SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Extended COBOL

Programmer Reference

UP-8059 Rev. 3-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) Extended COBOL Programmer Reference", UP-8059 Rev. 3.

This update includes changes to the job control procedure for release 7.1:

- Specification of catalog files
- Expanded explanation of parameters

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8059 Rev. 3-A. To receive the complete manual, order UP-8059 Rev. 3.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ, (less DE, GZ, HA) MZ, 18U, 19U, 20U, 21U, 75U and 76U	Mailing Lists DE, GZ, HA, 18, 19, 20, 21, 75 and 76 (Package A to UP-8059 Rev. 3, 23 pages plus Memo)	Library Memo for UP-8059 Rev. 3-A
		RELEASE DATE: September, 1981

SPERRY UNIVAC COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

Extended COBOL

Programmer Reference

UP-8059 Rev. 3

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) Extended COBOL Programmer Reference", UP-8059 Rev. 3.

This revision includes the following additions and changes:

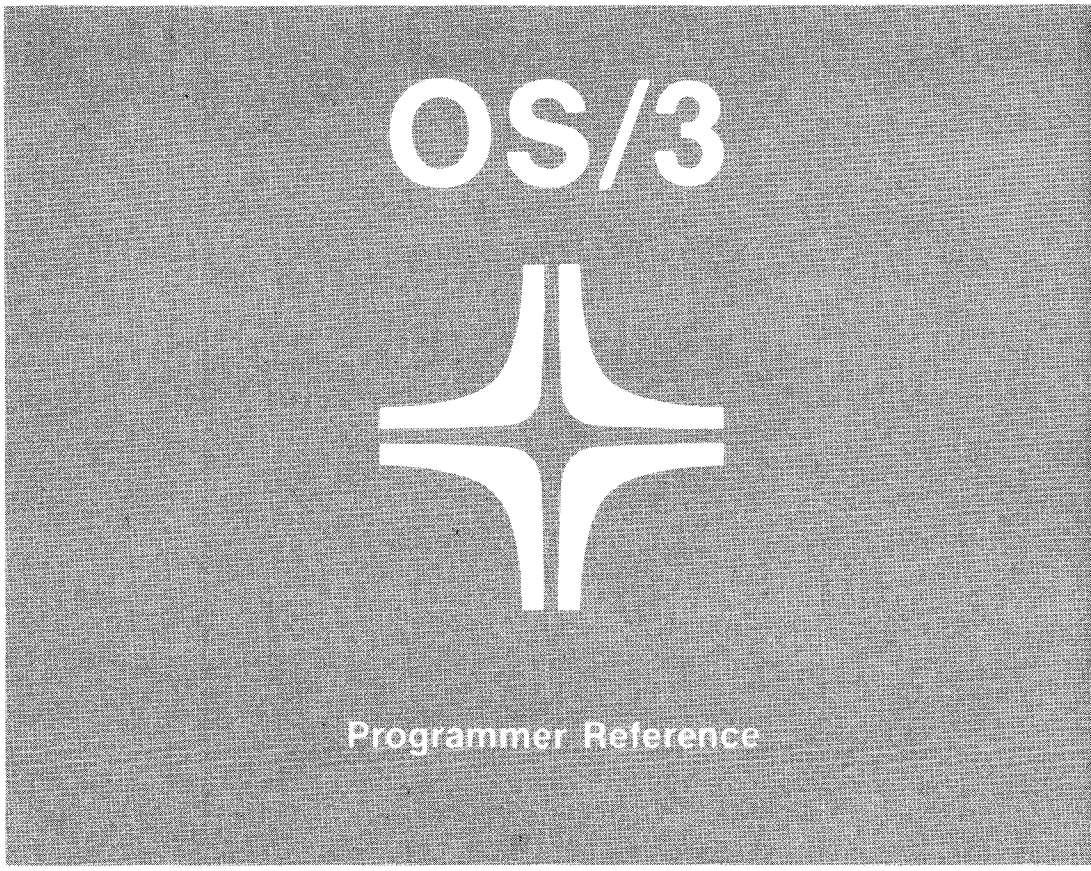
- Addition to Table 4–1 (Rules for Special Names)
- Change to BLKFAC formula
- Deletion in //△ EXEC operand
- Change to Table 11–5 Exceptions to COBOL verbs
- Addition of error messages
- Replace printout examples in Section E
- Addition of new pages and examples to Section E
- Addition to Reserved Word List
- Addition to DISPLAY rule

Other minor technical changes were also made.

Additional copies may be ordered by your local Sperry Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ,CZ (less DE,GZ,HA) MZ,18U,19U,20U,21U, 75U and 76U	Mailing Lists DE, GZ, HA, 18, 19, 20, 21, 75 and 76 (Covers and 298 pages)	Library Memo
		RELEASE DATE: October, 1980

Extended COBOL



Environment: 90/25, 30, 30B, 40 Systems

SPERRY UNIVAC

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: Update B – UP-8059 Rev. 3
RELEASE LEVEL: 7.1 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	B
Acknowledgment	1	Orig.
Preface	1 thru 3	Orig.
Contents	1, 2 3 4 thru 9	Orig. B Orig.
PART 1	Title Page	Orig.
1	1, 2	Orig.
2	1 thru 10	Orig.
PART 2	Title Page	Orig.
3	1, 2	Orig.
4	1 thru 13	Orig.
5	1 thru 14 15 16 thru 35	Orig. A Orig.
6	1 thru 3 4 5 thru 47 48 49 thru 61	Orig. B Orig. A Orig.
PART 3	Title Page	Orig.
7	1 thru 7	Orig.
8	1, 2	Orig.
9	1 thru 5	Orig.
10	1 thru 3	Orig.
11	1 thru 4 5 6 thru 28	Orig. B Orig.
12	1 thru 5	Orig.
13	1 thru 7	Orig.

Part/Section	Page Number	Update Level
PART 4	Title Page	Orig.
14	1 thru 4	Orig.
PART 5	Title Page	Orig.
Appendix A	1, 2	Orig.
Appendix B	1, 2	Orig.
Appendix C	1, 2	Orig.
Appendix D	1 thru 33 34	Orig. A
Appendix E	1 thru 15	Orig.
Appendix F	1 thru 13	Orig.
Appendix G	1 2 thru 4 4a 5 thru 9 10 10a thru 10c 11 thru 13 14, 15	Orig. A A Orig. A A A Orig.
Appendix H	1, 2	Orig.
Index	1 thru 4 5 6 thru 8	Orig. B Orig.
User Comment Sheet		

Part/Section	Page Number	Update Level

All the technical changes 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.

Acknowledgment

This manual is based on *American National Standard COBOL, X3.23–1974* developed by the American National Standards Institute. 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 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."

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY UNIVAC Operating System/3 (OS/3). This manual specifically describes the OS/3 COBOL extended compiler and its effective use. Its intended audience is not the novice, but the experienced programmer new to SPERRY UNIVAC operating systems, and to OS/3 in particular.

Two other manuals also are available for instruction and guidance in the use of OS/3 COBOL: one is a fundamental manual, and the other is the basic compiler manual. The fundamental COBOL manual, UP-7503.1 (current version) is useful for reviewing the language in some depth; however, it does not present the COBOL implementation for OS/3. The basic COBOL programmer reference manual, UP-8057 (current version) is intended for the novice programmer; some of the enhancements included in the extended compiler, such as sorting, additional verbs, and move options with various statements, are not discussed.

This manual is divided into the following parts:

- PART 1. COBOL LANGUAGE STRUCTURE

  Defines the rules, symbols, and minimum system configurations required to compile an OS/3 COBOL program. Also describes the character set, types of words, qualification, and subscripting and indexing and presents the layout of a coding form.

- PART 2. DIVISIONS IN COBOL

  Discusses the four divisions of COBOL, which are as follows:

  IDENTIFICATION – labels a program, providing entries of pertinent information regarding the author and installation of the program, when it was written and compiled, any security that might be involved, and its intended use.

  ENVIRONMENT – immediately follows the identification division entries and is coded to reflect specific user system configurations.

  DATA – divided into three sections:

  File Section – describes the records to be processed and the physical structure of files on which these records reside.

  Working-Storage Section – describes areas for intermediate or temporary storage of data that does not belong to any file.

  Linkage Section – describes data items that are passed by a calling program to a called program and are referred to by both the calling and the called program.

  PROCEDURE – specifies the instructions for the processor to use in solving the problem.

- PART 3. COMPILER FEATURES AND CAPABILITIES

  Describes options that can be used with the extended COBOL compiler, in addition to furnishing pertinent information that may be helpful in preparing a problem program.

  COMPILER OPTIONS AND LIBRARY TECHNIQUES – explains how to use specific COBOL options, such as generating certain listings in conjunction with compiling a problem program. The library techniques paragraphs describe SOURCE and COPY library input specifications.

  RERUN CLAUSE – provides a method of restarting the execution of a COBOL program at a checkpoint position, rather than at the beginning of the execution.

