user links a COBOL program as an overlay segment, the entire program (including the Data Division) resides in the segment.

The extended compiler places the compiled object module into MCL. Since MCL is initialized at the start of each job, a compilation must be followed by a subsequent job step linking the compiler output, within the same job.

Following is the job stream to link the output of the extended compiler:

```
// DVC 21    // VOL XXXX
// LBL RESV$LIB,DSPXXXX
// LFD RESV$
// EXEC DLINK,LOAD$LIB,,REL
```

To execute the linked program, the program may first be moved to USERLIB:

```
// DVC 21   // VOL XXXX
// LBL USERLIB,DSPXXXX
// LFD LIB1
// OPTION NOVOL
// EXEC LIBUPS,LOAD$LIB,,REL
// PARAM LIN=(1,NALT1,NALT2)
/$
FILL LIB1,I,1
     ADDL  program-name,ALL(MCL)
```

Linked object programs may be executed directly from the MCL area, in which case the preceding job step is unnecessary. Programs that exist in the MCL area are available only for the duration of the job in which they are created.

I.5. SEQUENTIALLY ORGANIZED DISC CONTROL STATEMENTS

The job control statements having sequential organization and used to define a file on disc are described in this paragraph. For a more detailed description of the various options, consult *UNIVAC 9400 System Job Control for Disc Systems Programmer Reference, UP-7585* (current version).

```
// DVC logical-unit [,ALT]
```

A DVC card must be included for each file to define the logical-unit-number of the device on which the first or only volume of the file resides.

The ALT parameter is used for a multivolume file if a second device is available.

```
// VOL volume-number [,volume-number, . . .]
```

This statement contains the volume-serial-number(s) of the disc pack(s) to be mounted on the device whose DVC card it immediately follows.

The volume-number of each volume in the file must be specified on the VOL card in the same order (volume-sequence-number order) as the volumes occur in the file.

```
// EXT { C } ,, { Addr } , Qty
       { N }   { Cyl  }
               { Trk  }
```

This statement provides information necessary for allocation of the file on the device. It is required the first time the program is run if the space for the file has not been previously allocated. Multivolume files must be allocated prior to execution of the program.

```
// LBL file-id, { file-serial-number } [,volume-sequence-number]
                { VCHECK            }
                { blank             }

   [,expiration-date][,creation-date]
```

The file-id identifies the file by name. This name is external to the COBOL program but may, for documentation purposes, be the same as the file-name used in the SELECT and FD entry.

file-serial-number

For files OPENed for OUTPUT, the file-serial-number of the first volume in the file becomes the file-serial-number for all volumes. Files OPENed for INPUT or I-O must have a file-serial-number, on all volumes, equal to the file-serial-number specified on the LBL card.

VCHECK

For files OPENed for OUTPUT, the file-serial-number of all volumes in the fileset is set equal to the volume-serial-number of the first volume. When the file is OPENed for INPUT or I-O, the file-serial-number of all volumes must equal the volume-serial-number of the first volume.

blank

File-serial-numbers are not checked or created.

volume-sequence-number

For files OPENed for OUTPUT, this value should be 1. This value will be incremented by 1 for each additional volume in the file. On INPUT or I-O files, the volume-sequence-number of the first volume must be equal to the volume-sequence-number on the LBL card. Each additional volume in the file must have a volume-sequence-number greater than the previous volume-sequence-number by 1. Volume-sequence-numbers are neither checked nor created if this operand is omitted.

expiration-date

This value is written if the file is OPENed for OUTPUT. It is not checked when the file is OPENed for INPUT or I-O. For documentation purposes, these values may appear in the file description in the VALUE OF clause.

creation-date

This value is written if the file is OPENed for OUTPUT. It is checked when the file is OPENed for INPUT or I-O, and, if it does not match, the job is aborted.

```
// LFD file-name, SQ, nn [,NEW]
```

file-name — The external name assigned to the file in the SELECT entry.

SQ — Defines a sequential file; corresponds to the ORGANIZATION IS SEQUENTIAL clause.

nn — The maximum number of extents which can appear for a volume of this file. The maximum possible is 16.

NEW — Indicates a new file. If this operand is not specified, the file already exists.

The following example shows a COBOL file definition and job control stream.

In this example, a new file of three extents is to be created. The first extent is to be a contiguous area of five cylinders, the second extent, a contiguous area of eight cylinders, while the third is for a contiguous area of four tracks. The name of the file is TEMPORARY FILE, the device name is FILE-A.

```
                                                          COL - 72
IDENTIFICATION  DIVISION.
PROGRAM-ID.     SEQDIS.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.

      SELECT TEMPORARY-FILE
      'FILE-A' DISC ORGANIZATION IS SEQUENTIAL.
       .
       .
       .
DATA  DIVISION.
FILE  SECTION.
FD    TEMPORARY-FILE
      LABEL RECORDS ARE STANDARD
      DATA RECORDS ARE name-1, name-2 ... name-n
      VALUE OF CREATION-DATE IS '69188'
      EXPIRATION-DATE IS '69365'.

01    name-1.
       .
       .
       .
// DVC 20
// VOL DSP006
// EXT C,0,CYL,5                                          X
//1    C,0,CYL,8                                          X
//2    C,0,TRK,4
// LBL TEMPORARY-FILE,,1,69365,69188
// LFD FILE-A,SQ,3,NEW
```

I.6. DIRECT OR RELATIVE ORGANIZATION DISC CONTROL STATEMENTS

The job control statements used to define a file on disc, whose organization is direct or relative, are defined in this paragraph. For a more detailed description of the various options, consult the *UNIVAC 9400 System Job Control for Disc Systems Programmers Reference, UP-7585* (current version).

// DVC logical-unit

A DVC statement must be included for each volume of the DAM file because all volumes of a direct access file must be on-line. There is a maximum of eight volumes to a file.

// VOL volume-number

This statement contains the volume-serial-number of the disc pack to be mounted on the device whose DVC card it immediately follows. One VOL statement should be included for each DVC statement.

```
// EXT {C} ,, {Addr} , Qty
       {N}   {Cyl }
             {Trk }
```

This statement provides information necessary for creation or expansion of the file on the device. It is required the first time the program is run if the space for the file has not been previously allocated.

```
// LBL file-id, {file-serial-number} , [volume-sequence-number
                {VCHECK           }
                {blank            }
   ,expiration-date ,creation-date]
```

The file-id identifies the file by name. This name is external to the COBOL program but may, for documentation purposes, be the same as the file-name used in the SELECT and FD entry.

file-serial-number

For files OPENed for OUTPUT, the volume-serial-number of the first volume in the file becomes the file-serial-number for all volumes. Files OPENed for INPUT or I/O must have a file-serial-number, on all volumes, equal to the file-serial-number specified on the LBL card.

VCHECK

For files OPENed for OUTPUT, the file-serial-number of all volumes in the file is set equal to the volume-serial-number of the first volume. When the file is OPENed for INPUT or I/O, the file-serial-number of all volumes must equal the volume-serial-number of the first volume.

blank

file-serial-numbers are not checked or created.

Volume-sequence number

For files OPENed for OUTPUT, this value should be 1. This value will be incremented by 1 for each additional volume in the file. On INPUT or I/O files, the volume-sequence-number of the first volume must be equal to the volume-sequence-number on the LBL card. Each additional volume in the file must have a volume-sequence-number greater than the previous volume-sequence-number by 1.

Volume-sequence-numbers are neither checked or created if this operand is omitted.

Expiration-date

This value is written if the file is OPENed for OUTPUT. It is not checked when the file is OPENed for INPUT or I/O. For documentation purposes, these values may appear in the file description in the VALUE OF clause.

Creation-date

This value is written if the file is OPENed for OUTPUT. It is checked when the file is OPENed for INPUT or I/O, and, if it does not match, the job is aborted.

```
// LFD file-name,DR,nn  [,NEW]
```

file-name — The external name assigned to the file in the SELECT entry.

DR — Defines a direct or relative access file; corresponds to the ORGANIZATION IS DIRECT or ORGANIZATION IS RELATIVE clause.

nn — The total number of extents which can appear for this file. The maximum possible is 16 per volume. For an eight-volume file with 16 extents per volume, this number would be 128.

NEW — Indicates a new file. If this operand is not specified, the file already exists.

The following examples shows a COBOL file definition and job control stream.

In this example, a new file of four extents on two volumes is to be created. The first extent is to be a contiguous area of five cylinders, the second extent, a contiguous area of eight cylinders, while the third is for a contiguous area of four tracks. The fourth extent is on the second volume and is to be a contiguous area of six cylinders.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     RANDIS.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.


       SELECT TEMPORARY-FILE   ASSIGN TO 'FILEA'
       DISC ACCESS IS RANDOM
       ORGANIZATION IS RELATIVE
       ACTUAL KEY IS DATA-NAME
        .
        .
        .
DATA   DIVISION.
FILE   SECTION.
FD     TEMPORARY-FILE
       LABEL RECORDS ARE STANDARD
       DATA RECORDS ARE name-1, name-2 ... name-n
       VALUE OF CREATION-DATE IS '69365'.

01     name-1.
        .
        .
        .
```

```
// DVC  20                                          COL-72
// VOL  DSP006
// EXT  C,,CYL,5
// 1    C,,CYL,8                                    X
// 2    C,,TRK,4                                    X
// DVC  21
// VOL  DSP007
// EXT  C,,CYL,6
// LBL  TEMPORARY FILE,,1,69365,69188
// LFD  FILEA,DR,4,NEW
```

I.7. INDEXED SEQUENTIAL DISC FILE CONTROL STATEMENTS

The job control statements used to define a file on disc, whose organization is indexed, are described in this paragraph. For a more detailed description of the various options, consult the *UNIVAC 9400 System Job Control Programmers Reference, UP-7585* (current version).

```
// DVC logical-unit
```

A DVC statement must be included for each volume of the ISAM file. All volumes of the file must be on-line. There is currently a maximum of four volumes to a file.

```
// VOL volume-number
```

This statement contains the volume-serial-number of the disc pack to be mounted on the device whose DVC card it immediately follows. One VOL statement should be included for each DVC statement.

```
// EXT C, extent-type, { Addr } , qty
                       { Cyl  }
                       { Trk  }
```

This statement provides information necessary for the creation of the file on the device. It is required, the first time the program is run if space for the file has not been previously allocated. If the file is new, the LFD statement must specify NEW and each DVC-VOL must be followed by an EXT statement.

The first volume of the file must have an EXT statement describing the extent for the optional master index and the required cylinder index.

```
// EXT C, 04, { Addr } , qty
              { Cyl  }
              { Trk  }
```

The first parameter must be C or omitted. The second parameter, extent type, must be 04, to identify an index area extent request. The third and fourth parameters specify the amount of space to be reserved for the index area.

There may be one or two additional extents allocated on the first volume extent; type = 01 defines a prime data area extent and type = 02 defines an independent overflow area extent. Both of these types must begin on cylinder boundaries and must be requested as an integral number of cylinders.

```
// LBL file-id, { file-serial-number } , [volume-sequence-number
                { VCHECK               }
                { blank                }
    ,expiration-date ,creation-date]
```

file-id

Identifies the file by name. This name is external to the COBOL program but may, for documentation purposes, be the same as the file-name used in the SELECT and FD entry.

file-serial-number

For files OPENed for OUTPUT, the volume-serial-number of the first volume in the file becomes the file-serial-number for all volumes. Files OPENed for INPUT or I/O must have a file-serial number, on all volumes, equal to the file-serial-number specified on the LBL card.

VCHECK

For files OPENed for OUTPUT, the file-serial-number of all volumes in the file is set equal to the volume-serial-number of the first volume. When the file is OPENed for INPUT or I/O, the file-serial-number, on all volumes must equal the volume-serial-number of the first volume.

blank

File-serial-numbers are not checked or created.

Volume-sequence-number

For files OPENed for OUTPUT, this value should be 1. This value will be incremented by 1 for each additional volume in the file. On INPUT or I/O files, the volume-sequence-number of the first volume must be equal to the volume-sequence-number on the LBL card. Each additional volume in the file must have a volume-sequence-number greater than the previous volume-sequence-number by 1.

Volume-sequence-numbers are neither checked nor created if this operand is omitted.

Expiration-date

This value is written if the file is OPENed for OUTPUT. It is not checked when the file is OPENed for INPUT or I/O. For documentation purposes, these values may appear in the file description in the VALUE OF clause.

Creation-date

This value is written if the file is OPENed for OUTPUT. It is checked when the file is OPENed for INPUT or I/O, and, if it does not match, the job is aborted.

```
// LFD file-name,IS,nn[,NEW]
```

File-name

The external-name assigned to the file in the SELECT entry.

IS

Defines an indexed sequential file; corresponds to the ORGANIZATION IS INDEXED clause.

nn

The total number of extents which can appear for this file. Only 1 prime data area extent is allowed per volume. The first volume must contain the index area extent. Any volume may contain the independent overflow area extent.

NEW

Indicates a new file. If this operand is not specified, the file already exists.

The following example shows a COBOL file definition and job control stream.

In this example, a new file of three extents on one volume is to be created. The first extent is to be an index area of two tracks. The second extent is to be prime data area of eight cylinders. The third extent is for an independent overflow area of one cylinder.

```
IDENTIFICATION DIVISION.                                    COL - 72
PROGRAM-ID.  INDXSQ.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.

SELECT TEMPORARY-FILE ASSIGNED TO 'FILEA' DISC
ACCESS IS SEQUENTIAL
ORGANIZATION IS INDEXED
RECORD KEY IS DATA-KEY.

DATA DIVISION.
FILE SECTION.

FD TEMPORARY-FILE
   LABEL RECORDS ARE STANDARD
   DATA RECORD IS DATA-RECORD.