  USE OF ACCEPT AND DISPLAY STATEMENTS – defines the statements to use in retrieving or displaying low-volume data from or to system hardware.

  TABLE HANDLING – examines the methods of table definition and referencing available in OS/3 COBOL. For a complete discussion of table handling, see the fundamentals of COBOL–table handling manual, UP-7503.2 (current version).

  PROCESSING TECHNIQUES FOR DIRECT ACCESS DEVICES – Explains the various access methods available on the OS/3 system and describes the COBOL statements needed to interface with them.

  SORTING – explains the use of the OS/3 COBOL SORT feature, which offers an efficient means for sorting records against a set of specified keys and for adding, deleting, or modifying records in the sort file.

  ASCII PROCESSING – describes the option for using ASCII data and processing files encoded in ASCII (American Standard Code for Information Interchange).

- PART 4. DEBUGGING AIDS

  Illustrates the techniques of detecting, diagnosing, and correcting errors in the COBOL source program with the aid of the compiler.

- PART 5. APPENDIXES

  Presents the following appendixes:

  A. CHARACTER SET – contains conversion tables for characters and the character collating sequence.

  B. RESERVED WORDS – lists words that are part of the COBOL language structure but are not used as user-defined words.

  C. INTERMEDIATE RESULTS IN ARITHMETIC OPERATIONS – describes the internal work areas for certain arithmetic statements.

  D. COMPILER DIAGNOSTICS – lists the texts of the numbered diagnostic messages issued by the compiler, their severity codes, the probable reason for the error or condition detected by the compiler, the COBOL rules that apply, and the recovery actions taken by the compiler. Also listed are the system console messages that require programmer action.

  E. COMPILER LISTINGS – describes the listings received through the use of the PARAM statements in the job control stream.

  F. CONVERSION MODE – describes a facility allowing users of IBM/360 DOS COBOL level-D to transfer into OS/3 COBOL.

G. JOB CONTROL STREAM REQUIREMENTS – lists and describes the keyword parameters of the procedure call statement used to generate job control statements needed for compilation. Examples of call statements and generated control streams are included.

H. SHARED CODE INTERFACE – describes the interface necessary when under control of the Series 90 Information Management System (IMS/90).

Other OS/3 publications, referenced in this manual, will be necessary or useful to the programmer working with the extended COBOL compiler:

- Supervisor user guide, UP-8075 (current version)

  Provides information needed to access the communication region of the OS/3, through which one job step may communicate with a following job step.

- Job control user guide, UP-8065 (current version)

  Provides information on the format and usage of job control statements for accessing UPSI switches, allocating devices, and passing parameters to the object program.

- Data management system user guide, UP-8068 (current version)

  Provides SPERRY UNIVAC OS/3 standard file label specifications.

- Sort/merge user guide/programmer reference manual, UP-8074 (current version)

  Contains detailed information, including job control language, on the use of the OS/3 sort/merge facility, which the extended COBOL compiler employs for all sort operations.

- Error messages programmer/operator reference manual, UP-8076 (current version)

  Lists and describes the system console messages issued during compilation by the compiler, emphasizing error conditions during execution, and relating to sort operations.

Contents

4. ENVIRONMENT DIVISION

5. DATA DIVISION

6. PROCEDURE DIVISION

E. COMPILER LISTINGS

F. CONVERSION MODE

G. JOB CONTROL STREAM REQUIREMENTS

H. SHARED CODE INTERFACE

INDEX

USER COMMENT SHEET

FIGURES

TABLES

PART 1. COBOL LANGUAGE STRUCTURE

1. Introduction

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

The various language elements comprising 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 requisite to proper functioning of the statement;
- optional words included at the discretion of the user;
- information that must be supplied by the user;
- elements in the statement involving a choice by the user; and
- optional functions of the statement.

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

- The order of presentation is indicated by the format statement itself.
- All COBOL reserved words appear in all capitals. They are also listed in Appendix B.
- Words in underlined capitals are key words, which must be present when the functions in which they appear are used. Those capitalized words not underlined are optional and may be included at the user's discretion to improve readability; there is no compiler action. All completely capitalized words, whether underlined or not, are part of the COBOL language and must be spelled exactly as indicated.
- All lowercase words represent generic terms to be supplied by the user when the functions of which they are a part are used.
- Braces {} enclose elements of a statement to indicate that one of the elements must be selected. 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.
- Brackets [] enclose optional functions to indicate their inclusion or omission at the user's direction. When two or more options are stacked within brackets, one or none of them may be specified.
- In some statements, certain portions may be used as many times as needed by the programmer. The ellipsis. . . indicates this repeatability. If there is a choice to be made from stacked options or if there is only a single possibility, brackets or braces are used as delimiters to indicate that portion of the statement which is repeatable.

1.2. EXTENDED COBOL COMPILER

The SPERRY UNIVAC Operating System/3 (OS/3) extended COBOL compiler conforms to the specifications in the *American National Standard COBOL, X3.23–1968.* The modules and levels implemented are shown in Table 1–1; where OS/3 COBOL features are an extension to these requirements, an annotation is made in the text.

Table 1–1. SPERRY UNIVAC OS/3 COBOL Module/Level Implementation

Module	Level
Nucleus	2
Sequential access	2
Random access	2
Sort	2
Segmentation	2
Table handling	3
Library	2

The minimum system configuration required for this compiler includes:

- 3 disc work areas and 1 system disc
- 1 card reader or substitute device
- 1 printer or substitute device
- 65,536-byte main storage

The extended COBOL compiler requires the micrologic expansion feature for both the compiler and the generated object program.

The compiler and all compiler-produced object programs normally operate on data represented in Extended Binary Coded Decimal Interchange Code (EBCDIC) under control of the OS/3.

A COBOL source program can be entered in the compiler from the job stream file or from a disc library file. The compiler produces, as its final output, a relocatable object program on disc. This object module must be processed by the linkage editor being executed.

2. General Specifications

2.1. COBOL CHARACTER SET

The SPERRY UNIVAC Operating System/3 (OS/3) COBOL character set is a 52-character subset of the OS/3 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 Δ or a blank space)

. Period

< Less than

(Left parenthesis

+ Plus sign

$ Currency sign

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

) Right parenthesis

; Semicolon

– 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 OS/3 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 paragraph;
- in the NOTE statement of the procedure division; or
- in nonnumeric literals.

The apostrophe or the quotation mark may be embedded in a nonnumeric literal by invoking the appropriate LST PARAM option to specify one or the other as the delimiter. (See Section 7.) Only one of the parameters may be used in any given program. The use of either overrides the interchangeability of the apostrophe and quotation mark.

The following paragraphs describe the general usage of the various OS/3 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, or contain a space.

2.1.2. Characters Used for Punctuation

COBOL punctuation characters are:

'	Apostrophe (character used as delimiter for a nonnumeric literal and as an optional character for the quotation mark)
(	Left parenthesis
)	Right parenthesis
	Blank or space (written on coding form as Δ or a blank space)
.	Period
,	Comma
;	Semicolon
"	Quotation mark (See apostrophe.)

NOTE:

The normal mode for the compiler is to equate the apostrophe and the quotation mark as meaning the same thing. To embed either character within a nonnumeric literal, the PARAM options described in 7.1 may be used.

The comma and semicolon, when used in the general format descriptions, are for readability only and are not required. When used, the comma and semicolon always must be followed by a space.

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 Expressions

The characters used in arithmetic expressions 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 are:

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

Two types of words are used in OS/3 COBOL: user-supplied and reserved. The user-supplied words are listed and defined in Table 2–1. 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 OS/3 COBOL reserved words.

Table 2–1. User-Supplied Words (Part 1 of 3)

User-Supplied Words	Rules
Data-name	1. Contains 1 through 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.
External-name	1. A nonnumeric literal of 1 to 8 characters 2. A user-supplied label that duplicates the LFD name used in the job control stream to name a COBOL file
File-name	1. Rules 1 through 4 for data-name 2. A word that names a file described in the data division
Sort-name	1. Rules 1 through 4 for data-name 2. A word that names a file described in the data division but which may be used by the sort function only

Table 2–1. User-Supplied Words (Part 2 of 3)

User-Supplied Words	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 are 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
Fixed-point 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, 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.
Floating-point numeric literal	1. A string of not more than 22 characters, including 0 through 9, signs (+ or −), decimal point, and the character E 2. A floating-point numeric literal must be of the form: $\left[\left\{\pm\right\}\right]$ mantissa E $\left[\left\{\pm\right\}\right]$ exponent. 3. The mantissa must contain at least one and not more than 16 digits and a decimal point. 4. The exponent must contain one or two digits. 5. If the mantissa or the exponent is unsigned, it is assumed to be positive. 6. The maximum magnitude is 0.72×10^{76} 7. The minimum magnitude is 5.4×10^{-79}

Table 2–1. User-Supplied Words (Part 3 of 3)

User-Supplied Words	Rules
Nonnumeric literal	1. A string of any characters of the OS/3 character set, excluding the quotation mark and the apostrophe (unless these have been embedded by use of the appropriate LST parameter (7.1)); 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 whenever a nonnumeric literal appears in the format.