01 DATA-RECORD.
   02 DATA-KEY.

//    DVC 20
//    VOL DSP006
//    EXT  C,04,TRK,2                                        X
//1   EXT  C,01,CYL,8                                        X
//2   EXT  C,02,CYL,1
//    LBL TEMPORARY-FILE,,1,69365,69188
//    LFD  FILEA,IS,3,NEW
```

It should be noted that if an ISAM file is to be recreated, the existing file must first be logically deleted from the disc pack VTOC (SCRATCHed). Refer to *UNIVAC 9400 System Data Management System Programmers Reference, UP-7629* (current version).

I.8. USE OF THE COMPILER PATCHING FACILITY

The 9400 System compiler has its own patch routine, which applies patches directly to segments as they are loaded into memory at compilation time. Each segment being patched has a header card which precedes the appropriate patch cards. (Only 1 header card per SEG is acceptable.)

Patch cards are placed in the job stream immediately preceding the IDENTIFICATION DIVISION card, and must be in ascending sequence according to segments involved. Following are the card formats:

HEADER CARD, beginning in card column 1:
$PTCH n SEG
where: n = Segment Identification
PATCH CARD, beginning in card column 1:
$PTCH t aaaa dppppd...
where: t = patch data type H = hexadecimal; C = EBCDIC
a = hexadecimal address of area to be patched
d = delimiter, patch data is bounded by apostrophes
p = contents of patches

Patches are released when necessary by Systems Programming with an explanation of the compiler problem involved.

I.9. COMPILER STATUS INDICATORS

The compiler sets the following status indicators in the user program switch indicator (UPSI) byte. These indicators may be used in conjunction with the // SKIP job control card:

- Switch-0 (X'80') is set to 1 if the compiler does not create a complete object module. This condition might be caused by an "insufficient memory available" diagnostic or a compiler abort.
- Switch-1 (X'40') is set to 1 if the compiler issues any diagnostic message with severity code other than P.

APPENDIX J. COMPILER DIAGNOSTICS AND CONSOLE MESSAGES

J.1. GENERAL

The diagnostic listing is produced as the last printed output of the compiler. Each diagnostic message contains the compiler-generated line number on which the error occurred, the diagnostic severity code, the diagnostic number, and the diagnostic message text.

The tables in this section are arranged by diagnostic number and contain a detailed explanation of each error condition.

The diagnostic severity code definitions are:

P – precautionary – No source language error was encountered, but an unusual or potentially undesirable condition was noted by the compiler.

C – changed – A character, word, clause, entry or statement in the source program is omitted or used incorrectly. To compensate for the error, the item has been changed by the compiler to avoid its deletion and reduce the probability of error propagation. Execution of the object time program may give unpredictable results.

U – uncorrectable – A source language error was detected which caused the compiler to delete a character, word, clause, entry, or statement from the source program. The compilation continues but other errors may result because of the deleted item. Execution of the object program in general, gives unpredictable results.

S – compiler restriction exceeded – The compilation continues but, to generate code for the excessive items, a recompilation is necessary after source program modification or with more storage assigned to the compiler.

J.2. COMPILE TIME DIAGNOSTICS

The following charts explain the error messages and the related recovery procedures. The messages are in ascending order based on the message number.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
001	P	ERROR IN SOURCE LINE SEQUENCE NUMBERING.	The characters in columns 1 to 6 of the source line are alphanumerically less than columns 1 to 6 of the previous source line.	The sequence number, columns 1 to 6 of the source line, is an optional entry used only by the programmer to establish a sequence among the various lines of coding.	The source line is processed as though the error had not occurred.
002	C	AREA-A NON-BLANK WITH HYPHEN IN COLUMN 7.	A nonblank character has been found in area A (columns 8 to 11) when continuation has been specified by a hyphen in column 7.	When continuation is specified by hyphen in column 7, the continued portion must begin in area B (columns 12 to 72).	The first nonblank character after column 7 is accepted as the beginning of continuation.
003	C	ERROR IN COLUMN 7 OF SOURCE LINE.	An invalid character has been found in column 7.	The only acceptable characters for column 7 are the space, hyphen (continuation), or asterisk (comment).	A space is assumed to have been found in column 7.
004	C	SPACE FOLLOWING LEFT PARENTHESIS.	One or more spaces have been detected following a left parenthesis.	In 9400 COBOL spaces must not separate left or right parentheses from that which they enclose.	Processing continues as if the space had not occurred.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
005	C	NON-NUMERIC LITERAL CONTINUATION DID NOT BEGIN WITH QUOTE OR APOSTROPHE.	The continued portion of a nonnumeric literal did not begin with a quote or apostrophe.	When continuation of a nonnumeric literal is specified by a hyphen in column 7, the continued portion must begin with a quote or apostrophe in area B.	Processing continues as if a quote or apostrophe occurred prior to the first nonblank character.
006	C	IMPROPER TERMINATION OF NON-NUMERIC LITERAL *LITERAL*.	The second of the two quotes or apostrophes which enclose a nonnumeric literal is not followed by a space or punctuation and a space. The first 30 characters of the nonnumeric literal are noted in the diagnostic.	The terminating quote or apostrophe enclosing a nonnumeric literal must be followed by a space or period and a space.	Processing continues as if a space had occurred.
007	C	EXCESSIVE CHARACTER STRING *CHAR-STRING*.	A character string which is greater than its maximum legal size has been detected. The first 30 characters of the string are noted in the diagnostic.	Maximum legal sizes are: 132 characters for nonnumeric literals, 20 characters for numeric literals (including sign and decimal point), 30 characters for nonliterals.	Processing continues after the excessive characters are discarded.
008	U	INVALID CHARACTER DETECTED IN *CHAR-STRING*.	An invalid character was found in the character string displayed in the diagnostic.	An invalid character is one which is in the COBOL character set but which is made invalid by the context in which it appears, i.e., PI'TURE.	The entire string is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
009	U	ILLEGAL CHARACTER DETECTED IN *CHAR-STRING*.	An illegal character was found in the character-string displayed in the diagnostic.	An illegal character is one that is not in the cobol character set, **i.e.**, #	The entire string is deleted.
010	C	NON-NUMERIC LITERAL OF SIZE 0 ENCOUNTERED.	Two quotes or apostrophes with no intervening characters were encountered.	A nonnumeric literal must have at least one character between the enclosing quotes or apostrophes.	A nonnumeric literal of one space character is assumed.
011	C	HYPHEN EXPECTED IN COLUMN 7.	A nonnumeric literal is being continued and a hyphen is missing from column 7.	A hyphen in column 7 and a quote or apostrophe in area B are needed to continue a nonnumeric literal.	Processing continues as if a hyphen were encountered.
012	C	HYPHENS IN COLUMN 7 AND QUOTE OR APOSTROPHE EXPECTED.	There is no terminating quote or apostrophe on the previous source line and no hyphen in column 7 or quote or apostrophe on the current source line to indicate continuation.	Continuation of a nonnumeric literal is specified by a hyphen in column 7 and a quote or apostrophe in area B preceding the continued portion of the nonnumeric literal.	The nonnumeric literal is terminated on the previous source line at column 72.
013	C	SPACE PRECEDING A RIGHT PARENTHESIS.	One or more spaces have been detected preceding right parenthesis.	In 9400 COBOL spaces must not separate left or right parentheses from that which they enclose.	Processing continues as if the space had not occurred.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
014	U	SYNTAX REQUIRES———. *CHAR-STRING* INVALID.	The character-string listed as invalid in the message text has produced a syntax error. The required item is a source string that would have correctly completed the clause, entry, or statement in error.	See 9400 COBOL reference manual for language formats.	If the error appears within a clause, such as ACCESS or OCCURS, the clause is deleted. If the error appears within an entry, such as the assign device type or an invalid name following FD, the entire entry is discarded. If the error appears within a statement, the statement is ignored. When a syntax error occurs, source strings are ignored until one of the following listed recovery types is detected, whereupon processing resumes. Recovery is possible on the string listed as invalid in the diagnostic.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
014 (Cont)					IDENTIFICATION, PROGRAM-ID, AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY, REMARKS, ENVIRONMENT CONFIGURATION, SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, any SPECIAL-NAME definition, INPUT-OUTPUT, FILE-CONTROL, SELECT, FOR, FILE-LIMIT, ACCESS, ACTUAL, SYMBOLIC, RELATIVE, ORGANIZATION, RESERVE, I-O-CONTROL, RERUN, SAME, APPLY, DATA, FILE, FD, SD, BLOCK, RECORD, LABEL, RECORDING, DATA, VALUE, OCCURS, PICTURE, USAGE SYNCHRONIZED, JUSTIFIED, BLANK, COMPUTATIONAL, COMP-3, COMP-4, DISPLAY, INDEX, SIZE, MAP, level number, WORKING-STORAGE, LINKAGE, PROCEDURE, PROCEDURE-NAME IN AREA A, any verb.
015	S	COMPILER ERROR	This diagnostic is only issued as the result of a compiler/system error.		The occurrence of this diagnostic should be reported using the SSFR procedure.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
016	U	FILE-NAME *FILE-NAME* NOT PREVIOUSLY SELECTED.	The file-name being referenced has not been defined in a SELECT entry.	A file-name referenced in a RERUN, MULTIPLE, VERIFY, RESTRICTED, BLOCK-COUNT or SAME AREA entry must appear in a SELECT entry.	The referenced file-name is deleted from the entry.
017	U	EXTERNAL-NAME *EXTERNAL-NAME* NOT PREVIOUSLY ASSIGNED.	The external-name being referenced has not been assigned in a SELECT entry.	The external-name specified in a RERUN entry must match the assigned external-name or, if external-name was not specified, the first 8 characters of the SELECT file-name.	The RERUN entry is deleted.
018	U	*CLAUSE* PREVIOUSLY SPECIFIED FOR *FILE-NAME*.	An entry, such as APPLY BLOCK-COUNT, has been multiply specified for the listed file-name.	An entry, such as APPLY BLOCK-COUNT, should be specified only once for a given file.	The duplicate entry is deleted.
019	U	*NAME* PREVIOUSLY DEFINED AS EXTERNAL-NAME OR FILE-NAME.	The listed name appears in more than one SELECT entry.	File-names and external-names specified in SELECT entries must be unique.	The entire SELECT entry is deleted.
020	U	MISSING DATA DIVISION HEADER.	The PROCEDURE DIVISION header has been encountered without prior detection of the DATA DIVISION header.	All four division headers must appear in every program and conform to the following order: IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE.	Processing continues with the PROCEDURE DIVISION header. If Data Division entries exist, they are ignored.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
021	U	MISSING DATA AND PROCEDURE DIVISION HEADER.	The end of the source program has been reached without a DATA DIVISION or PROCEDURE DIVISION header being encountered.	All four division headers must appear in every source program and conform to the following order: IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE.	If Data Division entries or Procedure Division statements exist, they are ignored.
022	C	RESERVE INTEGER *LITERAL* PROCESSED AS 1.	The number of alternate areas specified in the RESERVE clause is not acceptable.	The RESERVE clause must specify no or one alternate area.	One alternate area is allocated for this file.
023	U	FILE-NAME *FILE-NAME* CONFLICTS WITH PREVIOUS SAME AREA CLAUSE.	The listed file-name appears in multiple SAME AREA or SAME RECORD AREA clauses.	A file-name cannot be specified in more than one SAME AREA or SAME RECORD AREA clause.	The file-name in error is deleted from the SAME AREA clause.
024	U	*CLAUSE* CLAUSE IS OUTSIDE SELECT ENTRY.	A clause, such as SYMBOLIC, is not associated with the previously completed SELECT entry.	Clauses associated with a SELECT entry must appear within the entry, e.g., prior to the period that terminates the entry.	The clause is deleted.
025	U	CURRENCY SIGN SYMBOL *CHARACTER* INVALID.	The currency sign specified is not contained within the valid currency sign character set.	The currency sign symbol must be within the COBOL character set but cannot be one of the following: The digits 0 through 9 A B C D P R S V X Z space * + - . ; ( ) or ''.	The clause is deleted and the currency sign remains a $.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
026	P	EXTERNAL-NAME *EXTERNAL-NAME* TRUNCATED.	The external-name contains more than 8 characters.	Only the first 8 characters of the external-name are meaningful.	The excess characters in the external-name are deleted.
027	C	_____ HEADER REQUIRED AT THIS POINT.	The current source line must be preceded by the listed header.	The FILE-CONTROL header must precede the first SELECT entry, the SPECIAL-NAMES header must precede the first special-name, and the I-O-CONTROL header must precede the first RERUN, SAME, APPLY, or MULTIPLE FILE entry.	The header is assumed to have been encountered.
028	C	CLAUSE CONFLICTS WITH ACCESS METHOD SPECIFICATION.	OPTIONAL, RESERVE, and APPLY FILE-PREPARATION are applicable only to disc files with ACCESS SEQUENTIAL and ORGANIZATION SEQUENTIAL APPLY RESTRICTED SEARCH applies only to disc files with ACCESS RANDOM and ORGANIZATION DIRECT.	See Appendix D.	The clause in error is deleted. Line number reflects last statement in the SELECT clause.
029	U	*FILE-NAME* PREVIOUSLY SPECIFIED AS RERUN CONTROLLER.	The listed file-name appears in multiple RERUN entries as the RERUN controller.	A given file may control no more than one RERUN receiver.	The RERUN entry is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
030	U	INVALID SPECIFICATION OF RERUN RECEIVER *EXTERNAL-NAME*	The listed RERUN receiver is not a tape or disc.	RERUN receivers must be assigned to a tape or disc.	The RERUN entry is deleted.
031	S	ADDITIONAL MEMORY REQUIRED FOR SELECT PROCESSING.	The compiler does not have sufficient internal storage to process all of the SELECT entries.	Each SELECT entry requires 26 bytes of storage plus 1 byte for each character in the file-name. To increase the number of SELECTS that can be processed, recompile using smaller file-names or with more storage assigned to the compiler.	This SELECT entry and all others that follow are deleted.
032	U	DUPLICATE CLAUSE OR HEADER.	A clause such as ACTUAL or a header such as AUTHOR has been multiply specified.	All clauses must be unique within their associated entries. All headers must be unique.	The duplicate clause or header is deleted.
033	U	HEADER OUT OF SEQUENCE.	The header on the indicated line number is out of sequence.	The order of headers must be as defined.	The header is deleted.
034	U	CLAUSE APPLIES ONLY TO RANDOM ACCESS FILES.	The clause or entry at the indicated line number applies only to random access files.	VERIFY, RANDOM, RESTRICTED, ORGANIZATION, ACTUAL, SYMBOLIC, RELATIVE, or MULTIPLE apply only to random access files.	The clause or entry is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
035	U	CLAUSE NOT APPLICABLE FOR FILE-NAME.	The clause or entry at the indicated line number is not applicable for the listed file-name.	The following clauses or entries are not applicable for the indicated devices: BLOCK-COUNT – CARD READER, CARD-PUNCH, PRINTER, RANDOM ACCESS DEVICE. MULTIPLE – CARD-READER, CARD-PUNCH, PRINTER. OPTIONAL – CARD-PUNCH, PRINTER.	The clause or entry is deleted.
036	C	INVALID ACCESS-TYPE SPECIFICATION.	An invalid combination of ACCESS, ORGANIZATION, and KEY clauses has been specified.	The combinations of ACCESS, ORGANIZATION and KEY clauses are invalid: See Appendix D, Table D–1.	The file is classified as ACCESS SEQUENTIAL, ORGANIZATION SEQUENTIAL.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
037	C	COPY *STATEMENT REQUIRES* PERIOD.	Something other than a period was found following the library name of a COPY statement.	A period must follow the library name of a COPY statement.	A period is assumed to have been present.
038	C	LABEL RECORDS CLAUSE OMITTED FROM *FILE-NAME*.	A LABEL RECORDS clause has not been specified for the listed file-name.	The LABEL RECORDS clause is required for all files.	LABEL RECORDS OMITTED is assumed.
039	U	MISSING PROCEDURE DIVISION HEADER.	The end of the source program has been reached without detecting the PROCEDURE DIVISION header.	All four division headers must appear in every program and conform to the following order: IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE.	If Procedure Division statements exist, they are deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
040	C	*LITERAL* NOT A VALID LEVEL NUMBER.	The listed level number is erroneous because of its value or use.	(1) Level number values are restricted to 01 through 49, 66, 77, or 88. (2) The level number of the first data description following an FD or SD must be 01. (3) A level number 77 may not be used within the File Section.	(1) If a level number other than 01 through 49, 66, 77, or 88 is encountered; the level number is changed to 49 if the WORKING-STORAGE or LINKAGE SECTION header has not been encountered; otherwise, the level number is changed to 01. (2) If the first data descriptor in a record is not 01, a 01 filler is created by the compiler to precede the current data description. (3) The level number is changed to 01.
041	U	*CLAUSE* CLAUSE INVALID WITH ASSOCIATED LEVEL NUMBER.	The listed clause is not allowed with the specified level number.	(1) A REDEFINES clause may not be used with a level number 66, 77, 88, or a 01 in the File Section. (2) A PICTURE clause may not be used with a level number 66 or 88. (3) The MAP clause is not allowed with level number 66 or 88. (4) Multiple values can only appear with a level number 88. (5) The OCCURS clause is not permitted with a level number 01, 66, 77, or 88. (6) A RENAMES clause can only be used with a level 66. (7) The value clause cannot be used with a level number 66.	In rules 1 through 5, the clause is deleted. For rule 7 the first value is accepted; all others are deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
042	C	REDEFINES MUST BE FIRST CLAUSE.	The REDEFINES clause was not the first clause in the data description.	The REDEFINES clause must immediately follow the name of the data description.	The REDEFINES clause is accepted.
043	U	*CLAUSE* NOT SUPPORTED BY UNIVAC-9400 COBOL.	An obsolete COBOL clause has been encountered.	The SIZE clause is not within the 9400 COBOL language.	The SIZE clause is deleted.