Table 2–2. Reserved Words (Part 1 of 2)

Reserved Words	Rules
Verbs	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-3 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 whenever an elementary data item of integral value may appear.
Figurative constants	1. ZERO, ZEROS, or ZEROES generates one or more 0's. 2. SPACE or SPACES generates one or more spaces. 3. HIGH-VALUE or HIGH-VALUES generates one or more hexadecimal FF characters (all binary 1's); this character has the highest value in the OS/3 collating sequence. 4. LOW-VALUE or LOW-VALUES generates one or more hexadecimal 00 characters (all binary 0's); this character has the lowest value in the OS/3 collating sequence.

Table 2–2. Reserved Words (Part 2 of 2)

Reserved Words	Rules
Figurative constants (cont)	5. QUOTE or QUOTES generates one or more apostrophes ('), hexadecimal 7D; QUOTE(S) cannot be used in place of quotation marks (") or an apostrophe to bound a nonnumeric literal. 6. The ALL literal generates one or more of the literals following the ALL; the literal must be either a nonnumeric 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, and NOT are used to form compound conditions. 3. A series connective is the comma, which links two or more consecutive operands or statements; the use of a series connective is optional.

2.3. QUALIFICATION

Every name used in an OS/3 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 a 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 must 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 as part of the qualifier.

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. FD names, SD names, level 77 names, level 66 names, level 01 names (not in the file section), 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/indexes must appear to the right of the last qualifier name.

Format 1:

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

Format 2:

paragraph-name [{ IN | OF } section-name]

NOTE:

Figure 2–1 illustrates examples of qualification entries.

```
SEQUENCE NUMBER | CONTINUATION | A | B  TEXT
1             6 | 7            | 8  11 | 12     20     30     40     50     60
      *  THE FOLLOWING ENTRIES SHOW THE USE OF QUALIFICATION.

       DATA DIVISION.
       WORKING STORAGE SECTION.
       77  HOLD PIC X(10) VALUE SPACES.
       01  ALA.
           03 CITY PIC X(20).
           03 WARD PIC X(10).

       01  PENNA.
           03 CITY PIC X(20).
           03 WARD PIC X(10).


       PROCEDURE DIVISION.

       PARA-1.
           MOVE WARD OF PENNA TO HOLD.

      *  NOTE IF THE DATA NAMES CITY AND WARD WERE UNIQUE THEN
      *  QUALIFICATION IS UNNECESSARY.
```

Figure 2–1. Example of Qualification Entries

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 identifiers 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. The integer must not be zero. Zero is considered out of the scope of an OCCURS clause.
5. When more than one subscript or index is used in a reference, each must be separated by a comma and a space.

NOTE:

Table handling is discussed in Section 10. For a complete discussion of table handling, see the fundamentals of COBOL – table handling manual, UP-7503.2 (current version).

2.5. CODING FORM

Figure 2–2 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. The divisions of the form are explained in Table 2–3.

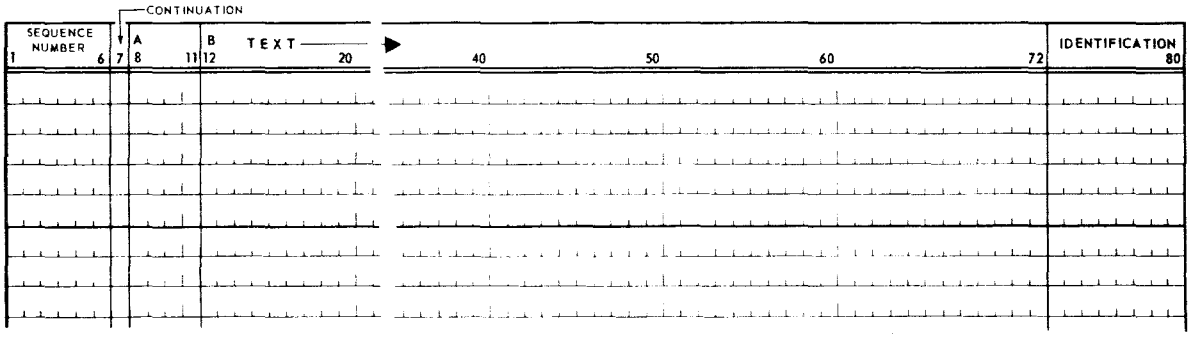


Figure 2–2. COBOL Programming Form

Table 2–3. Programming Form Column Usage

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 extends past one noncomment line. A break is used in the middle of a word, and 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 the word completed on the next line. If the continued line contains a nonnumeric literal without a closing delimiter (apostrophe or quotation mark), the first nonblank character in Area B of the continuation line must be one of these delimiters and the continuation starts with the character immediately after the delimiter.
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 except between a continuation set 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 used only for noncomment continuation lines.) This is an extension to *American National Standard COBOL* (1968).
7	EJECT	A slash (/) in column 7 signifies a comment line that causes the compiler to direct the printer to skip to the head of the form and print the comment. If the comment line is continued, it must follow the rules for comment continuation, as explained in the preceding paragraph.
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 continuation of entries from the last preceding line.
73–80	IDENTIFICATION	Card deck information (optional)

PART 2. DIVISIONS IN COBOL

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 paragraph will affect the object program in the SPERRY UNIVAC Operating System/3 (OS/3).

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. The identification division must be present in all source programs.

2. PROGRAM-ID always must 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 (within user's library) since it will identify the source program, object program elements, and associated documents. Hyphens within the first six characters are removed by the compiler due to OS/3 naming conventions.

   If the program name is not supplied or not accepted due to an error, the compiler automatically supplies the program name NOCOBNAM.

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.

Example:

An example of identification division entries is shown in Figure 3–1.

```
       CONTINUATION
SEQUENCE  |  A     B  TEXT ---------------------->
NUMBER    |
1       6 |7 8  11 12        20        30        40
001001     IDENTIFICATION DIVISION.
001002     PROGRAM-ID. TEST01.
001003     AUTHOR. SYSTEMS PUBLICATIONS.
001004     INSTALLATION. DEPT 6866.
001005     DATE-WRITTEN. OCT 12 1973.
001006     DATE-COMPILED. DEC 12 1973.
001007     SECURITY. NONE.
001008     REMARKS. USER REPORT # 1.
```

Figure 3–1. Example of Identification Division Entries

4. Environment Division

4.1. GENERAL

The environment division (Figure 4–1) specifies those elements of the COBOL program that depend upon the physical aspects of the SPERRY UNIVAC 90/30, 90/25, or 90/40 System.

Format:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  { UNIVAC-9030. }
                  { UNIVAC-9025. }
                  { UNIVAC-9040. }
OBJECT-COMPUTER.  { UNIVAC-9030 }
                  { UNIVAC-9025 }
                  { UNIVAC-9040 }
[ , MEMORY SIZE integer { CHARACTERS } ]  [SEGMENT-LIMIT IS priority-number].
                        { MODULES    }
                        { WORDS      }
[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 in the SPERRY UNIVAC Operating System/3 (OS/3). It may need to be rewritten when a program is to be compiled or executed on a different system configuration.

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

3. Section and paragraph headers must begin in margin A (columns 8–10); their associated entries must begin in margin B (columns 12–71).

4.2. CONFIGURATION SECTION

Definition:

The configuration section specifies the characteristics of the source and object processors and relates implementor-names to user-names.

Format:

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

4.2.1. SOURCE-COMPUTER Paragraph

Function:

Names the processor that will compile the source program.

Format:

```
SOURCE-COMPUTER.  { UNIVAC-9030. }
                  { UNIVAC-9025. }
                  { UNIVAC-9040. }
```

Rules:

The SOURCE-COMPUTER paragraph is for documentation only and does not affect the object program.

4.2.2. OBJECT-COMPUTER Paragraph

Function:

To specify the processor that will execute the object program and the size of main storage and the segment-limit priority number.

Format:

```
OBJECT-COMPUTER.  { UNIVAC-9030 }
                  { UNIVAC-9025 }
                  { UNIVAC-9040 }

[ MEMORY SIZE integer { CHARACTERS } ]  [SEGMENT-LIMIT IS priority-number].
                      { MODULES    }
                      { WORDS      }
```

Rules:

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

2. MEMORY SIZE is an optional clause defining main storage as an integer number (no sign, comma, or decimal point permitted) of WORDS, CHARACTERS, or MODULES for documentation only. 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.

4.2.3. SPECIAL-NAMES Paragraph

Function:

Relates 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-6]

[; 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]] }
   { IS mnemonic-name-7 [OFF STATUS IS condition-name-6 [ON STATUS IS condition-name-5]] }
   { ON STATUS IS condition-name-5 [OFF STATUS IS condition-name-6] }
   { OFF STATUS IS condition-name-6 [ON STATUS IS condition-name-5] } ] ...