044	C	LEVEL NUMBER *01/77* MUST BE IN AREA-A.	The level number 01 or 77 did not begin in area A.	All 01 or 77 level numbers must start in area A.	The level number is accepted.
045	C	COPY *STATEMENT REQUIRES* LIBRARY *NAME,* (CHAR. STRING) *INVALID.*	A COPY verb was not followed by a library name.	A library name: (1) is composed of no more than 8 characters of the set A through Z, 0 through 9, and the hyphen (-). (2) has at least one alphabet character. (3) does not have a hyphen as the first or last character. (4) is not a COBOL reserved word.	The first 8 characters of the string provided are used as a library name.
046	C	OCCURS CLAUSE INTEGER INVALID.	An OCCURS clause integer is zero or greater than 65,535. (In Format 2 of the OCCURS clause, integer-1 may be zero).	The minimum OCCURS value is 1. The maximum OCCURS value is 65,535.	If zero is used in Format 1 or as integer-2 in Format 2, the OCCURS clause is ignored. If an integer exceeds 65,535 the integer is assumed to be 1.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
047	C	LIBRARY *NAME* (CHARACTER STRING) *EXCEEDS EIGHT CHARACTERS.*	The library name following the COPY verb was found to be longer than 8 characters.	The name of a library structure may be a maximum of 8 characters long.	The first 8 characters of the name provided are used.
048	U	REMAINDER *OF THE LINE* FOLLOWING COPY STATEMENT *MUST BE* BLANK.	A nonblank character was found in the remainder of the line on which the COPY statement appears.	Since the COPY statement directs the compiler to access new lines of COBOL code, nothing may follow the COPY statement on the same line.	The remainder of the line is deleted.
049	C	DATA-NAME, FILE-NAME OR LEVEL NUMBER IN AREA-A	The name or number assigned to the file or data description begins in area A.	File-names, data-names, level number and filler must not begin in area A.	The name or level number is accepted.
050	C	APPLY CLAUSE OR SEGMENT-LIMIT *INTEGER* INVALID.	Cylinder overflow percent was specified as being greater than 90% for 8414 disc or greater than 80% for 8411 disc, the buffer offset value is not from 1 to 99, or the SEGMENT-LIMIT value is not from 1 to 49.	Cylinder overflow percent may not be greater than 90% for 8414 disc or greater than 80% for 8411 disc.	The overflow percent is set to 80 or 90 percent according to the type of disc, the buffer offset is set at 99, or the SEGMENT-LIMIT is set at 49.
051	C	APPLY CLAUSE INTEGER INVALID.	A BLOCK CONTAINS RECORDS clause has been specified with a recording mode of U. Buffer offset value exceeds 99.	Recording mode U states that records of the file are not blocked and may vary in length.	The block contains clause is deleted. The recording mode U is accepted or the buffer offset value is set to 99.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
052	U	CLAUSE NOT ASSOCIATED WITH FD OR DATA-NAME.	A clause, such as DATA RECORDS or PICTURE, is not associated with the previously completed file or data descriptor.	Clauses associated with file or data descriptions must appear within the entry, e.g., prior to the period that terminates the entry.	The clause is deleted.
053	C	NO DATA ENTRY FOR PREVIOUS FD OR SD.	The previous FD or SD does not have at least one record description associated with it.	A record description, with level number 01, must follow every FD or SD description.	The compiler creates a record description whose name is filler. The size of this record is set to the number of bytes specified in the RECORD CONTAINS CHARACTERS clause, if the clause was detected; otherwise, the size is set to 30 bytes.
054	U	FD OR SD NOT IN FILE SECTION.	An FD or SD has been detected outside the File Section.	Every file or sort description must be within the File Section.	The file or sort description is deleted. Any record descriptions following the FD or SD are accepted. They are allocated to either the Working-Storage or Linkage Section, depending on which header was last encountered.
055	C	LEVEL NUMBER *NUMBER* ENCOUNTERED PRIOR TO SECTION HEADER.	A data descriptor has been encountered prior to detection of a DATA DIVISION header.	If a data descriptor is the first entry in the Data Division, it must be preceded by a WORKING-STORAGE or LINKAGE SECTION header.	The compiler assumes the WORKING-STORAGE SECTION header has been encountered and allocates the data item to that section.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
056	U	LANGUAGE ELEMENT NOT IMPLEMENTED.	A COBOL language feature not supported in the basic compiler has been encountered.	The following language elements are not available with the basic compiler: SYSERR, RANDOM, ORGANIZATION, VERIFY, MASTER-INDEX, CYLINDER-INDEX, CYLINDER-OVERFLOW, EXTENDED-INSERTION, FILE-PREPARATION, KEYS, MULTIPLE, VALUE, CONDITION-NAMES, SD, COMPUTE, INSERT, SORT, RETURN, RELEASE, PERFORM FORMAT 3 AND 4, AND, OR, CORRESPONDING, ARITHMETIC EXPRESSIONS. The following language elements are not available in either compiler: I-O, CALL, entry within USE procedure	The clause, entry, or statement is deleted.
057	U	DATA ENTRY REQUIRES RENAMES OR VALUE CLAUSE.	A data descriptor with level number 66 has no RENAMES clause or a data descriptor with a level number of 88 has no VALUE clause.	A data descriptor whose level number is 66 must have a RENAMES clause, and a data descriptor whose level number is 88 must have a VALUE clause.	The data description is deleted.
058	U	LEVEL 88 *CONDITION-NAME* NOT PRECEDED BY DATA ENTRY.	The level 88 entry is the first entry in the Data Division.	See rules for CONDITION-NAME.	The compiler creates a level 01 named FILLER, length 1, signed for the conditional variable.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
059	U	LEVEL 66 *DATA-NAME* MUST APPEAR ONLY AT END OF HIERARCHY.	The level 66 entry was not followed by one of the following: a level 01 entry, an FD or SD entry, a level 77 entry, a level 66 entry, or a PROCEDURE DIVISION header.	See rules for RENAMES.	A level 01 named FILLER is created to follow the level 66 entry.
060	U	OCCURS DEPENDING CONFLICT ASSOCIATED WITH *DATA-NAME*	The data-name with the DEPENDING option of the OCCURS clause is not the last group entry in an 01 hierarchy or the data-name is subordinate to another OCCURS clause.	See rules for OCCURS clause with the DEPENDING option.	The DEPENDING option of the OCCURS clause is ignored, (maximum number of occurrences is assumed).
061	U	LEVEL NUMBER *LITERAL* IS NOT SUBORDINATE TO AN 01.	A data entry with a level between 02 and 49 follows a level 77 or DATA DIVISION header.	See rules for level number.	A level 01 named FILLER is created to precede the data entry.
062	U	USAGE IS INDEX IS INVALID FOR *DATA-NAME* WITH *CLAUSE* CLAUSE.	An additional clause was specified for a data entry with USAGE IS INDEX clause specified.	See rules for USAGE.	Compiler deletes USAGE IS INDEX clause specified for this data entry.
063	U	*DATA-NAME* DESCRIBED AS *TYPE* ITEM HAS *TYPE* INITIAL VALUE.	A class error between a data entry and the value specified for this entry was detected.	See rules for VALUE IS.	The compiler deletes the value specified for this data entry.
064	U	PICTURE INVALID FOR GROUP ITEM *DATA-NAME*.	The data entry was determined to be a group item from level number structure and a PICTURE clause conflicts with a group entry.	See rules for PICTURE.	The compiler deletes the PICTURE clause on the group item.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
065	U	*DATA-NAME* DESCRIBED AS *TYPE, CLAUSE* INVALID.	Conflict between description clauses of the data entry, i.e., data-name described as NUMERIC, JUSTIFIED invalid.	See Section 5 for rules for clauses in conflict.	The clause listed as invalid is deleted.
066	U	INITIAL VALUE INVALID, FOR *DATA-NAME*, IN FILE OR LINKAGE SECTION.	A value clause, not on a CONDITION NAME entry, was noted for an entry in the File or Linkage Sections of the Data Division.	See rules for VALUE IS.	The compiler deletes the VALUE clause on the data entry.
067	U	INITIAL VALUE INVALID FOR *DATA-NAME* SUB ORDINATE TO REDEFINES OR OCCURS.	A VALUE clause, on a data entry which was subordinate to either a REDEFINES, OCCURS, or VALUE clause, was found.	See rules for VALUE IS.	The compiler deletes the VALUE clause on the data entry.
068	U	*DATA-NAME* USAGE COMP-3 FROM PICTURE CONFLICT WITH USAGE *TYPE* CLAUSE.	A data entry with an H in the PICTURE (implying COMP-3) and a conflicting USAGE clause were noted.	See rules for PICTURE and USAGE.	The compiler assumes COMP-3 as valid and deletes the other USAGE clause.
069	C	SAME SORT OR SAME RECORD AREA CONFLICTS WITH SAME AREA CLAUSE.	Not all files in the SAME AREA clause appear in the SAME SORT AREA clause or SAME RECORD AREA clause.	See rules for SAME AREA clause.	The file is processed as though it had appeared in the SAME RECORD or SAME SORT AREA clause.
070	U	OCCURS CLAUSE INVALID FOR *DATA-NAME*, A *LEVEL #* ENTRY.	An OCCURS clause was noted on a data entry with a level number of 01 or 77.	See rules for OCCURS.	The compiler deletes the OCCURS clause.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
071	C	FILLER INVALID FOR *TYPE* ENTRY.	Data-name FILLER was on a data entry with a level of 66, 88, or a group item.	See rules for level number and data-name FILLER.	The compiler assumes a level of 01.
072	U	*DATA-NAME* CONTAINS BOTH A REDEFINES AND OCCURS CLAUSES.	A data entry was noted with both a REDEFINES clause and an OCCURS clause, i.e., each occurrence will redefine the area (not a table).	See rules for OCCURS.	The compiler deletes the OCCURS clause on the data entry.
073	C	ONE LEVEL NUMBER ALLOWED PER LINE.	More than one level number appears on the indicated line number.	See formats of the DATA DIVISION.	The level number is processed as though it were on a unique line number.
074	C	USAGE OF *DATA-NAME* CONFLICTS WITH USAGE OF GROUP.	A data entry usage conflicts with the usage of one or more of the group entries which this data entry is subordinate to or usage conflicts with a value on a group level.	See rules for USAGE and VALUE IS.	Compiler assumes entries usage as proper usage.
075	U	THE OCCURS CLAUSE ON *DATA-NAME* INVALID, 4 DIMENSION TABLE DESCRIBED.	A data entry with an OCCURS clause which would cause more than three levels of subscripting was encountered.	See rules for OCCURS.	The compiler deletes the OCCURS clause on the data entry.
076	U	FILE *FILE-NAME* HAS NO DATA RECORD.	A level 01 data record was not encountered for this file.	Format violated, see FILE SECTION. NOTE: There must be a data record description for each file.	No action is taken by the compiler.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
077	C	BLOCK-LENGTH-CHECK INVALID WITH RECORDING MODE.	Block-length-check is not allowed with all recording modes.	Block-length-check is appropriate with recording mode V or D.	The block-length-check is disregarded.
078	S	ADDITIONAL MEMORY REQUIRED FOR LABEL RECORD PROCESSING.	There is not enough memory available for holding of all the label name definitions for this file.	N/A	Compiler assumes label name definitions that will not fit, do not exist. Memory is required to hold the SELECTS and label name definitions. To allow processing of more label names, allocate more memory, shorten the size of the SELECTS, or define fewer label names.
079	U	BLOCKING CONFLICTS WITH ORGANIZATION ON FILE *FILE-NAME*.	A file with organization direct or relative with a blocking factor was encountered (blocking from BLOCK CONTAINS clause).		The compiler deletes the BLOCK CONTAINS clause.
080	C	FILE-NAME *FILE-NAME* DOES NOT APPEAR IN A SELECT.	A file which does not have a SELECT entry (matched by file-name) was encountered.	See rules for FILE-CONTROL	Compiler assumes a SELECT entry defined with file-name (of file) assigned to tape-6.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
081	C	INVALID RECORDING MODE FOR FILE *FILE-NAME*	(1) A file assigned to card reader and recording mode was V or U. (2) File assigned to DISC-8411, DISC-8414 with ORGANIZATION RELATIVE or INDEXED and RECORDING MODE was V or O. (3) File assigned to DISC-8411, DISC-8414 with ORGANIZATION DIRECT and RECORDING MODE was V.	Device restriction (card reader) access method restriction (DISC, DISC-8414)	Compiler assumes recording mode F for this file.
082	C	80 CHARACTER BLOCK LIMIT EXCEEDED BY CARD FILE *FILE-NAME*.	A BLOCK CONTAINS clause exceeds the maximum for a card device.	See rules for BLOCK CONTAINS.	The compiler assumes the maximum size (80) for BLOCK CONTAINS.
083	C	BLOCK CONTAINS EXCEEDS 1 RECORD ON CARD-READER FILE *FILE-NAME*.	A file assigned to a card device with BLOCK CONTAINS 2 or more records specified was encountered.	Device restriction	Compiler assumes BLOCK CONTAINS 1 record.
084	C	FILE *FILE-NAME* MUST HAVE LABEL RECORDS OMITTED.	A file assigned to a unit record device with other than LABEL RECORDS OMITTED was encountered.	Data management restriction.	Compiler assumes labels to be omitted.
085	C	BLOCK SIZE SPECIFIED FOR FILE *FILE-NAME* EXCEEDS MAXIMUM.	BLOCK CONTAINS clause contains value which exceeds maximum length for the device the file is assigned to.	See BLOCK CONTAINS.	The compiler assumes that the maximum length was specified.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
086	C	BLOCK SIZE SPECIFIED FOR FILE *FILE-NAME* LESS THAN MINIMUM.	A BLOCK CONTAINS clause value was encountered which is less than the minimum allowed for the device.	The compiler assumes the minimum length for the BLOCK CONTAINS clause.	See BLOCK CONTAINS.
087	U	DESCRIPTION FOR LABEL RECORD *LABEL NAME* NOT ENCOUNTERED.	A label name (from LABEL RECORDS ARE clause) with no 01 label description was encountered.	See rules for label records.	The compiler assumes that the label name does not exist.
088	C	FILE *FILE-NAME* MUST HAVE LABEL RECORDS STANDARD OR DATA NAME.	*filename* is assigned to direct access device (DISC, DISC-8414) and the LABEL RECORDS clause specified omitted.	File assigned to disc must have a LABEL RECORDS specification.	Compiler assumes LABEL RECORDS ARE STANDARD for the file.
089	C	FILE *FILE-NAME* MUST HAVE LABEL RECORDS STANDARD.	*filename* is assigned to a direct access device (DISC, DISC-8414) and has organization indexed, and LABEL RECORDS ARE OMITTED or data name is specified.	File with organization indexed must have LABEL RECORDS STANDARD.	Compiler assumes label records to be standard for the file.
090	C	FILE *FILE-NAME*, BLOCK CONTAINS CHARACTERS CLAUSE NOT ALLOWED.	File assigned to direct access device (DISC, DISC-8414) with ORGANIZATION INDEXED and BLOCK CONTAINS CHARACTERS was encountered.	File with organization must have BLOCK CONTAINS RECORDS or no BLOCK CONTAINS clause.	Compiler assumes there was no BLOCK CONTAINS clause specified.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
091	U	COPY SYNTAX REQUIRES _____ , *CHARACTER-STRING* INVALID.	The character-string listed as invalid has produced a syntax error. The required type of character-string is indicated.	See 6.7.7 for COPY rules.	The item in error and all items which follow it in the COPY clause are deleted.
092	S	REPLACING AREA OVERFLOW CAUSED BY *CHARACTER-STRING*.	The internal storage area used to save REPLACING items has been exhausted or the number of qualifiers associated with an identifier has exceeded internal storage area.	Compiler Restriction	The compiler ignores the balance of the clause which causes overflow. Recompile with additional memory allocated to the compiler or reduce the number of items, amount of qualification, or size of names in the REPLACING clause.
094	C	CHARACTER NUMBER *LITERAL* IS INVALID IN *TYPE* PICTURE *PICTURE-STRING*.	An illegal PICTURE character, a PICTURE character inconsistent with the PICTURE type, or a violation of the PICTURE precedence rules has been detected.	See Section 5 for the allowable PICTURE symbols and the rules for their usage.	In order not to delete the data descriptor, the compiler sets its PICTURE to S9.
095	C	THE *TYPE* PICTURE *PICTURE-STRING* IS INCOMPLETE.	As stated, the picture is incomplete and cannot be processed, e.g., SPPPP.	See Section 5 for the allowable PICTURE symbols and the rules for their use.	In order not to delete the data descriptor, the compiler sets its PICTURE to S9.
096	C	CHARACTER NUMBER *LITERAL* IS INVALID IN PICTURE *PICTURE-STRING*.	An illegal PICTURE character, a PICTURE character inconsistent with the PICTURE type, or a violation of the PICTURE precedence rules has been detected.	See Section 5 for the allowable PICTURE symbols and the rules for their usage.	The PICTURE characters prior to the character in error are accepted.
097	C	SIZE LIMIT OF *LITERAL* BYTES EXCEEDED BY PICTURE *PICTURE-STRING*.	The PICTURE specifies more storage than the maximum allowed for the PICTURE type.	The maximum size in bytes of numeric PICTURE is 18, alphabetic or alphanumeric is 4092, numeric edited or alphanumeric edited is 132.	In order not to delete the data descriptor, the compiler sets its PICTURE to S9.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
098	C	THE NUMBER OF DIGIT POSITIONS IN PICTURE *PICTURE-STRING* EXCEEDS 18.	The number of digit positions in the PICTURE exceeds 18.	The maximum number of digits allowed in a numeric or numeric edited PICTURE is 18.	In order not to delete the data descriptor, the compiler sets the PICTURE to S9.
099	C	A VALUE CONTAINED WITHIN PARENTHESES IS *=0 OR >4092* IN PICTURE *PICTURE-STRING*.	A value contained within parentheses is either zero or greater than 4092.	The number of times a PICTURE character is repeated, as specified by the value in parentheses following it, must be greater than zero and less than 4093.	The value within the parentheses is set to 1 and processing of the PICTURE continues.
100	C	A NUMBER DOES NOT FOLLOW A LEFT PARENTHESIS IN PICTURE *PICTURE-STRING*.	A left parenthesis within the PICTURE is not followed by a numeric integer.	Within parentheses, a numeric integer is used to specify the number of times the preceding PICTURE character is repeated.	In order not to delete the data descriptor, the compiler sets the PICTURE to S9.
101	C	RIGHT PARENTHESIS MISSING FROM PICTURE *PICTURE-STRING*.	A right parenthesis does not follow a numeric integer preceded by a left parenthesis.	Each left parenthesis in a PICTURE must be followed by a numeric integer and a right parenthesis.	In order not to delete the data descriptor, the compiler sets the PICTURE to S9.
102	C	BOTH LEADING AND TRAILING SIGN INSERTION SPECIFIED IN PICTURE *PICTURE-STRING*.	Two insertion sign characters have been encountered in the numeric edited PICTURE.	Specification of both leading and trailing sign insertion is not permitted.	In order not to delete the data descriptor, the compiler sets the PICTURE to S9.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
105	C	INITIAL VALUE TRUNCATED.	The value specified for the data item contains a greater number of characters than the data item, or is a numeric value that, when the decimal point is aligned, is larger than the maximum value the data item can contain.	The initial value cannot contain more characters then can fit into the data item.	The excess characters are truncated.