[; SYSIN IS mnemonic-name-8]
[; SYSIN-96 IS mnemonic-name-9]
[; SYSIN-128 IS mnemonic-name-10]
[; SYSLOG IS mnemonic-name-11].
```

where:

t

Is any digit 1 through 15.

m

Is any digit 0 through 31.

n

Is any digit 0 through 7.

Rules:

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

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 OS/3 COBOL character set and must not be one of the following characters:

   - Digits: 0 through 9
   - Alphabetic characters: A, B, C, D, E, 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	ΔΔΔF32,32
9(5)V99	1234567	F**.***,99	F12.345,67
9(9)V9(4)	0000098211289	Z(3).ZZ9,9(4)	ΔΔ9.821,1289

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 the supervisor user guide, UP-8075 (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 system console (using ACCEPT or DISPLAY statement; see Section 9) 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 1 and 7 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 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[-n] 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.

- 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.

- An individual switch can be interrogated by using condition-name in the ON/OFF STATUS option. For instance, in the following example control is 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.

- The entire UPSI byte may be interrogated by use of the ACCEPT verb. This is shown in the following example where procedure-name-1 is 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.
```

- 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 branch to procedure-name-1.

- 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 only with the ACCEPT or DISPLAY verbs.

- The UPSI switches also can 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 job control user guide, UP-8065 (current version).

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

12. SYSIN permits access to embedded data in the control stream when the embedded data is supplied on 80-column cards. Access is made via mnemonic-name-8 and the ACCEPT statement.

13. SYSIN-96 permits access to embedded data in the control stream when the embedded data is supplied on 96-column cards. Access is made via mnemonic-name-9 and the ACCEPT statement.

14. SYSIN-128 permits access to embedded data in the control stream when the embedded data is supplied on an 8413 diskette. Access is made via mnemonic-name-10 and the ACCEPT statement.

15. SYSLOG permits access to the system console and log file via mnemonic-name-11 and the DISPLAY statement.

16. 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.

NOTE:

See Section 9 for further discussion of ACCEPT and DISPLAY statements.

Table 4–1. Rules for SPECIAL-NAMES

SPECIAL-NAME	Stored as	Usable With		Format	Implied Description for ACCEPT or DISPLAY ③	Explanation
		ACCEPT	DISPLAY			
SYSCOM	12 alphanumeric characters	Yes	Yes	12 EBCDIC characters	PIC X(12)	See the supervisor user guide, UP-8075 (current version).
SYSDATE	6 numeric characters	Yes	No	yymmdd	PIC 9(6)	Current day
SYSTIME	8 numeric characters	Yes	No	hhmmss00	PIC 9(8)	Time of day
SYSCONSOLE	Variable-length alphanumeric characters	Yes	Yes	For DISPLAY: 55 characters per line, up to 250 For ACCEPT: 50 characters maximum	PIC X(n)	System console
SYSCHAN-t ②	Not applicable	No	No	Not applicable	Not applicable	To assign name to printer loop channel
SYSERR[-m]	Not applicable	No	No	Not applicable	Not applicable	Refer to Section 11.
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	120 characters/ line	PIC X(n)	Printer with LFD name of SYSLST
SYSIN	Variable-length alphanumeric characters	Yes	No	80 characters/ card	PIC X(n)	Embedded control stream data cards (80-column)
SYSIN-96	Variable-length alphanumeric characters	Yes	No	96 characters/ card	PIC X(n)	Embedded control stream data cards (96-column)
SYSIN-128	Variable-length alphanumeric characters	Yes	No	128 characters on diskette	PIC X(n)	Embedded control stream data (8413 diskette)
SYSLOG	Variable-length alphanumeric characters	No	Yes	55 characters	PIC X(n)	System console and log file (no operator response)
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 14.

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 Paragraph

Function:

The FILE-CONTROL paragraph names each file, identifies the hardware medium containing it, permits specific hardware assignments for the program, and specifies alternate input/output areas. The clauses following SELECT and ASSIGN 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 {EXTENDED  }
                   {RANDOM    }
                   {SEQUENTIAL}

[ ; PROCESSING MODE IS SEQUENTIAL]

[ ; ORGANIZATION IS {INDEXED   }
                    {RELATIVE  }
                    {SEQUENTIAL}

[ ; {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 clause, and a period must follow the entry.

2. A SELECT clause must be specified for the following:

   - Every file that is the subject of an FD or SD.
   - The external-name operand of a RERUN clause for which no FD or SD is supplied.

3. The keyword OPTIONAL, which may be applied to input files only, 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. If the file is not present in the job stream, control takes the path specified by the AT END statement on the first READ statement. The keyword OPTIONAL can be applied to input files only, and these files must be sequential.

4. 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 job control statement). The external name must be unique within a job step. 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. SPERRY UNIVAC OS/3 COBOL assigns the following implementor-names:

Device	Implementor-Name
51-column card reader	CARD-READER-51
66-column card reader	CARD-READER-66
80- or 96-column card reader or 8413 diskette subsystem	CARD-READER
Card punch or 8413 diskette subsystem	CARD-PUNCH
Line printer	PRINTER
SPERRY UNIVAC 8411 Disc Subsystem	DISC-8411
SPERRY UNIVAC 8414 Disc Subsystem	DISC-8414
SPERRY UNIVAC 8415 Disc Subsystem	DISC-8415
SPERRY UNIVAC 8416 Disc Subsystem	DISC or DISC-8416
SPERRY UNIVAC 8418 Disc Subsystem	DISC-8418
SPERRY UNIVAC 8430 Disc Subsystem	DISC-8430
SPERRY UNIVAC 8433 Disc Subsystem	DISC-8433
UNISERVO VI-C Magnetic Tape Subsystem	TAPE-6
All other tapes	TAPE

   The implementor-name, DISC, specifies an assignment to the 8416 disc subsystem. Because of track size differences, the user must ensure that the proper implementor-name is used when assigning discs.

   The implementor-name, CARD-READER, is used when reading 80- or 96-column cards or when reading data from an 8413 diskette device. If the record size specified in the data division is greater than the physical record size of the medium, the remaining character positions in the record will contain spaces.

5. 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, which specifies the devices needed for the problem program.

6. The RESERVE clause indicates the number of additional I/O areas desired. The keyword NO causes no additional I/O areas to be reserved; integer-2 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		Number of Additional I/O Areas Allocated if Clause Not Specified	Reserve Integer Allowed
CARD-READER		1	Yes
CARD-PUNCH		1	Yes
PRINTER		1	Yes
TAPE		1	Yes
DISC	ORGANIZATION SEQUENTIAL (or omitted)	1	Yes
	ORGANIZATION INDEXED	0	Yes
	ORGANIZATION RELATIVE	0	No

7. FILE-LIMIT clause serves as documentation only.

8. ACCESS MODE specifies the manner in which the records of a file are read and/or written. Absence of this clause results in assumption of sequential access.

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

10. 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 (1968).

11. ACTUAL KEY IS data-name-5. See RELATIVE KEY explanation.

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

    NOTE:

    In this case, the ORGANIZATION clause must appear first.

12. RELATIVE KEY IS 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 relative record numbers 1, 2, 3, and so on. 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. The RELATIVE KEY clause is an extension to American National Standard COBOL (1968).

13. SYMBOLIC KEY IS data-name-7 is used for indexed file organizations to supply the record identification for random retrieval and sequential positioning. The information associated with the RECORD KEY clause must be identical with the information associated with the SYMBOLIC KEY clause. Data-name-7 must consist of 3 to 249 characters (may be numeric computational). This clause is an extension to American National Standard COBOL (1968).

14. RECORD KEY IS data-name-8 is used for indexed-organized files to supply the record identification field. Data-name-8 must consist of 3 to 249 bytes. This clause is an extension to American National Standard COBOL (1968).

A detailed explanation of the various keys and types of file organization is given in Section 11.