106	U	INVALID POSITIONING OF KEY *DATA-NAME* IN HIERARCHY.	There must not be any items with an OCCURS clause between the table item and its keys.	See rules for KEYS under OCCURS clause.	The named KEY is processed as a regular data item, the KEY information is ignored.
107	S	ADDITIONAL MEMORY REQUIRED TO PROCESS HIERARCHY CONTAINING *DATA-NAME*.	There is not enough memory available to contain all entries subordinate to the 01 data entry. There are too many entries for the 01 hierarchy for memory allocated.		The compiler will not process the data entries which are not contained in memory. To compensate, shorten the hierarchy, shorten names in data entries, or assign more memory to compiler.
108	S	*DATA-NAME* EXCEEDS REDEFINES NESTING LIMIT.	There are too many levels of redefinition. This data entry exceeds the limit of redefinition.	See rules for REDEFINES.	The compiler assumes this entry does not have REDEFINES clause.
109	C	*DATA-NAME* WAS IMPROPER REDEFINES OBJECT *DATA-NAME*.	The redefined area is a redefining area; i.e., the object of the REDEFINES clause has or is subordinate to a REDEFINES clause.	See rules for REDEFINES.	The compiler assumes the redefinition of the last defined area with the same level as the subject of the REDEFINES clause.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
110	S	ADDITIONAL MEMORY REQUIRED TO PROCESS RENAMES QUALIFIER.	There is insufficient memory available to contain the RENAMES qualifier. This is due to a large hierarchy and/or a lot of RENAMES qualifiers.		The compiler assumes the qualifier does not exist.
111	U	DESCRIPTION OF *DATA-NAME* NOT ENCOUNTERED.	The definition of the entry is not in the current hierarchy.	See rules for qualification.	The compiler assumes the qualifier name in error does not exist.
112	C	RENAMES-OCCURS CONFLICT BETWEEN *DATA-NAME-1* and *DATA-NAME-2*.	The object of the RENAMES clause on data-name-1 has or is subordinate to an OCCURS clause.	See rules for level number.	The compiler assumes the last elementary item in the hierarchy is the object of the RENAMES clause.
113	C	REDEFINING AREA *DATA NAME* UNEQUAL TO SIZE OF REDEFINED AREA.	The calculated length of the redefined area is not the same as the length of the redefining area.	See rules for REDEFINES.	The compiler assumes the largest length was calculated for both areas.
114	C	SIZE OF ELEMENTARY ITEM *DATA-NAME* EXCEEDS MAXIMUM OF 4092.	An elementary item with a length larger than the maximum was encountered.	See data definition.	The compiler assumes the length to be 4092 for the elementary item.
115	C	SIZE OF WORKING-STORAGE GROUP ITEM *DATA-NAME* EXCEEDS MAXIMUM 65,535.	A group entry in WORKING-STORAGE has a length calculated to exceed the maximum.	See data definition.	The compiler assumes the length of the group item to be 65,535. The entire area specified is, however, allocated.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
116	C	SIZE OF NON-WORKING-STORAGE GROUP ITEM *DATA-NAME* EXCEEDS MAXIMUM OF 4092.	The length of a file or LINKAGE SECTION group item was calculated to be greater than the maximum.	See data definition.	The compiler assumes the maximum of 4092 was the calculated length of the group item.
117	U	INVALID LEVEL NUMBER STRUCTURE ENCOUNTERED AT *DATA-NAME*.	A level number equal to the level of the data entry should have appeared in the hierarchy directly subordinate to the 01.		The compiler assumes there was a level number on a data entry directly subordinate to the 01, i.e., 01 A LEVEL 02 MISSING 05 B 02 C INVALID LEVEL STRUCTURE
118	C	THE FIRST OBJECT OF LEVEL 66 ENTRY *DATA-NAME* ENDS AFTER THE SECOND OBJECT.	The first object of a RENAMES clause does not precede the area of the second object of the RENAMES clause.	See rules for RENAMES.	The compiler assumes the second object does not exist.
119	C	THE SECOND OBJECT OF THE LEVEL 66 ENTRY *DATA-NAME* STARTS BEFORE THE FIRST OBJECT.	The second object of a RENAMES clause does not precede the first object of the RENAMES clause.	See rules for RENAMES.	The compiler assumes the objects are reversed. (The first is the second and the second is the first.)

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
120	C	USAGE INDEX INVALID FOR CONDITIONAL VARIABLE *DATA-NAME*.	A condition name entry is defined for a data entry with a USAGE INDEX clause.	See rules for condition name.	The compiler assumes alphanumeric usage for the conditional variable.
121	C	RECORD *DATA-NAME* IS NOT SAME SIZE AS ALL PREVIOUS RECORDS IN A FIXED RECORDING MODE FILE.	A file described as F RECORDING MODE does not have data records with the same length.	See rules for RECORDING MODE.	The compiler assumes the largest data record length for calculation of record length for the file.
122	C	LABEL RECORD *DATA-NAME* SIZE NOT EQUAL 80 CHARACTERS.	A label record description with a length other than 80 was encountered.	9400 label specification has a length of 80 for labels.	The compiler assumes the length of label records to be 80.
124	C	BLOCK SIZE FOR *FILE-NAME* SMALLER THAN LARGEST RECORD.	The BLOCK CONTAINS CHARACTERS clause specifies a block length smaller than length of largest data record.		The compiler assumes the block length to be the length of the largest record.
125	C	SIZE OF *DATA-NAME* GREATER THAN RECORD CONTAINS FOR FILE *FILE-NAME*.	The RECORD CONTAINS clause specifies a record length smaller than largest record.		The compiler assumes that the largest hierarchy subordinate to the FD, specifies the length of the largest data record for the file.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
126	C	*FILE-NAME CLAUSE* LENGTH *CONDITION* ALLOWED FOR DEVICE.	The BLOCK CONTAINS clause or the RECORD CONTAINS clause exceeds maximum or is less than minimum for the device which the file is assigned to.	See BLOCK CONTAINS and RECORD CONTAINS.	The compiler assumes the limiting length for the clause in error.
127	C	RECORD CONTAINS CLAUSE FOR FILE *FILE-NAME* NOT EQUAL TO SIZE OF LARGEST RECORD.	The RECORD CONTAINS clause does not specify the length of the largest data record.		The compiler assumes that the length of the largest data record is specified in the RECORD CONTAINS clause.
128	P	BLOCK LENGTH OF FILE *FILE-NAME* PROHIBITS RUN TIME SPECIFICATION OF BLOCK NUMBERING.	The length of the block for the file is too large to allow block numbering.		No action. Precautionary warning.
129	U	REDEFINES NOT PERMITTED FOR RECORDS IN FILE SECTION.	A File Section level 01 with a REDEFINES clause was encountered.	See rules for REDEFINES.	The compiler assumes the REDEFINES clause does not exist.
130	U	SUBJECT OF REDEFINES *DATA-NAME* NOT IN SAME SECTION AS OBJECT OF REDEFINES.	The subject of a REDEFINES clause is not in same section as entry with REDEFINES.	See rules for REDEFINES.	The compiler assumes the REDEFINES clause does not exist.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
131	U	OBJECT OF REDEFINES, *DATA-NAME*, WITHIN RANGE OF OCCURS.	The object of a REDEFINES clause has or is subordinate to an OCCURS clause.	See rules for RENAMES.	The compiler assumes the REDEFINES clause does not exist.
132	U	REDEFINES OBJECT *DATA-NAME*, AND SUBJECT DATA-NAME DO NOT HAVE SAME LEVEL NUMBER.	The object and subjects of the REDEFINES clause do not have the same level numbers.	See rules for REDEFINES.	The compiler assumes the REDEFINES clause does not exist.
133	S	INDEX NAME *DATA-NAME* EXCEEDS COMPILER LIMITS.	The current compiler limit of index-names is 255. This entry is the 256 specified index-name.		The compiler starts index-name memory assignment over and reassigns the memory to the index-names being processed.
134	C	ELEMENTARY ITEM DATA-NAME HAS NO LENGTH SPECIFIED.	An elementary item, determined from level number structure, with no length specified or assumed was encountered.		The compiler assumes a length of 1, signed was specified.
135	C	OBJECT OF RENAMES *DATA-NAME* NOT FOUND WITHIN HIERARCHY.	The object of the RENAMES clause was not found in the immediate hierarchy.	See rules for RENAMES.	The compiler assumes the last elementary item of the heirarchy as the specified object of the RENAMES clause.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
136	C	OBJECT OF RENAMES *DATA-NAME* HAS ILLEGAL LEVEL NUMBER.	The object of the RENAMES clause has illegal level number.	See rules for RENAMES.	The compiler assumes the last elementary item as specified object of the RENAMES clause.
137	U	REDEFINES CLAUSE IN *DATA-NAME* HAS INVALID OBJECT.	The object of the REDEFINES clause is not a legal level for redefinition.	See rules for REDEFINES.	The compiler assumes the REDEFINES clause does not exist.
138	S	ADDITIONAL MEMORY REQUIRED FOR PROCEDURE NAME PROCESSING.	The compiler needs more memory in order to process the rest of the section and paragraph names.	Each procedure-name definition requires 16 bytes of storage plus 1 byte for each character in the name. To increase the number of procedure-names that can be processed, recompile using smaller names or with more storage assigned to the compiler.	This procedure-name definition and all others that follow are deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
139	C	PRIORITY NUMBER INCORRECT OR OUT OF SEQUENCE.	Priority number value does not fall in range of 0 to 99 or priority number >50 is not in ascending sequence.	The priority number must be an integer ranging in value from 0 through 99. Segments with priority number 50-99 are independent segments and must appear in the source program in ascending numeric order.	If segmentation has been specified (a previous segment with priority number >50) the last valid priority number is assigned to this section. If segmentation has not been encountered, a priority number of 0 is assumed.
140	U	NEITHER EXIT PROGRAM NOR RETURN STATEMENT ASSOCIATED WITH ENTRY OR USING STATEMENT.	An entry point has been specified for this program but the program contains no mechanism to return to caller.	All COBOL subprograms must contain either an exit program or a RETURN statement.	No corrective action is possible for this error. If the program is executed as a subprogram it will not return to the calling program.
141	U	NEITHER ENTRY NOR USING STATEMENT ASSOCIATED WITH EXIT PROGRAM OR RETURN STATEMENT.	Program contains mechanism to return to a calling program but no mechanism has been coded where the calling program may enter this program.	A COBOL program that is to be used as a subprogram must have an entry point.	No corrective action is possible for this error. It is impossible to execute this program as a subprogram.
142	U	NO ENTRY OR RETURN STATEMENT ASSOCIATED WITH LINKAGE SECTION.	No entry point has been specified for this subprogram.	The use of the LINKAGE SECTION implies that this is a subprogram. Subprograms must have entry and exit points.	No corrective action is taken.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
143	U	STRUCTURE OF CONDITIONAL SENTENCE INVALID, UNPAIRED ELSE ENCOUNTERED.	ELSE encountered in IF statement with no preceding IF verb to match it with.	In a conditional statement any ELSE encountered is considered to apply to the immediately preceding if that has not been already paired with an ELSE.	The conditional statement is terminated at this point.
144	P	PROCEDURE DIVISION DOES NOT CONTAIN A STOP RUN.	No STOP RUN statement is coded in this program. There is no way to bring this program to an orderly halt.	No rule has been violated; this diagnostic is strictly informative.	Results during execution are unpredictable.
145	U	EXIT WAS NOT THE ONLY STATEMENT IN PARAGRAPH.	EXIT statement is in paragraph which contains statements other than EXIT.	The EXIT sentence must be preceded by a paragraph-name and be the only sentence in the paragraph.	Nothing is deleted from the program but the statements following the EXIT verb are never executed.
146	C	THE BEFORE OPTION OF THE USE STATEMENT IS NOT APPLICABLE IN UNIVAC-9400 SYSTEM.	The BEFORE option is not allowed in UNIVAC 9400 System.	The BEFORE option is not applicable to the UNIVAC 9400 System, but is accepted for compatibility.	The AFTER option is assumed.
147	C	THE PROGRAM NAME IN CALL STATEMENT EXCEEDS EIGHT CHARACTERS.	Program name exceeds 8 characters in length.	A maximum of 8 characters is allowed in subprogram names.	The program name in CALL statement truncated to 8 characters.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
148	U	REFERENCE TO *NAME* CANNOT BE RESOLVED.	A definition of the listed name has not been encountered.	Every name referenced must be defined.	The statement containing the reference is deleted.
149	U	QUALIFIED REFERENCE TO *NAME* CANNOT BE RESOLVED.	A definition of the listed name has not been encountered under the specified qualifiers.	Every name referenced with qualification must be defined within the hierarchy associated with the highest level qualifier.	The statement containing the reference is deleted.
150	C	REFERENCE TO PROCEDURE *NAME* IS AMBIGUOUS, DEFINITION AT LINE *LITERAL* USED.	A definition of the listed paragraph-name has not been encountered within the section from which the reference is made, while multiple definitions exist outside the section of reference.	A reference to a non-unique paragraph-name where all definitions are outside the section from which the reference is made must be qualified.	The reference is resolved by the paragraph-name at the listed line number.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
151	U	REFERENCE TO *NAME* OF *NAME*, CANNOT BE RESOLVED DUE TO DEFINITION AT LINE *LITERAL*.	Normally this diagnostic indicates that a definition for the qualifier in a procedure reference has been encountered but is not a section-name. In the ambiguity mode of reference resolution (PARAM LST=A) this diagnostic is also generated when: (1) The highest qualifier of a data reference is not encountered in the Data Division but is encountered in the Procedure Division. (2) The qualifier of a procedure reference is not encountered in the Procedure Division but is encountered in the Data Division. This emplies that when the definition that will resolve the reference is added to the source program, the highest possible qualifier rule is violated.	The qualifier in a procedure reference must refer to a section-name. Highest possible qualifiers (level indicator names, section-names, level 01 and 77 names) must be unique in a program since a reference to the name cannot be qualified.	The statement containing the reference is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
152	C	REFERENCE TO *NAME* AMBIGUOUS DUE TO DEFINITION AT LINE *LITERAL*, DEFINITION AT LINE *LITERAL* USED.	This diagnostic is generated only in the ambiguity mode of reference resolution (PARAM LST=A) for an unqualified reference when a duplicate definition of the listed name has been encountered within the COBOL division implied by the reference type, e.g., GO TO implies Procedure Division; MOVE implies Data Division.	Every name in a COBOL program must be unique either because of different spelling or because of qualification.	The reference is resolved by the name at the listed line number.
153	C	IMPROPER DEFINITION OF *NAME* AT LINE *LITERAL* IMPLIED BY MANNER OF REFERENCE.	This diagnostic is generated only in the ambiguity mode of reference resolution (PARAM LST=A) for an unqualified reference when a duplicate definition of the listed name has been encountered in a COBOL division other than the division implied by the reference type and constitutes a violation of the highest possible qualifier rule.	Highest possible qualifiers (level indicator names, section-names, level 01 and 77 names) must be unique since a reference to the name cannot be qualified.	If the reference cannot be resolved within the COBOL division corresponding to the reference type, the statement is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATON		
			REASON	RULE	RECOVERY
154	C	*NAME* MUST BE UNIQUE DUPLICATE DEFINITION FOUND AT LINE *LITERAL*.	This diagnostic is generated only in the ambiguity mode of reference resolution (PARAM LST=A) for qualified references when a redefinition of the highest qualifier violates the highest possible qualifier rule.	Highest possible qualifiers (level indicator names, section-names, level 01 and 77 names) must be unique since a reference to the name cannot be qualified.	If the reference cannot be resolved within the COBOL division corresponding to the reference type, the statement is deleted.
155	C	*BEFORE* OPTION NOT APPLICABLE IN C-MODE.	The WRITE BEFORE ADVANCING option is not available in the conversion mode.	Compatibility requirement.	The BEFORE option is treated as though the AFTER option had been specified.
159	U	*VERB* STATEMENT CONTAINS INVALID OPERAND *DATA-NAME*.	The specified data item does not satisfy the requirements for the designated verb, for example, an alphabetic operand in an ADD statement.	See the general rules specified for the designated verb.	The statement containing the listed operand is deleted.
160	U	*VERB* STATEMENT OPERAND *DATA-NAME* IS IMPROPERLY SUBSCRIPTED.	The data item contains too many, too few, or an improper type of subscript.	References to items in a table must have the correct number of subscripts or indexes, subscripts must be unsigned numeric integers, subscripts and indexes must not be mixed in a single data reference, and references to items not in a table must not be subscripted.	The statement containing the subscript error is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
161	U	*VERB* STATEMENT CONTAINS INCONSISTENT OPERAND *DATA-NAME*.	The combination of operands in the statement conflict in their usage, for example, moving a numeric item to an alphabetic operand.	See the rules for the indicated verb statement.	The statement containing the inconsistent operand is deleted.
162	C	*VERB* STATEMENT CONTAINS SIGNED LITERAL *LITERAL*.	A signed literal has been encountered.	See the specific rules for the designated verb.	The sign of the literal is deleted.
163	U	COMPOSITE OF OPERANDS IN *VERB* STATEMENT EXCEEDS 18 DIGITS.	The superimposition of all operands to the left of the word giving exceeds 18 digits.	See rules for composite of operands for the specified verb.	The statement containing the composite error is deleted.
164	U	GO TO PRECEDES IMPERATIVE STATEMENT.	A GO TO statement is followed by other imperative statements.	A GO TO statement must be the last statement in a series of imperative statements. In a conditional statement, a GO TO must be followed by ELSE, IF, or a period.	The statements between the GO TO and the ELSE, IF, or period are deleted.
165	U	*VERB* STATEMENT OPERAND *DATA-NAME* NOT DEFINED IN LINKAGE SECTION.	An operand not defined in the Linkage Section has been encountered in an Entry or Procedure Division USING statement.	Data-names in an entry or Procedure Division USING statement must be defined in the Linkage Section.	The statement containing the listed operand is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
166	U	*VERB* STATEMENT OPERAND *DATA-NAME* IS NOT LEVEL NUMBER 01 OR 77.	A operand with a level number other than 01 or 77, has been detected in a CALL, ENTRY, or Procedure Division USING statement.	Data items in a CALL. ENTRY, or Procedure Division using statement are restricted to items whose level number is 01 or 77.	The verb is deleted from further compilation.
167	S	ADDITIONAL MEMORY REQUIRED TO PROCESS STATEMENT CONTAINING *DATA-NAME*.	This statement exceeds the internal storage area available to process statements with multiple operands.	The storage necessary to process a single operand varies from 18 to 250 bytes, depending on the number of characters in the data-name and whether the item OCCURS, has an edited picture, or is subscripted. The maximum storage available for statement processing is a function of the total storage available to the compiler. A limit of 100 symbols exists for a single condition. A symbol in this context is an operand, an arithmetic operator, a logical operator, a relational operator, or a class. (A condition-name test expands to multiple symbols depending on the number of values associated with the condition-name.)	The statement is deleted. Additional memory should be assigned to the compiler or the statement must be rewritten as multiple statements.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
168	U	*VERB* EXCEEDS LIMIT OF TEMPORARY DATA AREAS.	The maximum number of temporary arithmetic data areas has been exceeded.		Reduce the complexity of the expression or reduce the number of expressions in the statement.
169	U	*VERB* STATEMENT OPERAND *NAME* IS NOT RECORD OR FILE-NAME.	The input-output statement does not reference a record-name or file-name.	The following verbs must refer to record or file-names: OPEN, CLOSE, READ, WRITE, SORT, RELEASE, RETURN, INSERT, SEEK.	The statement in error is deleted.
170	U	SENTENCE PRODUCES EXCESSIVE OBJECT CODE	Object code cannot be produced for the entire sentence because of the sentence size.	Generally, a complete sentence is limited to between 2048 and 4096 bytes depending on the sentence structure.	Reduce the sentence size by rewriting it as several sentences/paragraphs.
173	U	*VERB* STATEMENT OPERAND *NAME* REFERS TO FILE RECORD AREA.	Both operands in the statement refer to the same storage area.	The operand specified in the WRITE FROM, INSERT FROM, or READ INTO options, may not occupy the same storage area as the record or file-name.	The statement is deleted.
174	U	*VERB* STATEMENT RECORD-NAME *NAME* IS NOT DEFINED IN FILE SECTION.	The listed operand is not defined in the File Section.	WRITE, INSERT, and RELEASE refer to items defined in the File Section.	The statement is deleted.
176	U	DIVIDE STATEMENT PRODUCES MEANINGLESS RESULT.	The description of the operands in a DIVIDE statement is such that only zeros could result for the quotient in the specified receiver.		The DIVIDE statement is deleted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
177	U	*VERB* STATEMENT CONFLICTS WITH SEGMENTATION RULES.	A branching verb is invalidly specified with respect to the rules of segmentation, or an ALTER statement refers to a paragraph that does not begin with a GO TO.	See the rules on segmentation for the listed verb.	The statement in error is deleted.
178	U	*VERB* STATEMENT INCOMPLETE OR CONTAINS INVALID OPERAND OR OPTION.	An operand conflicts with a specified option or with another operand, or an option that must be specified for a given statement was not encountered. For example, a WRITE to a mass storage device must contain an INVALID KEY clause.	See the rules for the specified verb.	The statement is deleted.
179	U	INTERNAL LABEL TABLE OVERFLOW.	Either a sentence requires more than 256 internal labels or more than 24 internal labels are active.		Requirements for internal labels may be lowered by reducing the number of statements in a sentence.
180	C	CLASS OF LITERAL CONFLICTS WITH CLASS OF *DATA-NAME*.	A nonnumeric literal containing numeric characters is being moved to an alphabetic item, or a nonnumeric literal containing nonnumeric characters is being moved to a numeric item.	The class of all characters contained in a nonnumeric literal must be consistent with the class of the receiving item.	The nonnumeric literal is accepted.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
181	P	*NAME/LITERAL* TRUNCATED DURING MOVE.	The data-name or literal being moved contains a greater number of character positions than the receiver or, when decimal point aligned, contains a greater number of digit positions than the receiver.	Truncation occurs when any portion of the item being moved cannot be contained in the receiving operand.	The data-name or literal is moved and truncated.
182	U	COMPLETE TRUNCATION OF *NAME/LITERAL/RESULT*.	Decimal point alignment is such that no portion of the item being moved can be contained in the receiving operand.		The MOVE statement or arithmetic giving MOVE is deleted.
183	U	REDUNDANT ROUND OPERAND *DATA-NAME*.	The numeric description of the arithmetic result is such that no excess digit positions are available for rounding into the listed operand.	Rounding is possible only when an arithmetic result contains at least one excess digit from which the round operation can be based.	The round operation is deleted.
184	P	REDUNDANT SIZE ERROR OPERAND *DATA-NAME*.	The numeric description of the arithmetic result is such that its value could never exceed the largest value that can be contained in the listed operand.	A size error is possible only if the arithmetic result contains more significant digit positions than the resultant identifier.	The size error test is performed.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
186	C	PERFORM STATEMENT LITERAL EXCEEDS 32,767.	The TIMES literal in the perform statement exceeds the maximum allowable value.	The maximum value of a PERFORM TIMES literal is 32,767.	The accepted TIMES count is the rightmost 15 bits of the original value when converted to binary. This value is between 1 and 32,767.
187	C	ADVANCING LITERAL EXCEEDS LIMIT.	The WRITE ADVANCING literal exceeds the maximum allowable value.	The maximum number of lines that can be advanced is 127 in the normal mode and 3 in the conversion mode.	The advancing line count is set to 1.
188	U	FILE AT LINE *LITERAL* NOT _____ WITHIN PROGRAM.	An OPEN or CLOSE has not been specified for the file or the OPEN is inconsistent with the activity associated with the file.	Every file must be opened and closed. Files written on must be opened for output or I-O, files read from must be opened for input or I-O.	Results during execution are unpredictable.

MESSAGE NUMBER	SEVERITY CODE	DIAGNOSTIC TEXT	EXPLANATION		
			REASON	RULE	RECOVERY
190	S	ADDITIONAL MEMORY REQUIRED TO PRODUCE OBJECT CODE LISTING.	The compiler does not have sufficient storage to produce the object code listing.	Recompilation is necessary with more storage assigned to the compiler.	The object module is produced.
191	S	ADDITIONAL MEMORY REQUIRED TO PRODUCE OBJECT PROGRAM.	The compiler does not have sufficient storage to maintain the compile time tables necessary to create the object module output for this program.		A recompilation is necessary with more storage assigned to the compiler.
192	C	KEY SIZES FOR FILE AT LINE *LITERAL* NOT *EQUAL*.	Record key size unequal to symbolic key size.	Record key and symbolic key sizes must be equal.	Symbolic key size is changed to record key size.

J.3. CONSOLE MESSAGES

During compilation, the compiler generates messages for general information purposes and messages which describe nonstandard compilation, such as a compiler abort.

During execution, COBOL-generated user programs may encounter one of several error conditions which are emphasized by console messages.

For description and explanation of COBOL generated console messages, see *UNIVAC 9400 Operations Handbook – UP-7871* (current version).

APPENDIX K. COMPILER LISTINGS

K.1. SOURCE CODE LISTING

A source code listing header line appears at the start of each source code listing. It identifies the compiler, the compiler version, the date of the compilation, and the time of day at which the COBOL program was compiled. If the date and time are to appear correctly in the source code listing header line, they must be set by the operator through the operator commands SET DATE and SET CLOCK when the supervisor is loaded.

At the top of each page of the source code listing there is a page header which identifies the page number and sections of the COBOL source code listing which appear under the page header. These sections are the line number, sequence number, source statement, and identification.

The line number (LINE NO.) is a compiler-generated number which identifies the particular line of COBOL source code with which it appears. The line number is used to reference lines of COBOL source code in the diagnostic listing, the object program listing, the Data Division memory map, the Procedure Division memory map, and the cross reference listing.

If the COPY verb is used, the letter C appears after the compiler-generated line number, to indicate lines of source code taken from the copy library.

The source item sequence number is listed under SEQ. (card columns 1 to 6). The sequence number field (card columns 1 to 6) is optional for the COBOL programmer.

Under SOURCE STATEMENT the text (card columns 7 to 72) of the COBOL source program is listed.

Under IDEN., program identification information (card columns 73 to 80) is listed. This is an optional entry made by the COBOL programmer to provide identification or card deck information. The COBOL compiler takes no action upon it.

K.2. DATA DIVISION STORAGE MAP AND CROSS REFERENCE LISTING

The storage map header line contains the PROGRAM-ID name, the compiler version, and the date and time of compilation.

The page header locates the following information:

LINE – The compiler-generated line number on which the data item is defined.

LEVEL – The level indicator or level number assigned to the item. An * indicates that the item was generated by the compiler, as with TALLY.

DATA-NAME – The name of the item.

REG – Where applicable, the hardware register number which contains the address used as a base value for referencing the item. If a permanent register has not been dedicated to cover the item, an * is listed.

DISP – The displacement of the item relative to the address contained in the item's cover register. The number is expressed in hexadecimal.

ADDR – The address of the item, relative to the first byte of the program. If blank, the address varies due to blocking, double buffering, etc. The number is expressed in hexadecimal.

LENGTH – The length in bytes of the item.

TYPE – The class or type of the item where:

GP = Group item
A/N = Alphanumeric item
A = Alphabetic item
NUP = Numeric unpacked item
IDN = index-data-name
IDX = index-name
AE = Alphabetic edited
NE = Numeric edited
NP = Numeric packed
VGP = Variable group item.

PTLOC – The decimal point location of the item where:

– integer indicates the number of fractional digit positions plus the number of leading P's in the PICTURE, i.e., –5 for PIC PP999 or PIC 9.99999 or PIC 99V99999.

+ integer indicates the number of trailing P's in the PICTURE, i.e., +5 for PIC 99P(5)

OCC – The number of occurrences of the item as specified by the OCCURS clause.

If the cross reference list has been specified, the line numbers where one or more references to the item were made are listed under LINE NUMBERS OF REFERENCES.

K.3. PROCEDURE DIVISION STORAGE MAP AND CROSS REFERENCE LISTING

The storage map header line will contain the PROGRAM-ID name, the compiler version, and the date and time of compilation.

The page header locates the following information:

LINE – The compiler-generated line number on which the item is defined.

SECTION – If the item is a section-name, it is listed here.

PARAGRAPH – If the item is not a section-name, it is listed here.

PRIORITY – The priority number of the section-name.

ADDR – The address of the procedure, relative to the first byte of the program. If the name is not referenced in the program, NO REF is listed here. The number is expressed in hexadecimal.

GO TO – An E indicates that the procedure is the object of a GO TO.

PERFORM

ENTRY – An E indicates that the procedure is the object of a PERFORM.

EXIT – An X indicates that the procedure contains a PERFORM exit point.

ALTER – An A indicates that the procedure is ALTERed.

SORT

ENTRY – An E indicates that the procedure is the entry point of a SORT procedure.

EXIT – An X indicates that the procedure contains a SORT procedure exit point.

DEBUG – An * indicates that the procedure is the object of a debug packet.

If the cross reference list has been specified, the line numbers where one or more references to the procedure have been made, are listed under LINE NUMBERS OF REFERENCES.

K.4. OBJECT CODE LISTING AND EXTERNAL REFERENCES

The object code listing header line contains the compiler version number and the date and time of the compilation.

Following the report header line is a list of external reference symbols (EXTRN and ENTRY names). These are the symbols whose object time address cannot be calculated at compile time and must be resolved by the linker. The program name and segment names are also listed here so that their object time address can be determined by the linker. A two-character ESID number follows each name. This number is used as a link between the ESID associated with all address constants and the element base to which that address is relative.

The following are compiler assigned names.

DP$00000
DP$00001
DP$00002
↓
ETC
} for DEPENDING Procedures

DEB$G00
DEB$G01
↓
ETC
} Debug Packets

The first entry in the list is the program name and its ESID number of 02. The program name is the PROGRAM-ID name. If the COBOL program is segmented, the segment names follow. The eight-character segment name is composed of the first six characters of the program-name and a two-character segment number. The segment number 01 will be assigned to the first section-name whose priority number exceeds 49; 02 to the next section with a different priority number greater than 49, etc. The ESID of the first segment is 03, the next is 04, etc.

The next group of names identify various external programs required in the execution of the COBOL program, such as the Data Management modules and special COBOL object time subroutines.

The last group of symbols are names that appear in CALL statements.

The object code listing page header identifies the following information:

LINE # – The compiler-generated line number on which each PROCEDURE DIVISION statement exists.

BASE/DISPL – This field lists the hardware base register number used to contain the cover address for the line of code. The displacement from the address in the cover register to this line of code is also displayed.

If this field is blank, no cover is needed for the line of code or the cover register assignment at object time varies and cannot be defined.

ADDRESS – The program-relative address where the line of code resides.

CONTENTS OF MEMORY – The actual hexadecimal description of the code or constants produced. An ESID number appears to the right of each address constant (DC A).

OPERAND ADDRESS – The program-relative address of the data or constant area being referenced. If this field is blank, the item is being addressed indirectly.

OPCODE – The mnemonic name for the constant or instruction produced on this line. If this field is blank and the 'contents of memory' field contains zeros, alignment is being effected for the next line of code.

COMMENTS – This field defines the purpose for which the code was generated. For code in the Procedure Division, the source program verb is listed.

Prior to the Procedure Division, the following numbers are used to locate the indicated items and areas.

1. Inter-segment GO TO Subroutine

   Used when control is passed from one segment of a segmented program to another.

2. Inter-segment PERFORM Subroutine

   Used when a PERFORM references a section or paragraph in another segment.

3. PERFORM EXIT Subroutine

   Called at end of paragraph or section referenced as PERFORM EXIT to determine if PERFORM is active or not.

4. CVB

   Converts packed decimal to binary.

5. CVD

   Converts binary to packed decimal.

6. Multiply Half-word Subroutine

   Determines product of two binary halfwords.

7. CVB and Multiply Half-word Subroutine

   Converts a packed decimal number to binary and multiplies it by another binary number.

8. GO TO DEPENDING Subroutine

   PERFORM function required by GO TO DEPENDING function.

9. Converts separate sign to embedded sign.

10. Converts embedded sign to separate sign.

11. Same as 10.

12. Calculate occurrence number.

13. Alter Fixed Segment Subroutine.

15. Transient Storage Area

    Storage area used to perform certain intermediate calculations.

16. Special Constants

    Constants required by verb generators.

17. Address of USING argument area

    Pointer to area used to pass USING arguments to CALLed routines; also used by ACCEPT and DISPLAY functions.

18. Address of USE procedure table

    Pointer to table of USE procedure addresses.

19. Address of ALTERed GO TO Table

    Pointer to table of ALTERed GO TO's in priority segments.

20. Start of BAT Table

    A table of addresses used to reference Data Division entries.

21. Start of PEP Table

    A table of addresses of referenced procedures.

22. Start of DTF Block Addresses

    A table of addresses which define the starting points of DTF's and the COBOL prefixes for each.

23. Start of EXTRNs for COBOL Subroutines

    EXTRNed address of subroutines required by certain COBOL functions.

24. VCON Reference Table

    A table of addresses created by CALL statements compiled as VCON's.

25. PERFORM EXIT Storage Area

    Area used to save address and other indicators for PERFORM functions.

26. Index-Name Storage Area

    Area used to store values of indexes, TALLY also in this area.

27. PERFORM N TIMES Counter Storage Area

28. Start of DTF Tables

    A series of tables used to define files for input/output functions.

29. Start of ALTERed GO TO Table

    A table of ALTERed GO TO's in priority segments.

30. Start of USE Procedure Table

    A table used to reference USE procedures containing necessary indicators and addresses.

31. Start of Data Division Initial Values

    Start of listing of constants produced by VALUE clause in Working-Storage Section.

32. Start of Procedure Division Constants

    Area contains those values and constants required by Procedure Division literals and functions.

APPENDIX L. USE OF ACCEPT AND DISPLAY STATEMENTS

L.1. GENERAL

When the mnemonic-name specified in the following format is not provided, the ACCEPT data will be from the job stream. A display upon the console is assumed if the UPON option is omitted.

ACCEPT identifier FROM mnemonic-name

L.2. ACCEPT FROM JOB STREAM

This form of ACCEPT is used to retrieve data images and certain control statements from the job control stream. COBOL programs are permitted to access their control streams in order to retrieve //bPARAM job control statements and data images. A maximum of 4095 bytes of data may be retrieved with a single ACCEPT. The number of bytes accepted is not required to be a multiple of 80. Two ACCEPTS of 20 character items require two cards.

Job Stream Set-up:

```
// EXEC operand 1, operand 2, operand 3, operand 4
```

The EXEC statement (execute) is the last statement processed by job control before the execution of the program (job step) named in the statement.

```
/$
```

The $ statement is used to indicate the beginning of a stream of data that is to be diverted to a file for subsequent retrieval by the job. All statements following the $ statement up to and including the first /* (end-of-data) statement are filed on the resident direct access storage device. Although this statement is required by job control, it will not be transferred to the COBOL program.

```
DATA IMAGE 1
DATA IMAGE 2
.          .
.          .
.          .
DATA IMAGE n
/*
```

The /* statement is used to indicate the end of a data stream introduced with the control stream. This statement is required by job control but will not be transferred to the COBOL program. An attempt to retrieve this statement will result in an error condition within the COBOL program.

Control Stream Errors:

When the control stream is unable to deliver an image to the COBOL program (that is, if the next sequential record in the control stream is not a //bPARAM job control state-statement, or a data image), control is transferred to the object time error subroutine. The subroutine logs the following message on the console:

CE01 ERROR-DATA FOR ACCEPT NOT AVAILABLE

If the COBOL program attempts to retrieve a /* image from the control stream, an error condition results. Control is transferred to the object time error subroutine. The subroutine logs the following message on the console:

CE02 ERROR-INSUFFICIENT DATA FOR ACCEPT

These errors cause the run to be aborted.

→ ACCEPTS from the job stream are not permitted inside a USE for LABEL PROCEDURE.

L.3. ACCEPT IDENTIFIER FROM SYSCONSOLE

The maximum number of characters that may be entered for a single ACCEPT is 4092. Due to system and hardware restrictions it is only possible to enter 60 characters of text per line.

The following will help to illustrate the mechanics for entering data via the console typewriter.

When the ACCEPT statement is encountered in the COBOL program, one of the following messages will be typed:

1. CA10 ACCEPT READY
2. CA10 ACCEPT READY – (The hyphen is typed whenever the maximum (60) number of characters for one line is to be entered, otherwise the hyphen is not printed. The message will be typed for each line to be entered.)

The operator, when replying to a console ACCEPT, must enter "job number R b" followed by the text.

Example of accepting into a USAGE DISPLAY data item:

1. *R14:37 18 CA10 ACCEPT READY
2. @14:37 18R TEST ACCEPT(EOM)

Line 1:

The * printed on line 1 indicates an OPR message with an operator reply expected.

The R printed indicates an operator action is required; namely a reply to the message presented. The following six characters indicate the time of day. The next three characters are job number; NNb job number 18 issued the ACCEPT. The next five characters are the message prefix identifying the message origin; this message is from COBOL (C) ACCEPT (A) and message type 10 followed by a blank. The remaining characters are the message text from ACCEPT.

Line 2:

In order to reply to the ACCEPT (OPR), the operator must press the ATTENTION KEY. This causes the @ to be printed on the console followed by six characters for the time of day. The operator then types JOB#R$\overline{\text{b}}$ (in the example 18R$\overline{\text{b}}$) followed by the message text to be accepted.

When the operator types less than the number of characters expected, the EOM (X'37') character is accepted into the area and the remaining positions are space-filled (X'40').

Information provided for a COMP item must be provided in DISPLAY format, with the correct number of digits and sign, as noted in the item description.

Example of accepting into a signed COMPUTATIONAL-3 data item (PIC S999):

Enter a computational constant +265.

1. *R14:38 18 CA10 ACCEPT READY
2. @14:38 18R 26E(EOM)

Enter a computational constant −349.

1. *R14:38 18 CA10 ACCEPT READY
2. @14:38 18R 34R(EOM)

Note that the last character entered must account for the sign of the last integer entered. (E=X'C5',R=X'D9') ACCEPTs from SYSCONSOLE are not permitted inside a USE for LABEL PROCEDURE.

L.4. ACCEPT IDENTIFIER FROM SYSDATE

ACCEPT FROM SYSDATE causes the date to be made available to the program in the format yymmdd (PIC 9(6)). This information is moved to the identifier under the rules for a COBOL MOVE.

When the date is set through the job stream (// SET DATE, MM/DD/YY) the date is stored in the user's job preamble. Although not a system convention, COBOL expects external SET DATE of mmddyy. If the date is not set via the job stream, Job Control will move the date from the system information block (SIB) into the user's job preamble. The date in the SIB is entered via the console by the operator. This is accomplished by using the operator SET command to enter the current data.

By setting the date from the job stream, the user may pre-date or post-date his jobs.

L.5. ACCEPT IDENTIFIER FROM SYSTIME

The diagnostic listing header line contains the program-ID name, the compiler version, and the date and time of compilation.

An explanation of the diagnostic text can be found in Appendix J.

ACCEPT FROM SYSTIME causes the time of day to be made available to the program in the format hhmm0000(PIC 9(8)), where hh is the hour and mm is the minute (hhmm does not exceed 2359). This information is moved to the identifier under the rules for a COBOL MOVE.

L.6. ACCEPT IDENTIFIER FROM SYSSWCH

ACCEPT FROM SYSSWCH permits the COBOL program to access the user program switch indicator (UPSI) byte which is the last byte of the 12-byte communication region in the job preamble. An eight-byte item is created containing EBCDIC 0 or EBCDIC 1 to represent the OFF or ON status of the individual UPSI bits/switches, respectively (i.e., if SYSSWCH-0 and SYSSWCH-2 are ON and all others are OFF, ACCEPT FROM SYSSWCH makes available to the program an 8-character item containing 10100000).

L.7. ACCEPT IDENTIFIER FROM SYSCOM

ACCEPT FROM SYSCOM allows the COBOL program to receive information from the communication region in the job preamble. When this ACCEPT is encountered, the 12-byte communication region is moved to the 12 bytes described by the identifier. It is through the communication region that one job step may communicate with a following job step.

NOTE: The twelfth byte of the communication region is the UPSI byte.

L.8. DISPLAY IDENTIFIER UPON SYSCONSOLE

DISPLAY UPON SYSCONSOLE permits the COBOL program to display messages upon the console typewriter. Message size is limited to 4092 contiguous characters.

Each line has the message prefix CD105. The message text is 59 characters per line. When the message text is larger than 59 characters, it is continued on the next line. For signed numeric items, a separate character is displayed immediately after the operand. If an EOM character (Ⓢ ,X'37') is encountered within the message text, it is printed and printing is terminated.

L.9. DISPLAY IDENTIFIER UPON SYSSWCH

DISPLAY UPON SYSSWCH permits the COBOL program to change the entire UPSI byte.

The eight bytes described by the identifier are converted into individual bit settings and resultant 8 bits are stored in the UPSI byte. A value of X'F1' causes a bit (UPSI switch) to be turned ON (1 value).

The UPSI byte may be initialized prior to execution by the SET statement in the Job Control stream (// SET UPSI, switch setting).

L.10. DISPLAY IDENTIFIER UPON SYSSWCH-n

DISPLAY UPON SYSSWCH-n allows the COBOL program to change an individual switch (bit setting) in UPSI. The eight switches in UPSI are numbered 0 through 7 from left to right. A one-byte identifier (PIC X) is used to alter UPSI SWITCH-n. A value of zero (X'F0') causes the switch to be turned OFF (0 value); any other value causes the switch to be turned ON (1 value).

L.11. DISPLAY IDENTIFIER UPON SYSCOM

DISPLAY UPON SYSCOM allows the COBOL program to alter the contents of the communication region. The 12 bytes described by the identifier are moved into the 12-byte communication region in the job preamble.

The communication region is initialized to binary zeros prior to the first job step by Job Control. Through use of the SET statement (// SET COMREG, character string), the communication region may be set to an initial value. Information may be passed from job step to job step in the communication region. The communication region is not changed during job steps.

L.12. DISPLAY IDENTIFIER UPON SYSLST

DISPLAY UPON SYSLST permits the COBOL programmer to display messages upon the printer. Displays are in 132-character multiples and are printed after advancing paper one line. For signed numeric items, a separate sign character is displayed immediately following the operand.

The LFD name assigned to the printer in the Job Control stream must be SYSLST.

At least one DISPLAY to SYSLST must be performed in the nondeclarative portion of the PROCEDURE DIVISION before any are performed within the DECLARATIVE portion.

APPENDIX M. DEBUGGING LANGUAGE

M.1. GENERAL

The source program debugging statements, READY TRACE, RESET TRACE, EXHIBIT, and *DEBUG are extensions to the American National Standard COBOL language.

The output resulting from the execution of a debugging statement is displayed upon the printer (LFD name = SYSLST). The output may be transferred to tape or disc by including the appropriate Job Control statement options and format information. Printing is performed after a one line paper advance. The print cooperative/symbiont mechanism permits subsequent listing.

The debugging statements may be included between Procedure Division statements, or the statements may be put in packet form at the end of the Procedure Division (see M.5).

M.2. READY TRACE

Function:

The execution of a READY TRACE statement produces the output:

'TRACE ON AT line-number'.

When a section or a paragraph is entered for execution, the following output is produced:

'section-name (or unqualified-paragraph-name) line-number'

Format:

READY TRACE.

Rule:

This statement may appear anywhere in the Procedure Division or in a compile-time debugging packet.

M.3. RESET TRACE

Function:

The execution of the RESET TRACE statement terminates the functions initiated by READY TRACE and produces the following output:

'TRACE OFF line-number'

Format:

RESET TRACE.

Rule:

This statement may appear anywhere in the Procedure Division or in a debugging packet.

M.4. EXHIBIT

Function:

The execution of the EXHIBIT statement results in a formatted display of identifiers or nonnumeric literals listed in the statement.

Format:

$$\text{EXHIBIT} \left\{ \begin{matrix} \text{NAMED} \\ \text{CHANGED} \\ \text{CHANGED NAMED} \end{matrix} \right\} \left\{ \begin{matrix} \textit{identifier-1} \\ \textit{nonnumeric literal-1} \end{matrix} \right\}$$

$$\left[, \left\{ \begin{matrix} \textit{identifier-n} \\ \textit{nonnumeric literal-n} \end{matrix} \right\} \ldots \right] .$$

Rules:

(1) An identifier may not be an index-data-item.

(2) An identifier length may not exceed 256 bytes.

(3) Nonnumeric literals may not exceed 132 characters in length.

(4) Displayed operands are continued as described by the DISPLAY statement. A maximum logical record size of 132 characters is assumed.

(5) An EXHIBIT statement may appear anywhere in the Procedure Division or in a debugging packet.

(6) The NAMED option produces a noncolumnar display of all operands specified in the EXHIBIT statement. The operands are displayed in source order and are formatted as follows:

(a) Identifier

identifying name␣equal sign␣identifiers value␣

The *identifying name* includes qualifiers and subscripts. A maximum of 130 characters will be displayed.

The *identifiers value* may be a maximum of 256 characters. If the identifier is a signed numeric elementary item, a sign is also displayed following the value.

(b) Nonnumeric literal

nonnumeric literal␣

(7) The CHANGED NAMED option produces a noncolumnar display of all nonnumeric literals and, conditionally, the identifiers specified in the EXHIBIT statement. The format sequence of the displayed operands is as described in Rule (6). If the value of the identifier has not changed since the previous execution of this EXHIBIT statement, the identifier is not displayed and space is not reserved for the value in the print record.

All identifier values are considered changed on the initial execution of the statement. If the EXHIBIT statement does not contain any nonnumeric literals and the value of all identifiers is the same as when this EXHIBIT was previously executed, neither a display nor a form advance occurs.

(8) The CHANGED option produces a columnar display of all nonnumeric literals and the changed values of all identifiers.

If the value of the identifier has not changed since the previous execution of this EXHIBIT statement, the positions reserved for the identifier value are displayed containing spaces. All identifier values are considered changed on the initial execution of the EXHIBIT statement.

When the statement contains only identifiers and none of the values have changed, one line of spaces is displayed. The operands are displayed in the order in which they appear in the statement and have the following format:

(a) Identifier

identifier value␣

The identifier value may be a maximum of 256 characters. If the identifier is a signed numeric elementary item, its sign is displayed following the value.

(b) Nonnumeric literal

non-numeric literal␣

(9) If two distinct EXHIBIT CHANGED NAMED or two EXHIBIT CHANGED statements appear in one program, each specifying the same *identifiers*, the changes in value of those identifiers are associated with each of the two separate statements. Depending on the path of program flow, the values of the identifier saved for comparison may differ for each of the two statements.

(10) Variable length identifiers are not permitted as operands with the CHANGED or CHANGED NAMED option.

M.5. THE DEBUGGING PACKET

A packet contains debugging statements referring to a paragraph name or a section name in the Procedure Division. The debug packets are grouped together and placed immediately following the source program. The packet statements are compiled with the source program and are executed at object time; the packets produce the same result as placing the debug statements directly in the source program following a section name or a paragraph name.

Each debug packet is preceded by a control card which has the following format:

```
1       8
*DEBUG location
```

Location refers to a section or paragraph name which starts anywhere within margin A. The name, which may be qualified, indicates the starting point in the program where execution of the packet is to begin. *Location* cannot be a paragraph name within any debug packet and the same location must not be used in more than one debug control card.

A debug packet may consist of procedural statements such as GO TO, PERFORM, or ALTER. These statements may refer to a procedure name in any debug packet or in the main body of the Procedure Division.

When the source COBOL program is on a library file (tape or disc), the library module containing the source program may also contain *DEBUG control cards. Regardless of whether the library module contains any *DEBUG cards, when the compiler reaches the end of the library module, it will determine if any additional *DEBUG cards are present in the job stream. If there are *DEBUG cards in the job stream, they are processed as if they were contained at the end of the library module. If there are no *DEBUG cards present in the job stream, the process of reading COBOL input to the compiler is terminated.

Example:

```
// EXEC COBOL,LOAD$LIB,,REL
// PARAM IN = PROGNAME/LIBIN
// PARAM LST = (O,C,S)
/$
*DEBUG_____________________
_____________________
_____________________
*DEBUG_____________________
_____________________
_____________________
/*
.
.
.
```

APPENDIX N. ASCII PROCESSING

N.1. GENERAL

When the PARAM statement is specified in the job control stream, the COBOL compiler produces an object program which assumes that the contents of any data item, with an implied or explicit USAGE IS DISPLAY, is represented in American Standard Code for Information Interchange character set (ASCII).

The format of the PARAM statement is:

```
// PARAM OUT=A
```

Although an EBCDIC or ASCII object program may be generated by the compiler, compilation is always performed in EBCDIC mode, as is the source program input and all listable output. The following discussion applies to programs produced under control of the ASCII option.

When the ASCII mode is selected, the computer character set at execution time consists of the 128 ASCII character codes (normally, the computer character set consists of the full 256 EBCDIC characters). Compiler-created object programs are sensitive at the hardware instruction level to the ASCII character set.

N.2. FIGURATIVE CONSTANTS

The value associated with the figurative constant HIGH-VALUE is hexadecimal 7F. QUOTE, ZERO, and SPACE take on the corresponding ASCII value. LOW-VALUE remains a hexadecimal 00.

N.3. COMPUTATIONAL ITEMS

Values associated with WORKING-STORAGE items whose USAGE IS DISPLAY are allocated to the program in the appropriate ASCII character codes. The allocation of values for computational forms of data is not effected by the interchange mode selection. These values are generated in the corresponding computational format. When computational items are moved or compared against DISPLAY items, they are automatically converted to the ASCII character code values.

N.4. SIGNED ITEMS

The ASCII character set does not facilitate the use of the traditional overpunch sign convention; however, 9400 implementation of ASCII does permit overpunching. See 5.3.13. An ASCII signed numeric DISPLAY item should have an S in its PICTURE and be associated with a SIGN IS SEPARATE CHARACTER clause. Signed numeric items used in conjunction with arithmetic operations are automatically converted to the appropriate signed computational format.

Signed numeric values of PROCEDURE DIVISION constants must be specified with the sign as the leftmost character regardless of the SEPARATE SIGN option.

N.5. APPLY ASCII

ASCII files must be declared to the compiler by the APPLY ASCII ON file-name clause. ASCII files must also be declared to the supervisor by the ASCII parameter on the LFD job control statement. Any file may be classified as an ASCII file except those assigned to DISC. A mix of ASCII and non-ASCII files is permitted in the COBOL program.

N.6. RECORDING MODE

Regardless of the mode specified for files assigned to TAPE, user labels and data records are presented to the program in their external character code representation. No translation of input or output records is performed by the system. The RECORDING MODE IS D clause may be specified for ASCII tape files which contain variable length records.

An option within the APPLY ASCII ON file-name clause allows the specification of a buffer offset for any tape input file or the activation of the block length check feature on tape files with RECORDING MODE D.

If a file assigned to the printer is declared as being an ASCII file, the device will accept only ASCII records. Multiple print files, assigned to the same device, with different mode specifications are not permitted.

Depending on the mode specified for a file assigned to a CARD-READER, the system will present input records to the program in either ASCII or EBCDIC. Multiple card reader files assigned to the same device with different mode specifications are not permitted.

Depending on the mode specified for a file assigned to a PUNCH, the system will accept output records from the program in either ASCII or EBCDIC. Multiple card punch files assigned to the same device with different mode specifications are not permitted.

N.7. CONSTANTS AND LITERALS

PROCEDURE DIVISION numeric literals and nonnumeric literals used in conjunction with items whose USAGE IS DISPLAY are allocated within the object program as ASCII constants. Literals associated with computational data items are allocated in the corresponding computational format.

N.8. CONDITIONAL TEST

The results of an IF statement associated with an item whose USAGE IS DISPLAY is based upon the assumption that the item is represented in the ASCII character set.

N.9. DISPLAY

DISPLAYs upon SYSLST and output from TRACE/EXHIBIT statements require that ASCII be specified on the LFD control card for SYSLST. If, at the time this output is generated, a user print file assigned to the same device is OPEN but was not declared as an ASCII file, the device will not accept the debug output records.

N.10. ACCEPT

Since the console assumes the mode of the problem program, only ASCII data should be DISPLAYed upon SYSCONSOLE. When ACCEPTing from SYSCONSOLE, the user will receive ASCII data.

Data ACCEPTed from the job stream will also be in ASCII. This includes // PARAM cards.

N.11. SORT

Sorting is controlled entirely by the SORT KEY specifications and the values contained within the KEY fields. SORT keys whose category is other than numeric are processed according to their hexadecimal value. Numeric keys are treated algebraicly regardless of the item's operational sign location.

N.12. CONVERSION OF DISPLAY ITEMS

The TRANSFORM statement may be used to convert DISPLAY items from one character representation to another.

A special format of the TRANSFORM statement will facilitate translation from ASCII to EBCDIC or from EBCDIC to ASCII via compiler created translate tables. This option is specified when the reserved words ASCII or EBCDIC appear following the word FROM or TO. If TRANSFORM rules are specified by nonnumeric literals, transformation across code bases is not possible.

N.13. COBOL SOURCE LANGUAGE ADDITIONS

Format:

```
APPLY ASCII [WITH BUFFER-OFFSET {FOR BLOCK-LENGTH-CHECK}]
                                {OF integer CHARACTERS }
ON file-name [, file-name]. . . .
```

Rules:

1. The APPLY ASCII clause identifies each file that contains or receives ASCII data (see 4.3.2).

2. The BUFFER-OFFSET option applies only to files that are assigned to TAPE. The absence of this option implies the absence of a buffer offset.

3. The integer CHARACTERS option specifies the number of additional characters that appear at the front of each data block in the file. Integer may have a value of 0 to 99. The specified offset applies only to files OPEN for INPUT. The offset area cannot be referenced by the program nor can it be created when the file is OPEN for OUTPUT.

4. The BLOCK-LENGTH-CHECK option applies only to tape files whose RECORDING MODE IS D. When specified, input data blocks are assumed to possess a four character buffer offset which contains the length of the block. Data Management routines validate that each block read contains the number of characters specified in this field. When the file is being created, the block length is placed in the four-character buffer offset area.

N.14. THE RECORDING MODE CLAUSE

Format:

$$\underline{\text{RECORDING}}\ \text{MODE IS}\left\{\begin{matrix}\underline{U}\\ \underline{V}\\ \underline{F}\\ \underline{D}\end{matrix}\right\}$$

Rules:

1. The RECORDING MODE clause is expanded to include the specification of 'D' type records (see 5.2.1.4)

2. A recording mode of 'D' may be specified for ASCII tape files with variable-length records.

3. Tape files declared as ASCII may also have a recording mode of 'V' since, for ASCII files, 'D' and 'V' are synonymous. The 'D' mode is for compatability with other implementors.

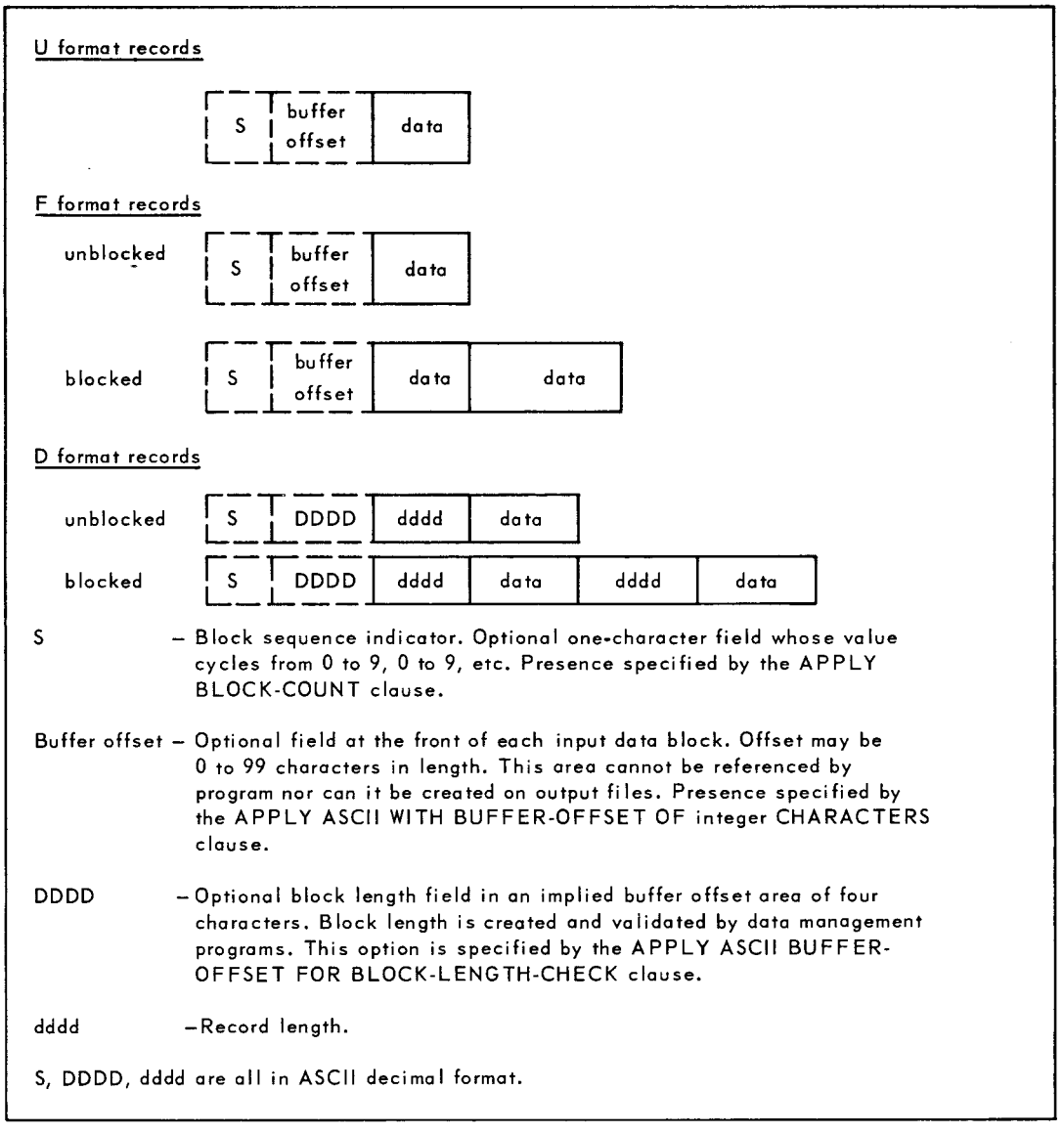


Figure N–1. ASCII Physical Tape Formats

RECORDING MODE IS	FILE DECLARED AS	LABEL RECORDS SPECIFICATIONS	APPLY BUFFER-OFFSET		APPLY BLOCK-LENGTH-CHECK	APPLY BLOCK-COUNT
			INPUT	OUTPUT		
U	EBCDIC	OMITTED STANDARD ① data-name ③				optional
	ASCII	OMITTED STANDARD ② data-name ③	0 to 99			optional ⑤
F blocked or unblocked	EBCDIC	OMITTED STANDARD ① data-name ③				optional
	ASCII	OMITTED STANDARD ② data-name ③	0 to 99			optional ⑤
V blocked or unblocked	EBCDIC	OMITTED STANDARD ① data-name ③			automatic	optional
	ASCII					
D blocked or unblocked	EBCDIC					
	ASCII	OMITTED STANDARD ② data-name ③	0 to 99 ④		optional ④	optional ⑤

NOTES:

① De facto standard as defined by UNIVAC 9400 System Data Management System Programmer Reference, UP-7629 (current version)

② ANSI standard.

③ Implies presence of system standard labels 1 or 2.

④ BLOCK-LENGTH-CHECK specifies that a buffer offset of four characters contains the length of the block for verification by data management programs.

⑤ Specifies a one-character cyclic block sequence indicator.

Table N-1. Characteristics of Tape Files Available to COBOL Users

ASCII		CONTROL CHARACTER	SYMBOL	EBCDIC		SIGNED NUMBER
HEX	DEC			HEX	DEC	
00	0	NUL		00	0	
01	1	SOH		01	1	
02	2	STX		02	2	
03	3	ETX		03	3	
04	4	EOT		37	55	
05	5	ENQ		2D	45	
06	6	ACK		2E	46	
07	7	BEL		2F	47	
08	8	BS		16	22	
09	9	HT		05	05	
0A	10	LF		25	37	
0B	11	VT		0B	11	
0C	12	FF		0C	12	
0D	13	CR		0D	13	
0E	14	SO		0E	14	
0F	15	SI		0F	15	
10	16	DLE		10	16	
11	17	DC1		11	17	
12	18	DC2		12	18	
13	19	DC3		13	19	
14	20	DC4		3C	60	
15	21	NAK		3D	61	
16	22	SYN		32	50	
17	23	ETB		26	38	
18	24	CAN		18	24	
19	25	EM		19	25	
1A	26	SUB		3F	63	
1B	27	ESC		27	39	
1C	28	FS		1C	28	
1D	29	GS		1D	29	
1E	30	RS		1E	30	
1F	31	US		1F	31	
20	32	SP, SPACE		40	64	
21	33		!	4F	79	
22	34		"	7F	127	
23	35		#	7B	123	
24	36		$	5B	91	
25	37		%	6C	108	
26	38		&	50	80	
27	39		'	7D	125	
28	40		(	4D	77	
29	41		)	5D	93	
2A	42		*	5C	92	
2B	43		+	4E	78	
2C	44		,	6B	107	
2D	45		–	60	96	
2E	46		.	4B	75	
2F	47		/	61	97	
30	48		0	F0	240	

Table N–2. ASCII/EBCDIC Conversion (Part 1 of 3)

ASCII		CONTROL CHARACTER	SYMBOL	EBCDIC		SIGNED NUMBER
HEX	DEC			HEX	DEC	
31	49		1	F1	241	
32	50		2	F2	242	
33	51		3	F3	243	
34	52		4	F4	244	
35	53		5	F5	245	
36	54		6	F6	246	
37	55		7	F7	247	
38	56		8	F8	248	
39	57		9	F9	249	
3A	58		:	7A	122	
3B	59		;	5E	94	
3C	60		<	4C	76	
3D	61		=	7E	126	
3E	62		>	6E	110	
3F	63		?	6F	111	
40	64		@	7C	124	
41	65		A	C1	193	+1
42	66		B	C2	194	+2
43	67		C	C3	195	+3
44	68		D	C4	196	+4
45	69		E	C5	197	+5
46	70		F	C6	198	+6
47	71		G	C7	199	+7
48	72		H	C8	200	+8
49	73		I	C9	201	+9
4A	74		J	D1	209	−1
4B	75		K	D2	210	−2
4C	76		L	D3	211	−3
4D	77		M	D4	212	−4
4E	78		N	D5	213	−5
4F	79		O	D6	214	−6
50	80		P	D7	215	−7
51	81		Q	D8	216	−8
52	82		R	D9	217	−9
53	83		S	E2	226	
54	84		T	E3	227	
55	85		U	E4	228	
56	86		V	E5	229	
57	87		W	E6	230	
58	88		X	E7	231	
59	89		Y	E8	232	
5A	90		Z	E9	233	
5B	91		[	4A	74	
5C	92		\	E0	224	
5D	93		]	5A	90	
5E	94			5F	95	
5F	95		—	6D	109	
60	96		`	79	121	
61	97		a	81	129	
62	98		b	82	130	
63	99		c	83	131	

Table N–2. ASCII/EBCDIC Conversion (Part 2 of 3)

ASCII		CONTROL CHARACTER	SYMBOL	EBCDIC		SIGNED NUMBER
HEX	DEC			HEX	DEC	
64	100		d	84	132	
65	101		e	85	133	
66	102		f	86	134	
67	103		g	87	135	
68	104		h	88	136	
69	105		i	89	137	
6A	106		j	91	145	
6B	107		k	92	146	
6C	108		l	93	147	
6D	109		m	94	148	
6E	110		n	95	149	
6F	111		o	96	150	
70	112		p	97	151	
71	113		q	98	152	
72	114		r	99	153	
73	115		s	A2	162	
74	116		t	A3	163	
75	117		u	A4	164	
76	118		v	A5	165	
77	119		w	A6	166	
78	120		x	A7	167	
79	121		y	A8	168	
7A	122		z	A9	169	
7B	123		{	C0	192	
7C	124		¦	6A	106	
7D	125		}	D0	208	
7E	126		~	A1	161	
7F	127	DEL		07	07	
80	128	ISR		20*	32	
81	129	SSB		21*	33	
82	130	FSB		22*	34	

*For edit mask conversion only.

Table N–2. ASCII/EBCDIC Conversion (Part 3 of 3)

APPENDIX O. RERUN CLAUSE

O.1. GENERAL

The RERUN facility provides a means of recording the status and environment of a COBOL program at a specified point in the processing of that program. Once recorded, this status and environment may be re-established and execution of the COBOL program resumed from this point. The RERUN facility causes linkage between the COBOL program and the 9400 checkpoint facility. The RESTART ability is provided by the RESTART job control statement.

O.2. RERUN CLAUSE

The RERUN clause may appear in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION. The format of the rerun clause is:

RERUN ON *external-name* EVERY *integer* RECORDS OF *file-name-1* [, *file-name-2*,...]

The external-name in the format above must appear in a SELECT statement. The device specified by external-name is the RERUN receiver; this device receives the checkpoint records which contain the status and environment of the COBOL program. File-name-1, file-name-2, etc., are RERUN controllers; these files dictate when the check point records are to be issued. The same rerun receiver may appear in any number of RERUN clauses; a RERUN controller may appear in only one RERUN clause. The allowable range for integer is 1 to 9,999,999.

O.3. CHECKPOINTING

Checkpoint records are issued whenever integer records occur for a rerun controller. The rerun controller's record counter is set to 0 when the controller is opened and incremented by 1 before each READ, WRITE, or INSERT issued to the controller. When the rerun controller is opened as I-O, a WRITE does not cause the record counter to be incremented.

If the rerun receiver is a tape device, it may be dedicated to receiving checkpoint records or it may receive other program output. If the rerun receiver is dedicated, it will be OPENed automatically with the assumption that label records are omitted. If the rerun receiver is being shared with other program output, it is the programmer's responsibility to ensure that the receiver is OPENed for OUTPUT whenever checkpoint records are issued. Checkpoint records will not be issued if the receiver is not OPEN for OUTPUT.

If the receiver is a disc device, it must be dedicated to receiving checkpoint records. The device must appear in a SELECT statement but not in an FD.

If the RERUN facility is to be used in a single, linked object program which is composed of multiple COBOL object programs, a RERUN clause must appear in each COBOL module which performs I/O. When a RERUN controller is opened, the files of that module are added to the list of files for the entire program. Therefore, in each COBOL object module, a RERUN controller should be opened before any other I/O is done in that module. Otherwise, it is possible that checkpoint records will be issued which do not contain the status of all the active files in the entire program.

O.4. RESTARTING

To initiate the restart of a previously checkpointed program, the original control stream, with the addition of a // RSTRT control statement immediately following the //JOB control statement, should be entered. The format of the // RSTRT statement is specified in *UNIVAC 9400 System Job Control Programmer Reference, UP-7793* (current version).

O.5. NOTES AND RESTRICTIONS

Due to the manner in which the sort facility's temporary scratch files are processed, checkpoints which reflect usable restart information cannot be issued. Therefore, even though integer records may occur for a rerun controller, checkpoints will not be issued if a sort is active.

The restart facility will verify and reposition the programs files; however, the programmer must ensure that the data on those files has not changed since the time of the checkpoint, or that any change will not affect the execution of the restarted program. For example, a disc file is being updated and a checkpoint is issued just prior to reading record number 400. The program is aborted for external reasons after record number 500 is updated and rewritten. The restarted program will begin updating at record number 400; however, records number 400 through 500 have already been updated.

The restart facility will reposition the program's control stream to the card following the last card ACCEPTed before the checkpoint was taken. This is done by searching the control stream for a card match. If duplicate cards exist in the control stream, it is the responsibility of the programmer to ensure that the first card match is the proper one.

APPENDIX P. CONVERSION MODE

P.1. GENERAL

To facilitate the conversion of a nonstandard COBOL to UNIVAC 9400 COBOL, a conversion facility has been built into the UNIVAC 9400 COBOL compiler. This facility, called the conversion mode (or C-mode), accepts COBOL source code and alters it to American National Standards Institute (ANSI) specifications, or flags it so the programmer is made aware of the need for changes.

The source code that has been chosen as input to the conversion mode is IBM System/360 DOS COBOL level D (COBOL-D) because it is representative of high level nonstandard COBOL.

P.2. CONVERSION MODE OPERATION

A PARAM card option is available to energize the conversion mode of the UNIVAC 9400 COBOL complier.

The conversion mode availability does not imply total source program transfer capability. Its intent is to minimize the volume and complexity of source program alterations necessary to successfully compile a given COBOL-D program. Every attempt is made to provide software support for those language differences that would, under a totally manual conversion process, require a knowledge of the source program intent and logic flow. Source program statements that must be altered prior to compilation are, in most cases, independent of program design.

There are several methods by which the conversion mode operates on a COBOL-D source program. In addition to accepting portions of the alien syntax and interpreting that syntax in a COBOL-D manner, the complier alters the meaning of certain source clauses and statements. For example, a COBOL-D COMP clause is treated as a UNIVAC 9400 COMP-4 clause.

In the conversion mode, various compiler processing paths are altered to effect a change in the semantic interpretation of a COBOL-D clause or statement, as in the case of contradiction across compilers associated with the IF NUMERIC statement.

Occasionally, an entire processing philosophy can be reversed. In the conversion mode, the compiler assumes that ASCII print control characters are utilized in all print files. In addition, a special COBOL-supplied object time subroutine is provided to ensure acceptable object program print speed. This software bridges the gap between the exclusive use in COBOL-D programs of the WRITE AFTER ADVANCING statement and the associated UNIVAC 9400 System hardware limitation.

This appendix describes the known differences that exist between COBOL-D and UNIVAC 9400 ANS COBOL. It also defines the language differences that the conversion mode renders transparent. Those language differences for which no automatic software support is possible are also identified here, along with the appropriate source program change requirement.

When functioning in the conversion mode, many of the compiler's ANS language features are disabled. Therefore, it is not recommended that a COBOL-D program once converted, be modified to take advantage of the many additional UNIVAC 9400 language capabilities without first being totally converted to ANS COBOL.

In the normal ANS COBOL mode, COBOL-D language differences are not permitted. The special processing interpretations and software extensions available in the conversion mode are not supported in the ANS mode; that is, control of character print files are unique to the conversion mode.

NOTE:

The conversion mode and the ASCII mode of the compiler are mutually exclusive. This restriction is imposed because of the incompatibility between the IBM and UNIVAC ASCII implementation philosophy. UNIVAC supports internal ASCII processing, while IBM effects data translations at the extremities. This is remedied by use of the TRANSFORM verb in conjunction with input/output processing.

P.3. CONVERSION MODE SYNTAX

The differences between COBOL-D and UNIVAC 9400 COBOL are described in the following paragraphs within each program division.

P.3.1. Identification Division

- PROGRAM-ID. program-name.

  COBOL-D

  Program-name is one to eight characters enclosed in quotation marks.

  9400

  Program-name is one to thirty characters not enclosed in quotation marks. Only the first six characters, excluding hyphens, are used to identify the object program.

  C-mode

  9400 will accept a 1- to 8-character name enclosed in quotation marks. Only the first six characters, excluding hyphens, are used to identify the object program.

P.3.2. Environment Division

- CONFIGURATION SECTION heading.

  COBOL-D

  Optional.

9400

Required.

C-mode

Optional.

- SOURCE/OBJECT COMPUTER clause.

COBOL-D

IBM-360 model-number.

9400

UNIVAC-9400.

C-mode

The compiler accepts any SOURCE/OBJECT COMPUTER entries that are valid for COBOL-D.

- SPECIAL-NAMES paragraph/DECIMAL-POINT IS COMMA clause

COBOL-D

Does not exist. Reversal of decimal point and comma is activated by a parameter on the CBL control card.

9400

Reversal of decimal point and comma controlled by SPECIAL-NAMES entry.

C-mode

No automatic support. The convertor has to insert a special-names paragraph and the DECIMAL-POINT IS COMMA clause into the source program before compiling.

If the CONSOLE or SYSLST option of an ACCEPT/DISPLAY statement is used in the program, the compiler will automatically produce a special-names entry, internally, for the program. CONSOLE will be equated to SYSCONSOLE, and SYSLST will be equated to SYSLST.

- SELECT/ASSIGN clause

COBOL-D

ASSIGN TO 'external-name' { DIRECT-ACCESS / UTILITY / UNIT-RECORD }

device-number UNIT(s)

9400

ASSIGN TO 'external-name' integer implementor-name

C-mode

No automatic support. The SELECT statement, with respective ASSIGN clauses, must be replaced by the appropriate UNIVAC 9400 SELECT/ASSIGN clauses before compilation.

- ACCESS clause

COBOL-D

The word 'IS' is optional.

9400

The word 'IS' is required.

C-mode

The word 'IS' is optional.

- KEY clauses

COBOL-D

The word 'IS' is optional.

9400

The word 'IS' is required.

C-mode

The word 'IS' is optional.

- I-O-CONTROL paragraph entries

COBOL-D

Allows the clauses of the I-O-CONTROL paragraph to be separated by periods.

9400

Allows the clauses to be separated by a comma or a semicolon. A period must follow the last entry in the paragraph.

C-mode

No automatic support. The embedded periods within the I-O-CONTROL paragraph must be removed

- RERUN clause

COBOL-D

RERUN ON 'external-name' { DIRECT ACCESS / UTILITY } device-number

UNIT(s) EVERY integer RECORDS OF file-name.

External-name may not be the same as the external-name in an ASSIGN clause.

Allows a maximum of 20 external devices to be used to store checkpoint records — only one of these can be a direct access device.

Checkpoint records are written preceding the execution of integer for a READ, WRITE, or REWRITE statement. Integer may not exceed 8,388,607.

9400

RERUN ON 'external-name' EVERY integer RECORDS OF file-name

The external-name must be specified in an ASSIGN clause.

The only restriction on the devices is the compiler limit of 63 devices per program.

Integer may not exceed 9,999,999.

C-mode

No automatic support. The RERUN clause must be replaced by one that conforms to the OS/7 COBOL format. A SELECT statement must be added for each external-name in each RERUN clause.

- APPLY clause for FORM-OVERFLOW

COBOL-D

APPLY overflow-name TO FORM-OVERFLOW ON file-name.

9400

This clause is not supported.

C-mode

No automatic support. Remove the APPLY FORM-OVERFLOW clause from the source program. Add a USE FOR FORM-OVERFLOW procedure in the Declaratives portion of the Procedure Division for detection of page breaks.

- APPLY clause for RESTRICTED SEARCH

  COBOL-D

  The word 'ON' is optional.

  9400

  The word 'ON' is required.

  C-mode

  The word 'ON' is optional.

- COPY library-name

  COBOL-D

  Library names are enclosed in quotation marks.

  9400

  Library names are not enclosed in quotation marks.

  C-mode

  Library names are enclosed in quotation marks. All libraries are expected to be in UNIVAC 9700 format.

P.3.3. Data Division

- Data formats

  COBOL-D

  COMPUTATIONAL-1 specifies short floating-point format; COMPUTATIONAL-2 specifies long floating-point format.

  9400

  COMPUTATIONAL-1 and 2 are not supported.

  C-mode

  COMP-1 and 2 are not supported.

- LABEL RECORDS clause

  COBOL-D

  Optional clause. If omitted, LABEL RECORDS OMITTED is assumed. For LABEL RECORDS ARE data-name, the data names must be 01 or 77 in the Linkage Section.

  9400

  Required clause. If the clause is omitted, a diagnostic is produced and OMITTED is assumed (unless device is disc, then labels are assumed to be STANDARD). For LABEL RECORDS ARE data-name, the data-name record description must be subordinate to the file description.

  C-mode

  Optional clause. Same default as COBOL-D. Label data-names must be in Linkage Section as 01 or 77 level items.

- PICTURE clause

  COBOL-D

  An external floating-point item may be defined by a PICTURE which contains an 'E' and two sign characters. The sterling currency feature may be specified by extensions to the PICTURE clause.

  9400

  Neither the sterling currency feature nor the external floating-point PICTURE description are supported.

  C-mode

  Neither the sterling currency feature nor the external floating-point PICTURE description are supported.

- USAGE clause

  COBOL-D

  The USAGE IS COMPUTATIONAL clause indicates that the data is in binary format.

  If USAGE IS COMP, COMP-1, or COMP-2, intra-record slack bytes are added by the compiler to ensure that the data is aligned on a half-word, full-word, or double-word boundary.

  9400

  USAGE IS COMPUTATIONAL indicates that the data is in packed decimal format. Binary data formats are not supported. COMP-1 and 2 are not supported.

  C-mode

  There is no support for binary or floating-point data formats.

- Working-Storage Section

COBOL

All 01's are aligned on a double-word boundary.

9400

Level 01's in Working-Storage Section are not aligned.

C-mode

All level 01's in Working-Storage Section are not aligned on a double-word boundary.

- COPY specifications

COBOL-D

The COPY statement is allowed on 77 items in the Working-Storage and Linkage Sections.

9400

The COPY statement is not allowed on 77 items.

C-mode

The COPY statement is allowed on level-number 77 items in the Working-Storage and Linkage Sections. However, the implied replacing feature is not supported. Replacing can be accomplished by use of explicit REPLACING clauses. All COPY libraries are expected to conform to UNIVAC 9400 formats.

P.3.4. Procedure Division

- ACCEPT statement

COBOL-D

A maximum of 72 characters may be accepted from the console.

When the FROM option is not used, one logical record will be retrieved from the system logical input device (SYSIPT).

Since a special-names paragraph is not available, the only acceptable FROM option is CONSOLE.

If /* is encountered on an ACCEPT statement, a fall through to the next source statement is effected. End-of-file detection is the user's responsibility.

9400

A maximum of 4095 characters may be accepted from the console.

When the FROM option is not used, a maximum of 4095 characters (52 card images) are retrieved from the job stream.

If /* is encountered on an ACCEPT statement, an object-time diagnostic is issued and the program is terminated.

C-mode

SYSIPT is equivalent to the UNIVAC 9400 job stream file.

The compiler creates an internal special-name definition to equate CONSOLE to SYSCONSOLE.

- DISPLAY statement

COBOL-D

When the UPON option is omitted, SYSLST is assumed.

Displays may be directed to SYSPUNCH.

The sign of a numeric item is not displayed as a separate character, i.e., -32 displayed as 3K.

9400

When the UPON option is omitted, SYSCONSOLE is assumed.

Displays to a punch are not supported.

The sign of a numeric item is displayed as a separate character, i.e., -32 displayed as 32-.

C-mode

When the UPON option is omitted, SYSLST is assumed. The compiler creates an internal special-name definition to equate SYSLST to SYSLST.

Restriction: Displays to a punch are not supported. The sign of a numeric item is displayed as a separate character.

- IF statement

COBOL-D

A class test may be performed on an item whose usage is either DISPLAY or COMP-3 (packed decimal).

An IF NUMERIC test always assumes the item is signed, for example:

```
DATA-AA PIC X VALUE IS 'A'.
```

An IF NUMERIC test on DATA-AA yields a 'yes'.

9400

A class test may be performed only on an item whose USAGE IS DISPLAY, but not floating-point display.

An IF NUMERIC test does not assume an item is signed. The sign is interrogated only if the item is declared to be signed; for example:

```
DATA-AA PIC X VALUE IS 'A'.
```

An IF DATA-AA NUMERIC results in a 'no'.

C-mode

A class test may be performed on an item whose USAGE IS COMP-3 (packed decimal) or floating-point display.

An IF NUMERIC test always assumes the item is signed.

- INCLUDE (COPY) statement

COBOL-D

An INCLUDE statement in the Procedure Division implies a COPY function.

9400

The INCLUDE statement is not supported. The COPY verb must be used.

C-mode

The INCLUDE statement is equated to the COPY function. Library names enclosed in quotation marks are accepted. COPY libraries are expected to be in UNIVAC 9400 format.

- MOVE statement

COBOL-D

When an unsigned numeric item is moved to a signed numeric item, the sign of the receiver is set to 'F'.

9400

When an unsigned numeric item is moved to a signed numeric item, the sign of the receiver is set to plus.

C-mode

When an unsigned numeric item is moved to a signed numeric item, the sign of the receiver is set to 'F'.

- ON statement

  COBOL-D

  This statement is supported.

  9400

  This debugging statement is not supported.

  C-mode

  No automatic support. This clause is not supported.

- READ statement

  See P.5 for disc considerations.

- STOP statement

  COBOL-D

  When the STOP RUN statement is encountered in a called program, control is returned to the calling program.

  9400

  A STOP RUN statement causes an end-of-job termination sequence.

  C-mode

  When a STOP RUN statement is encountered in a called program, it is treated as an EXIT PROGRAM statement.

- USE AFTER STANDARD ERROR PROCEDURE

  COBOL-D

  The word 'PROCEDURE' is optional.

  9400

  The word 'PROCEDURE' is required.

  C-mode

  The word 'PROCEDURE' is optional.

- USE FOR LABEL PROCEDURE

COBOL-D

```
USE FOR { CREATING } { BEGINNING } LABELS
        { CHECKING } { ENDING    }

    ON { INPUT  }
       { OUTPUT }   file-name.
```

9400

```
USE { BEFORE } STANDARD { BEGINNING } { REEL }
    { AFTER  }          { ENDING    } { UNIT } LABEL
                                      { FILE }

    PROCEDURE ON { file-name }
                 { INPUT     }
                 { OUTPUT    }
                 { I-O       } .
```

C-mode

No automatic support. The USE statement for label procedures must be rewritten as per the UNIVAC 9400 format.

- WRITE statement

See P.4 printer considerations, and P.5 for disc considerations.

- *DEBUG card

COBOL-D

*DEBUG packets precede the source deck.

9400

*DEBUG packets follow the source deck.

C-mode

No automatic support. The *DEBUG packets must be moved from in front of the source program and placed behind the source program.

P.3.5. Reserved Words

C-mode

The following UNIVAC 9400 COBOL reserved words may currently exist in COBOL-D source programs as user-defined words. Their use as user names will not be allowed by the UNIVAC 9400 COBOL compiler.

ASCENDING
ASCII

BEFORE
BLOCK-COUNT
BLOCK-LENGTH-CHECK
BUFFER-OFFSET

CARD-PUNCH
CARD-READER
CARD-READER-51
CARD-READER-66
CHARACTERS
COMMA
COMPUTATIONAL
COMPUTATIONAL-3
COMPUTATIONAL-4
COMP
COMP-3
COMP-4
CORR
CORRESPONDING
CURRENCY
CYLINDER-INDEX
CYLINDER-OVERFLOW

DECIMAL-POINT
DESCENDING
DISC
DISC-8411
DISC-8414
DOWN

EQUALS
EXTENDED-INSERTION
EBCDIC

FILE-LIMIT
FILE-LIMITS
FILE-PREPARATION

INDICES
INDEX
INSERT

JUST

LINE

MAP
MASTER-INDEX
MEMORY
MODULE
MORE-LABELS
MULTIPLE

OFF
OPTIONAL

PERCENT
PIC
POSITION
PRINTER
PROGRAM

RELEASE
REMAINDER
RENAMES

SEARCH
SEGMENT-LIMIT
SEPARATE

SEEK
SET
SIGN
SORT
SPECIAL-NAMES
STATUS
SYNC
SYNCHRONIZED
SYSCHAN-4
SYSCHAN-5
SYSCHAN-6
SYSCHAN-7
SYSCHAN-8
SYSCHAN-9
SYSCHAN-10
SYSCHAN-11
SYSCHAN-12
SYSCHAN-13
SYSCHAN-14
SYSCHAN-15
SYSCOM
SYSCONSOLE
SYSDATE
SYSERR
SYSERR-0
SYSERR-1
SYSERR-2
SYSERR-3
SYSERR-4
SYSERR-5
SYSERR-6
SYSERR-7
SYSERR-8
SYSERR-9
SYSERR-10
SYSERR-11
SYSERR-12
SYSERR-13
SYSERR-14
SYSERR-15
SYSSWCH
SYSSWCH-0
SYSSWCH-1
SYSSWCH-2
SYSSWCH-3
SYSSWCH-4
SYSSWCH-5

SYSSWCH-6
SYSSWCH-7
SYSTIME

TAPE
TAPES
TAPE-6
THROUGH
TIME
TRACKS
TRAILING

UNIVAC-9400
UP

VALUES
VERIFY

WORDS
WHEN

P.4. PRINTER FILE SUPPORT

Support is available for printer files in the conversion mode of the COBOL compiler; the aim is to be as compatible as possible with COBOL-D printer file processing within the limits of hardware differences.

Due to the manner in which the UNIVAC 0768 Printer Subsystem hardware operates, an advance-then-print command (WRITE AFTER ADVANCING) results in the effective half-speed of the device. In COBOL-D, advance-then-print is the only option allowed.

When in the conversion mode, the compiler produces object code to change logical advance-then-print commands into physical print-then-advance operations. This causes full-speed operation of the UNIVAC 0768 Printer Subsystem. All printer files must be defined and referenced according to COBOL-D rules. Each printer file must have fixed recording mode and first character control. COBOL-D control characters must be used; consequently, neither a BEFORE ADVANCING nor an ADVANCING mnemonic-name will be supported in the source language. The only acceptable format for a printer WRITE statement is:

WRITE record-name FROM identifier

AFTER ADVANCING { identifier / literal } LINES.

Restrictions:

COBOL-D allows an APPLY FORM-OVERFLOW clause in the I-O-CONTROL paragraph of the Environment Division. The APPLY FORM-OVERFLOW clause must be converted to a USE FOR FORM-OVERFLOW procedure in the declaratives portion of the Procedure Division.

In COBOL-D, when APPLY FORM-OVERFLOW is specified, one line is printed after the overflow punch in the carriage control loop is detected. Due to the manner in which the logical write commands are converted into physical commands, three lines are printed after overflow is detected.

No action is taken when form overflow is encountered unless specified by a USE FOR FORM-OVERFLOW procedure.

To overcome the problem of three lines being printed after the overflow punch in the carriage control loop is crossed, the overflow punch must be moved back on the carriage control loop by two logical print commands (two lines if single spacing, four lines if double spacing, etc.). If the overflow punch crosses or coincides with another carriage control punch, the program cannot produce the proper print formats when the program is executed. The program is not transferable.

The testing of the condition-name specified in the APPLY FORM-OVERFLOW clause must be deleted from the existing Procedure Division and must not be used in the USE FOR FORM-OVERFLOW procedure. An alternate method, instead of deleting the processing associated with the overflow condition-name, is to leave the testing of the condition-name as is and to use the USE FOR FORM-OVERFLOW procedure as a place to set the condition-name to the true state.

The IBM model 1403 printer supports carriage-control channels 1 through 12. The UNIVAC 0768 Printer Subsystem supports carriage control channels 4 through 15 with both 14 and 15 signifying home paper. The COBOL-D carriage control references are translated as follows:

COBOL-D Control Character		Carriage Control Punch
1	home paper	15
2		12
3		13
4		4
5		5
6		6
7		7
8		8
9		9
A		10
B		11
C	form-overflow	9

P.5. DISC FILE SUPPORT

The following paragraphs detail considerations for conversion of COBOL source programs dealing with files assigned to direct access devices.

To facilitate an understanding of the differences between the COBOL compilers, a clause-by-clause, verb-by-verb difference description follows, by file organization.

P.5.1. Sequential Organization

- SELECT/ASSIGN clause

  The SELECT/ASSIGN clause requires a source program change to meet the format requirements of UNIVAC 9400 COBOL.

- APPLY VERIFY clause (not availabe in COBOL-D)

  When in C-mode, the compiler automatically sets the verify function without regard to the APPLY clause present in the source program.

- LABEL RECORD definition

  In C-mode, the compiler accepts the LABEL RECORD definition in the Linkage Section.

- REWRITE verb

  In C-mode, the compiler accepts the REWRITE verb when the file is opened for I/O.

- INVALID KEY phrase

  When C-mode is active, the compiler causes transfer to the USE FOR ERROR procedure or initiates an end-of-job sequence when an INVALID KEY condition is detected and there is no INVALID KEY phrase specified.

P.5.2. Indexed Organization

- SELECT/ASSIGN clause

  The SELECT statement with its ASSIGN clause requires a source program change to meet the format requirements of UNIVAC 9400

- APPLY VERIFY clause (not available in COBOL-D)

  In C-mode, the compiler automatically sets the verify function without regard to the APPLY clause.

- APPLY MASTER-INDEX clause (not available in COBOL-D)

  The APPLY MASTER-INDEX clause must be added to the source program if the file is to be created with a master index.

  NOTE:

  COBOL-D specifies this option via the job control stream.

- APPLY CYLINDER-OVERFLOW clause (not available in COBOL-D)

  If this clause is not inserted into the source program, the compiler specifies that 20% of each prime data cylinder is to be reserved for cylinder overflow area.

- APPLY CYLINDER-INDEX AREA clause (not available in COBOL-D)

  If this clause is not specified in the source program, the compiler does not allocate main storage area to accommodate the cylinder index.

- APPLY EXTENDED-INSERTION AREA clause (not available in COBOL-D)

  When this APPLY clause is not specified, the compiler does not allocate additional main storage for efficient addition of records to an update file.

- RECORD KEY description

  In C-mode, the record key size must not be less than 3 nor greater than 255 bytes.

- SYMBOLIC KEY description

  In C-mode, the symbolic key size must not be less than 3 nor greater than 255 bytes.

- WRITE verb

  When the C-mode is active, the compiler accepts the WRITE verb for the INSERT function when the file is opened for I/O.

- INVALID KEY phrase

  When C-mode is active, the compiler causes transfer to the USE FOR ERROR procedure or initiates an end-of-job sequence when an invalid key condition is detected and there is no INVALID KEY phrase specified.

- REWRITE verb

  In C-mode, the compiler accepts the REWRITE statement when the file is opened for I/O.

- Error testing in USE FOR ERROR procedures

  Replace any calls on DTF interrogation subprograms by tests of SYSERR's (defined in Special-Names paragraph) to determine error status.

P.5.3. Direct Organization (Absolute Addressing)

- SELECT/ASSIGN clause

  The SELECT/ASSIGN clause must be rewritten to conform to the OS/7 COBOL format.

- APPLY RESTRICTED SEARCH clause

  In C-mode, the restricted search logic of COBOL-D is employed. Absence of this clause implies a search to end of cylinder.

- APPLY VERIFY clause (not available in COBOL-D)

  When C-mode is active, the compiler automatically sets the verify function without regard to the APPLY clause.

- APPLY FILE-PREPARATION clause (not available in COBOL-D)

  In C-mode, the compiler automatically sets the file preparation function without regard to the APPLY clause.

- LABEL RECORD definition

  In C-mode, LABEL RECORD definitions in the Linkage Section are accepted.

- ACTUAL KEY usage

  If conversion mode is specified, the compiler accepts a track address in the form:

  m bb cc hh r

- ACTUAL KEY description

  In UNIVAC 9400 C-mode, the ACTUAL KEY PICTURE must be changed to reflect the M, bb, cc, hh, r fields as decimal integer (packed) fields.

- SYMBOLIC KEY description

  The symbolic key length must not be less than 3 nor greater than 255 bytes when the C-mode is active.

- WRITE verb

  In C-mode, the compiler accepts the WRITE verb for the INSERT function when the file is opened for I/O.

- REWRITE verb

  When in C-mode, the compiler accepts the REWRITE statement when the file is opened for I/O.

- INVALID KEY phrase

  In C-mode, the compiler causes a transfer to the USE FOR ERROR procedure or initiates an end-of-job sequence when an invalid key condition is detected and there is no INVALID KEY option.

- Error testing in USE FOR ERROR procedures

  Replace the calls on DTF interrogation subprograms by SYSERR tests to determine error status. SYSERR's are defined in the special-names paragraph.

INDEX