4.3.2. I-O-CONTROL Paragraph

Function:

Specifies the following:

- Input/output techniques
- Main storage area shared by various files
- Location of each file on multiple-file-reel
- 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 } ] AREA FOR file-name-3 { , file-name-4 } ... ] ...
                { SORT }
[; MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-2]
    [file-name-6 [POSITION integer-3] ] ... ] ...
[; 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 [, file-name-22] ... ] ...
```

†Accepted for SPERRY UNIVAC Operating System/4 (OS/4) and Operating System/7 (OS/7) compatibility only.

Rules:

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

2. The RERUN clause 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 processor 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 are 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. The SAME AREA clause specifies that two or more files are to use the same main storage 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 formats of such files must not conflict). The SAME RECORD AREA clause should be used only when necessary because it reduces efficiency.

   IF the SAME SORT AREA clause is used, at least one of the file-names must be a sort file and the subject of an SD. Storage areas 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.

5. 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.

6. The APPLY BLOCK-COUNT causes a 3-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.

7. The APPLY FILE-PREPARATION clause indicates that the tracks allocated to a relative organized file are to be recorded with initializing data prior to creation of a file. The track initialization occurs after an OPEN OUTPUT command is issued.

8. The APPLY MASTER-INDEX clause is only accepted for OS/4 and OS/7 compatibility. In OS/3, this clause serves for documentation only.

9. 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 top index entries.

10. The APPLY CYLINDER-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. If no overflow exists, then no new records can be inserted into the file. Integer-6 is an unsigned number.

11. The APPLY EXTENDED-INSERTION clause is accepted for OS/4 and OS/7 compatibility. In OS/3, this clause serves for documentation only.

12. The use of the APPLY ASCII clause, which identifies each file that contains or receives ASCII data, is explained in Section 13.

NOTE:

APPLY clauses (rules 5 through 12) are extensions to American National Standard COBOL (1968). Further discussion of I-O-CONTROL is given in Section 11.

PROGRAM

SEQUENCE NUMBER 1 6	CONTINUATION 7	A 8 11	B 12 TEXT 20 30 40
001010		ENVI	RONMENT DIVISION.
001011		CONF	IGURATION SECTION.
001012		SOUR	CE-COMPUTER. UNIVAC-9030.
001013		OBJE	CT-COMPUTER. UNIVAC-9030.
001014		SPEC	IAL-NAMES.
001015			SYSCONSOLE IS TYPEIT.
001016		INPU	T-OUTPUT SECTION.
001017		FILE	-CONTROL.
001018			SELECT INPUT1 ASSIGN TO TAPE-6.
001019			SELECT LIST ASSIGN TO PRINTER.
001020			SELECT CDS ASSIGN TO CARD-READER.
001021		I-O-	CONTROL.
001022			APPLY BLOCK-COUNT ON INPUT1.

Figure 4—1. Example of Environment Division Entries

5. Data Division

5.1. GENERAL

Every data item referenced in the procedure division of a SPERRY UNIVAC Operating System/3 (OS/3) COBOL program must be described in the data division (Figure 5–1) 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 as described in 5.3.

Format:

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

*Extension to American National Standard COBOL (1968).

Rules:

1. The division 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 because they cannot be qualified. The same is 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 OS/3 COBOL. Data type is determined by the PICTURE and USAGE clauses. See 5.3.4 for legal PICTURE characters for each data type.

Table 5–1. Main Storage Allocation

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
Alphanumeric edited	2	132	2	132
Numeric edited	2	132	2	132
Decimal numeric display	1	18	1	18
Floating-point numeric display	6	22	6	22
Numeric COMP or numeric COMP 4	1	18	2	8
Numeric COMP-1	Not applicable	Not applicable	4	4
Numeric COMP-2	Not applicable	Not applicable	8	8
Numeric COMP-3	1 (plus sign)	18 (plus sign)	1	10
Index name	Not applicable	Not applicable	8	8
Index data item	Not applicable	Not applicable	8	8

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 and describe in detail each record format used in the file.

Format:

```
{ FD file-name-1 (file description clauses)                    }
{ {01 record-name-1 (record description clauses)} ...         } ...
```

5.2.1. File Description

Function:

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

Format:

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

Rule:

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

5.2.1.1. BLOCK CONTAINS Clause

Function:

Specifies the size of a physical record.

Format:

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

Rules:

1. Integer-1 and integer-2 must be unsigned integers other than 0.

2. If the RECORDS option is specified and RECORDING MODE is F, this clause specifies the number of records per block.

   If the RECORDS option is specified and RECORDING MODE is V or D, this clause specifies the number of the longest records per block. It is also possible for a block to contain a greater number of the shorter records, depending on the differences in record sizes.

3. When CHARACTERS is specified, this clause specifies the number of characters (bytes) per block (physical record).

   a. For files not assigned to disc devices, this does not include the three bytes for the APPLY BLOCK-COUNT for tape, but does include the block header and record header bytes if recording mode is V. (The block header is four bytes per block, and the record header is four bytes per logical record.)

   b. For files assigned to disc devices, this number includes all the control fields associated with the data portion of the disc block. This number does not include the key field and count field lengths associated with those fields of the disc block. Table 5–2 shows the size of the control fields associated with the block.

Table 5–2. Control Field Sizes

Field	Bytes per Field			
	Organization: Sequential, Relative		Organization: Indexed	
	Recording Mode		Recording Mode	
	F	V	F	V
Block Header (BLKHDR)	0	4	2	2
Record Header (RECHDR)	0	4	0	2
Indexed Record Pointer (LINK)	0	0	5	5

The values are defined according to the recording mode and organization of the file. The programmer must define the values for the logical record size and a blocking factor. The blocking factor (BLKFAC) equals the number of logical records per physical block. The logical record size (LOGRECSIZE) is also determined by the recording mode.

If the recording mode is F, the logical record size is the size of the 01 record defined in the file FD. If the recording mode is V, the size of the logical record is equal to the size of the largest logical record.

Thus, the size of the physical block may be calculated according to the following formula:

BLKHDR + (BLKFAC * (RECHDR + LOGRECSIZE + LINK))

4. This clause may be specified for CARD-READER and CARD-PUNCH files to provide greater processing efficiency if the device is an 8413 diskette subsystem. In this case, the BLOCK clause does not specify the size of a physical block, but specifies the size of the buffer areas to be used for multisector I/O.

   a. If the RECORDS option is used, the size of the buffer area may be calculated by using the following formula:

      BLKFAC*(RECHDR + LOGRECSIZE)

      The maximum buffer size is 1024 bytes; therefore, the blocking factor (BLKFAC) the programmer selects must be equal to or less than 1024 divided by (RECHDR + LOGRECSIZE).

   b. If the CHARACTERS option is used and RECORDING MODE IS F, the BLOCK CLAUSE integer may be any multiple of (RECHDR + LOGRECSIZE) up to 1024.

5. When CHARACTERS and RECORDS are both omitted, CHARACTERS is assumed.

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

7. If both integer-1 and integer-2 are specified, integer-1 is treated as documentation only. Block size ranges are given in Table 5–3.

5.2.1.2. RECORD CONTAINS Clause

Function:

Specifies 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 0; 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:

   - 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.

   - If both integer-1 and integer-2 are shown, they refer to the minimum and maximum size data record.

Table 5–3. Block Size Ranges

Hardware Device and Implementor Name	Bytes per Block							
	Organization: Sequential, Relative				Organization: Indexed			
	Recording F & U		Recording V		Recording F & U		Recording V	
	Min	Max	Min	Max	Min	Max	Min	Max
CARD-READER	1	1024 ⑤	9	1024 ⑤	–	–	–	–
CARD-READER-51	1	51	–	–	–	–	–	–
CARD-READER-66	1	66	–	–	–	–	–	–
CARD-PUNCH	1	1024 ⑤	9	1024 ⑤	–	–	–	–
PRINTER	1	①	9	②	–	–	–	–
UNISERVO VI-C (TAPE-6)	18 ③	4,096	18 ③	4,096	–	–	–	–
UNISERVO VI-C (TAPE-6) with block numbering	18 ③	4,092	18 ③	4,092	–	–	–	–
Other tapes ④ (TAPE)	18 ③	32,767	18 ③	32,767	–	–	–	–
Other tapes ④ (TAPE) with block numbering	18 ③	32,763	18 ③	32,763	–	–	–	–
8411 disc (DISC-8411)	1	3,625	9	3,625	4	3,625	14	3,625
8414 disc (DISC-8414)	1	7,294	9	7,294	4	7,294	14	7,294
8415 disc (DISC-8415)	1	10,240	9	10,240	10	10,240	12	10,240
8416 disc (DISC-8416 or DISC)	1	10,240	9	10,240	10	10,240	12	10,240
8418 disc (DISC-8418)	1	10,240	9	10,240	10	10,240	12	10,240
8430 disc (DISC-8430)	1	13,030	9	13,030	10	13,030	12	13,030
8433 disc (DISC-8433)	1	13,030	9	13,030	10	13,030	12	13,030

NOTES:

① For 768 size = 132; for 770 size = 160; for 773 size = 144.

② For 768 size = 140; for 770 size = 168; for 773 size = 152.

③ Minimum size is 20 if tape is RERUN receiver.

④ Maximum size is 8192 if multiplexer channel is used.

⑤ Note that the maximum physical block is 128 characters (8413 diskette), 96 characters (96-column card), or 80 characters (80-column card). The larger block size is used to specify multisector I/O when the device is an 8413 diskette. (See 5.2.1.1, Rule 4.)

5.2.1.3. LABEL RECORDS Clause

Function:

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

Format:

```
LABEL { RECORDS ARE } { OMITTED                              }
      { RECORD IS   } { STANDARD                             }
                      { data-name-1 [,data-name-2] . . .     }
```

Rules:

1. The OMITTED clause 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. The STANDARD clause specifies that standard file labels exist for the file or the device to which the file is assigned, and the labels conform to OS/3 label specifications. (Refer to the data management user guide, UP-8068 (current version).) Standard user labels may also be present, but the STANDARD clause specifies that they are not to be checked on input files, or written on output files.

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

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

   - Data-name-1 [,data-name-2] ... must have a record description subordinate to this file description.
   - For input files, data management provides access to standard user label information in the area described by data-name-1.
   - For output files, the user moves user label information into the area described by data-name-1 for data managment to write to the output file.
   - User label records can be referenced only in USE procedures in the declaratives section (6.2).

4. The label record specifications for the various device types are as shown in Table 5–4.

Table 5–4. Label Record Specifications

Device		Labels Omitted	Labels Standard	Labels Data-name
PRINTER		Yes	No	No
CARD-READER		Yes	No	No
CARD-PUNCH		Yes	No	No
TAPE		Yes	Yes	Yes
DISC	ORGANIZATION SEQUENTIAL	No	Yes	Yes
	ORGANIZATION RELATIVE	No	Yes	Yes
	ORGANIZATION INDEXED	No	Yes	No

5.2.1.4. RECORDING MODE* Clause

Function:

Specifies the format of the logical record comprising the file.

Format:

RECORDING MODE IS { D / F / U / V }

Rules:

1. The D mode may be specified for ASCII tape files with variable-length records.
2. The F mode (fixed-length format) is specified when all the logical records in the file are of the same length.
3. The U mode (undefined format) states that the records of this file are not blocked and may vary in length. This mode is not allowed in SORT files (SD), nor is it available for disc files.
4. The V mode (variable-length format) is specified when records within a file vary in length.
5. The following chart describes the recording mode assumed when the clause is omitted.

**Extension to American National Standard COBOL (1968).*

Device		Assumed Format
PRINTER		F
CARD-READER		F
CARD-PUNCH		F
TAPE		V
DISC	ORGANIZATION SEQUENTIAL	V
	ORGANIZATION RELATIVE	F
	ORGANIZATION INDEXED	F

5.2.1.5. VALUE OF Clause

Function:

Describes a particular item in the standard file label record associated with a file; this clause serves as documentation only.

Format:

```
VALUE OF [ unqualified-data-name IS { data-name-3 } ] ...
                                    { literal-1    }
```

5.2.1.6. DATA RECORDS Clause

Function:

Specifies the names of the logical records in a file.

Format:

```
DATA { RECORDS ARE } data-name-4 [, data-name-5] ...
     { RECORD IS   }
```

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:

Identifies the beginning of a sort file description (SD) and supplies the name of the file.

Format:

```
SD file-name
[ ; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]
[ ; RECORDING MODE IS { D } ]
                      { F }
                      { V }
[ ; DATA { RECORDS ARE } data-name-1 [ , data-name-2] ... ]
         { RECORD IS   }
```

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. File-name may also appear in SAME RECORD AREA and SAME SORT AREA clauses in the environment division.
6. A summary of the OS/3 COBOL SORT formats is given in Section 12.

Extension to American National Standard COBOL (1968).

5.3. DATA DESCRIPTION

Function:

Defines the characteristics of a particular data item.

Format 1:

```
level-number  { FILLER                    }
              { unqualified-data-name-1    }
[ ; REDEFINES unqualified-data-name-2]
[   OCCURS integer-2 TIMES [ { ASCENDING  } KEY IS data-name-2 [ , data-name-2] ... ] ...
                           { DESCENDING }
      [INDEXED BY index-name-1 [ , index-name-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] ... ]
]
[ ; { PIC     } IS character-string ]
    { PICTURE }
[                          { COMP                }
                           { COMPUTATIONAL       }
                           { COMP-1              }
                           { COMPUTATIONAL-1     }
                           { COMP-2              }
; [USAGE IS]               { COMPUTATIONAL-2     }
                           { COMP-3              }
                           { COMPUTATIONAL-3     }
                           { COMP-4              }
                           { COMPUTATIONAL-4     }
                           { DISPLAY             }
                           { INDEX               }
]
[ ; MAP IS integer-3 CHARACTERS]
[ ; { SYNC         } [ { LEFT  } ] ]
    { SYNCHRONIZED }   { RIGHT }
[ ; { JUST      } RIGHT ]
    { JUSTIFIED }
[ ; VALUE IS literal]
[ ; BLANK WHEN ZERO]
[ ; { [SIGN IS] { LEADING  } SEPARATE CHARACTER }
    {           { TRAILING }                   }
    { [SIGN IS] TRAILING                       }
]
```

Format 2:

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

Format 3:

```
88 condition-name; { VALUES ARE } literal-1 [THRU literal-2]
                   { VALUE IS  }
    [literal-3 [THRU literal-4] ] ...
```

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

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

Rules:

1. A level number is required as the first element in each data description entry.
2. Level-number 01 through 09 may be expressed without the leading 0's.
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 levels 66, 77, and 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 defined 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 Clause

Function:

Allows the same area of computer main storage 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. The REDEFINES clause must not be applied to level 01 entries in the file or linkage sections, 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 detected. 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, an 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.

9. See rule 5 in 5.3.6 for use of REDEFINES with SYNCHRONIZED clause.

5.3.3. OCCURS Clause

Function:

Eliminates the need for separate entries for repeated data, and supplies information required for the application of subscripts or indexes.

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  } ]
                                                               { DESCENDING }
    KEY IS data-name-2 [ , data-name-3] ... ] ... [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 that is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than the SEARCH statement. 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, since its format is dependent on the hardware and storage is allocated by the compiler.

3. The data description clauses associated with an item that 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:

   - contains an 01, a 66, a 77, or an 88 level-number; and
   - describes an item whose size is variable. 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 clause is used to indicate 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:

   - all of the items identified by the data-names in KEY IS must be within the group item which is the subject of this entry; and
   - none of the items identified by data-names in KEY IS 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 that 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 has 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 0). The area allocated, multiplied by the number of occurrences, cannot exceed 65,535. Integer-1 may be positive or 0 but must be less than integer-2. The integer-1 TO option is an extension to American National Standard COBOL (1968).

12. A data description entry that contains format 2 of the OCCURS clause may be followed, within that record description, only by data description entries which are 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 exceed the value of data-name-1, unpredictable. The data description of data-name-1 must describe a positive integer.

16. When the user references a group item whose subordinate entry has a format 2 of the OCCURS clause, the actual length of the group item is determined by the current value of data-name-1. The user, therefore, should initialize the value of data-name-1 before any operation on the group item takes place.

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

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

19. See rule 3 in 5.3.6 for use of OCCURS with the SYNCHRONIZED clause.

5.3.4. PICTURE Clause

Function:

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

Format:

```
[ ; { PIC     } IS character-string ]
    { PICTURE }
```

Rules:

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

2. The PICTURE character-string can consist of 1 to 30 characters.

3. Five categories of data can be described with a PICTURE clause:

   - Alphabetic
   - Numeric
   - Alphanumeric
   - Alphanumeric edited
   - Numeric edited

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

4. To define an item as alphabetic:

   - Its picture character-string may consist of only the symbol A.
   - Its contents, when represented in standard data format, must be any combination of the 26 letters of the alphabet and the space.
   - Maximum number of character positions allowed is 4092.

5. To define an item as numeric:

   There are two types of numeric items: fixed-point numeric items and floating-point numeric items. To define an item as fixed-point numeric:

   - The PICTURE character-string may consist of only the symbols 9, P, S, V, and H.
   - The PICTURE character-string must contain at least one 9.
   - The maximum number of digits is 18.
   - The maximum number of occurrences of P in a picture-string is 17.
   - The contents, when represented in standard data format, must be a combination of the numerals 0 through 9. The item may include an operational sign.

   There are two kinds of floating-point numeric items*: computational (COMP-1 or COMP-2) and display. To define a floating-point display numeric item:

   - A floating-point display item has a picture-string in the form:

     $\{\pm\}$ mantissaE $\{\pm\}$ exponent

   - A plus or minus sign must begin the picture-string. The plus sign is used when both plus and minus signs are present in the data. When a positive quantity in the data is represented by a space, the minus sign is used in the picture-string. This sign occupies one byte of storage.
   - Only certain symbols (. 9 V) may be used in the mantissa. Up to 16 occurrences of 9 are permitted. The period is used to represent an actual decimal point, and V is used to represent an assumed decimal point. One or the other is necessary in the mantissa. The V does not occupy any storage.
   - E is used to signal the exponent portion of the item. This character occupies one byte of storage.
   - A second sign precedes the exponent. The same rules apply as described for the sign preceding the mantissa.
   - The exponent is represented by two 9 symbols.
   - The value of a floating-point display numeric item is equal to the product of the mantissa and the power of 10 represented by the exponent. The value must fall within the range:

     $\pm 5.4 \times 10^{-79}$ to $\pm 0.72 \times 10^{76}$.

Extension to American National Standard COBOL (1968).

- A VALUE clause cannot be associated with a floating-point display numeric item.

To define an item as floating-point numeric computational (COMP-1, COMP-2), the PICTURE clause is not used; see USAGE clause, rule 9, 5.3.5.

6. To define an item as alphanumeric:

   - 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.
   - Its contents, when represented in standard data format, are any combination of characters in the UNIVAC OS/3 system character set.
   - Maximum number of character positions allowed is 4092.

7. To define an item as alphanumeric edited:

   - Its character-string is restricted to combinations of the symbols A, X, 9, B, and 0 and must contain:
     - at least one B and one X; or
     - at least one 0 and one X; or
     - at least one 0 and one A; or
     - at least one A and one B.
   - Its contents, when represented in standard data format, are any combination of characters in the OS/3 system character set.
   - The maximum number of character positions allowed is 132.

8. To define an item as numeric edited:

   - Its character-string is restricted to certain combinations of the 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.
   - The maximum number of P's permitted is 17.
   - Its contents, when represented in standard data format, must consist of only the numerals 0 through 9, plus editing symbols indicated.
   - The maximum number of character positions allowed is 132.

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

   S V . CR DB E H

10. An integer enclosed in parentheses following any of the symbols:

    A , X 9 P Z * B 0 + – $

    indicates the number of consecutive occurrences of the symbol.

11. See Table 5–6 for the order of precedence for characters used as symbols in a character-string.

12. See Table 5–7 for examples of source fields and receiving fields.

Table 5–5. PICTURE Symbols (Part 1 of 2)

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 and 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 and 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 $ + or –	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 $ – or +	Can be preceded only by: V . , $ + – P B 0 (zero)
, (comma)	Insert comma in character position unless the preceding position is blank or asterisk-filled	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	May not be last character
B	Insert a blank or space in character position unless previous character is blank or asterisk-filled	Any symbol except S and 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

Table 5–5. PICTURE Symbols (Part 2 of 2)

Picture Symbol	Represents	Can Be Used in Combination With	Special Picture Position
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 – P B 0 (zero). If only one used, it can only be preceded by + – or P or V
0 (zero)	Insert 0 in character position unless previous character is blank or asterisk-filled	Any symbol except S and H	None
+	Insert + in character position if data item value positive and – 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 consecutive + cannot be used with A X – S CR DB Z H * or more than one $ sign	If only one + is used, it must be either first or last (except for P and V, and excepting its use with E where it may be first and also immediately follow the E); if more than one is used, it must be the first symbol except for the $ sign
– (minus)	Insert – in character position if data item value negative and 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 consecutive – cannot be used with: A X + S CR DB * Z H or more than one $ sign	If only one – is used, it must be either first or last (except for P and V, and excepting its use with E where it may be first and also immediately follow the E); if more than one is used, it must be the first symbol except for the $ sign
* H	COMP-3	S P V 9	None
* E	Denotes exponentiation	+ – 9 . V	Between mantissa and exponent of a floating-point numeric display item

*Extension to American National Standard COBOL (1968).

Table 5–6. Precedence Rules in PICTURES

		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 symbols at the top of the column may precede, in a given character-string, the symbols 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 represents their use to the left of the numeric character positions and the second their use to the right of the numeric character positions.

4. Z, *, nonfixed 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 5–7. Source and Receiving Fields

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	1△2△3△45
X(5)	A1B2C	XBX00XXX	A△100B2C
A(5)	ABCDE	ABB0AAA0BX	A△△0BCD0△E
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)	AB△△△
A(5)	ABCDE	AA	AB
99PPP	12	9(5)	12000
VPPP99	12	.9(5)	.00012
V9(5)	12345	Z(5).99	△△△△.12
V9(5)	12345	9(5).999	00000.123

5.3.5. USAGE Clause

Function:

Specifies the format of a data item in main storage.

Format:

```
[USAGE IS] { COMP
           { COMPUTATIONAL
           { COMP-1*
           { COMPUTATIONAL-1*
           { COMP-2*
           { COMPUTATIONAL-2*
           { COMP-3*
           { COMPUTATIONAL-3*
           { COMP-4*
           { COMPUTATIONAL-4*
           { 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 USAGE IS DISPLAY option specifies that the item is stored in character form, one character per byte; it is used for alphabetic, alphanumeric, alphanumeric edited, numeric edited, decimal numeric display, and floating-point numeric display.

   The compiler performs the necessary conversions when decimal numeric display items or floating-point numeric display items are used for computations; for instance, the latter items are converted to their equivalent floating-point values in the form of the number used in arithmetic operations.

3. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value corresponding 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 input-output 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-3 specifies packed decimal format, where:

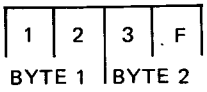
   - If the number of digits in the item is odd, the object program main storage area allocated for this item is an even number of half bytes.

**Extension to American National Standard COBOL (1968).*

Example:

PIC 999 VALUE 123 USAGE COMP-3.

Main Storage:

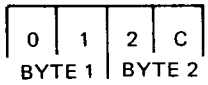


If the number of digits in an item is even, an extra half byte is in the object program main storage allocated for this item. The item's PICTURE is unchanged.

Example:

PIC S99 VALUE 12 USAGE COMP-3.

Main Storage:



The compiler ensures that the unused half byte is always set to 0 when information is stored in this item. The compiler assumes that when the item is referenced it contains a valid packed decimal number, with 0 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.

8. COMP and COMP-4 specify the binary format. The amount of storage allocated depends on the number of digits in the PICTURE:

Number of 9's in the PICTURE	Storage Allocated
1 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes

COMPUTATIONAL items are logically equivalent to signed COMP-3 items except for their internal representation and storage allocation.

In general, COMP-3 items may be manipulated more efficiently than COMP or COMP-4 items. COMP or COMP-4 items are more efficient that COMP-3 items when used as subscripts. ←

9. COMP-1 and COMP-2 specify the floating-point computational format. COMP-1 specifies a single precision floating-point item. A COMP-1 item occupies 4 bytes. COMP-2 specifies a double precision floating-point item which occupies 8 bytes.

In the procedure division, a floating-point item is disallowed wherever an integral value is necessary.

5.3.6. SYNCHRONIZED Clause

Function:

When computations are performed in COBOL, the data items involved must be aligned before the operations can be performed. Normally the compiled object program aligns the items automatically before performing computations, but the user may align items requiring arithmetic operations in the data division by using the SYNCHRONIZED clause. This results in the use of more storage for the program, but the execution times for arithmetic operations may be greatly reduced.

Format:

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

Rules:

1. The optional keywords LEFT and RIGHT are included in the format for compatibility only.

2. The SYNC clause operates by adding slack bytes to records containing items to be synchronized. Slack bytes are unused bytes inserted preceding each synchronized item in the record and padding the record so that the synchronized item appears on the proper boundary. The boundary used depends on the format of the item as it is defined by the USAGE clause:

Item	Length of Item	Alignment Boundary
{COMP / COMP-4}	One to four 9's in the PICTURE	Half word
	Five to eighteen 9's in the PICTURE	Full word
COMP-1		Full word
COMP-2		Double word

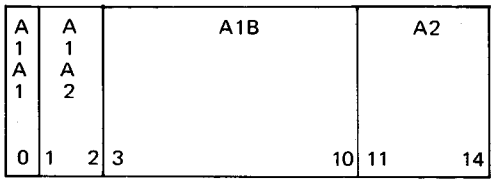
For DISPLAY and COMP-3 items, the SYNC clause has no effect.

The SYNCHRONIZED clause may appear on either an elementary item or a level-01 item. If used on a level-01 item, the SYNCHRONIZED clause applies to every elementary item within the level-01 item. The SYNCHRONIZED clause does not affect the length of elementary items.

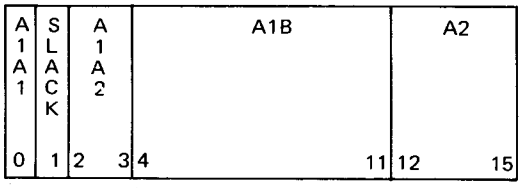
Assume a record is described as:

```
01  A.
02  A1.
03  A1A.
04  A1A1 PIC X.
04  A1A2 PIC S9 USAGE COMP.
03  A1B USAGE COMP-2.
02  A2 USAGE COMP-1.
```

Without using the SYNC clause, this results in storage assigned as follows:

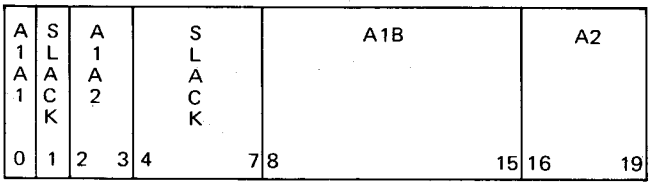


If the fifth line is changed to specify that A1A2 is to be synchronized (04 A1A2 PIC S9 USAGE COMP SYNC.), one slack byte is inserted to align A1A2 on a half-word boundary:



The slack byte is inserted in position 1; in essence, it is a 04 level item and is included in the length of A1A. Only this one slack byte was necessary to achieve the necessary alignment.

The SYNC clause could be specified for the entire record (01 A SYNC.). In this case, every elementary item is affected, yielding storage assignments as follows:



In the example, A1A1 need not be aligned because it is not a computational item. A1A2 is a COMP item and is aligned on a half-word boundary by the insertion of one slack byte. A1B is a COMP-2 item and requires alignment on a double-word boundary; this is provided by the insertion of four slack bytes. A2 fell on a double-word boundary and since it required only a full-word boundary, no slack bytes were needed.

The algorithm used by the compiler to determine the insertion of slack bytes is:

- As each item to be synchronized is encountered, the total number of bytes occupied by all the elementary items up to but not including this one is added to the total number of slack bytes already inserted.

- This total divided by x, where:

x	Item	Length
2	COMP	1 to 4 digits
4	COMP	5 to 18 digits
4	COMP-1	
8	COMP-2	

- If there is no remainder for the division, no slack bytes are necessary. If there is a remainder, the number of slack bytes required is equal to x minus the remainder.

For the last example, the algorithm would be used as follows:

- For the first synchronized item, A1A2, the total number of bytes in the record so far is 1; x for this COMP item is 2; the remainder of the division is 1. Thus, x (2) minus 1 equals 1; therefore, 1 is the number of slack bytes required.

- For A1B, a COMP-2 item, the storage already occupied is 1 (for A1A1) + 1 (the first slack byte) + 2 (for A1A2), a total of 4. The value of x to be used is 8, and the remainder of the division is 4; therefore, x (8) minus 4 equals 4, so four slack bytes were inserted in positions 4 through 7 to align A1B.

- When A2 is encountered, the total storage already occupied is 16; when this is divided by 4, the value of x for A2, there is no remainder. No slack bytes were required.

3. Should synchronized items be specified for a record which contains an OCCURS clause, slack bytes are inserted as described in rule 2, and slack bytes may also be inserted between group items within the record to ensure the alignment of each occurrence of the group.

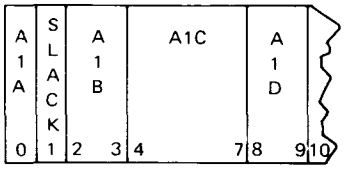
NOTE:

Even without the SYNCHRONIZED clause, if the first occurrence of an item resides on its natural machine boundary, the compiler adds any slack bytes necessary to ensure alignment of each occurrence.

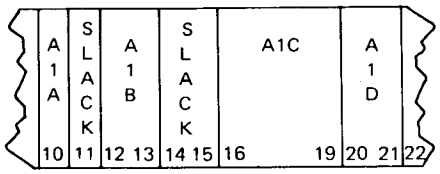
For example:

```
01  A SYNC.

02  A1 OCCURS 3.

03  A1A PIC X.

03  A1B PIC S9 USAGE COMP.

03  A1C USAGE COMP-1.

03  A1D PIC S9 USAGE COMP.
```

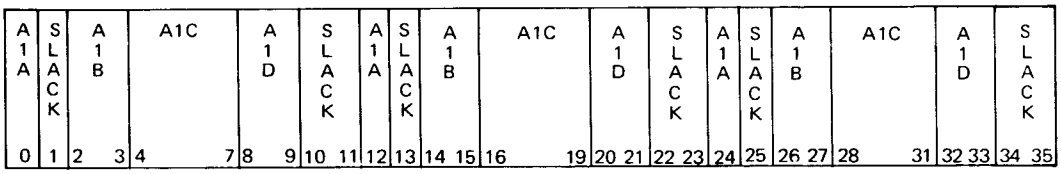
One occurrence would be synchronized as:



If the second occurrence began immediately with byte 10, slack bytes in the second occurrence would have to be:



because A1C must be aligned on a full-word boundary. Of course, the group cannot have different lengths with each occurrence. Therefore, slack bytes are inserted between occurrences so that each occurrence has the same length and the proper alignment of elementary items. The actual storage use for the example is:



The slack bytes in positions 10 and 11, and in positions 22 and 23, were inserted between groups. The algorithm used is:

- The total number of bytes occupied by the group, including slack bytes, is divided by the largest value of x necessary in the group.
- If there is no remainder, no slack bytes are inserted between groups. Otherwise, the number of slack bytes necessary is equal to x minus the remainder.

For the example given, the process would be:

- The total number of bytes occupied in one occurrence of the group is 10 bytes. This is divided by 4, the x value for A1C, a COMP-1 item.
- The remainder of the division is 2; x (4) minus 2 equals 2, so the number of slack bytes necessary for each occurrence of the record is 2.

4. The compiler aligns all 01 level entries on double-word boundaries.

The algorithm to be used by the programmer in these cases is:

- Add the lengths of all elementary data items and slack bytes in this record. Add 4 more to the total if variable-length records are used.
- Divide the total by the largest value of x required for an item in the record.
- If there is no remainder, no slack bytes are required. If there is a remainder, x minus the remainder is the number of slack bytes which must be inserted. This can be accomplished by including a 02 level FILLER defined as the correct length.

5. If the SYNCHRONIZED clause appears on an item with a REDEFINES clause, the user must ensure that the item being redefined is properly aligned for the data item that REDEFINES it. No slack bytes are added.

   Likewise, if synchronization is necessary for the first elementary item under an item with a REDEFINES clause, no slack bytes are added.

5.3.7. JUSTIFIED Clause

Function:

Specifies nonstandard positioning of data within a receiving data item.

Format:

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

Rules:

1. The JUSTIFIED clause may be specified only at the elementary item level.
2. This clause may not be used for numeric or numeric-edited data, because 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 items 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 Clause

Function:

Defines the initial value of a working-storage item, or specifies the value associated with a condition-name.

Format 1:

VALUE IS literal

Format 2:

$$\left\{\begin{matrix}\underline{\text{VALUES}}\ \text{ARE} \\ \underline{\text{VALUE}}\ \text{IS}\end{matrix}\right\}\ \text{literal-1}\ [\underline{\text{THRU}}\ \text{literal-2}]\ \ldots\ [\ ,\ \text{literal-3}\ [\underline{\text{THRU}}\ \text{literal-4}]\ ]\ \ldots$$

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:

   - This format 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.
   - 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.
   - In the file section and the linkage section, the VALUE clause must not be used except for condition-name entries.
   - 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.
   - 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 except for condition-names entries.
   - The VALUE clause must not be specified for a group item containing items with descriptions including JUST, SYNC, any COMP usage, or USAGE INDEX.
   - The VALUE clause is not permitted for floating-point display entries.
   - 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. The following rules apply to format 2:

   - All condition-name entries are level-number 88. See 5.3.12 for a full description of condition-name.
   - When the THRU keyword 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 clauses stated for condition-name entries are 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.

6. Floating-point numeric literals may be used only on COMP-1 and COMP-2 operands.

5.3.9. BLANK WHEN ZERO Clause

Function:

Sets the value of a receiving item to space when the value of the sending item is 0.

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 0.

5.3.10. MAP* Clause

Function:

Specifies the size of a data item in bytes in main storage.

Format:

MAP IS integer-3, CHARACTERS

Rule:

The MAP clause does not affect the object program in OS/3 COBOL; however, it is acceptable to the compiler for compatibility purposes.

5.3.11. RENAMES Clause

Function:

Permits alternate, possibly overlapping, groupings of elementary items.

Format:

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

**Extension to American National Standard COBOL (1968).*

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 Clause

Function:

Assigns a name for a specific value or range of values.

Format:

88 condition-name; { **VALUES ARE** / **VALUE IS** } **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:

   - 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.
```

- 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* Clause

Function:

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

Format 1:

```
[SIGN IS] { LEADING  } SEPARATE CHARACTER
          { TRAILING }
```

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, excepting floating-point display.

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

4. If format 1 is used, the character 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 character 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.

**Extension to American National Standard COBOL (1968).*

5.4. WORKING-STORAGE SECTION

Definition:

That section of the data division used to describe areas of main storage that 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-level-description-entry]
[record-description-entry] ...

[88 (condition-name entry)]

5.4.1. Independent Entries

Function:

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 [optional clauses]

Rules:

1. Level-numer 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 Entry

Function:

Describes 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 bearing a definite relationship to each other may be grouped into records 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 because 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 available in a calling program, but 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 level-number 88 entries.

2. Record description entries in the linkage section provide names and descriptions, but storage within the program is not reserved because the data exists elsewhere.

3. The linkage section is required in any program containing an ENTRY statement with a USING option or the procedure division USING option for a called program.

4. See 6.8 for examples of calling and called programs.

```
SEQUENCE   CONTINUATION
NUMBER      A      B   TEXT
1      6  7 8   11 12         20            30            40
002001      DATA   DIVISION.
002002      FILE   SECTION.
002003      FD     CDS
002004             LABEL RECORDS ARE OMITTED
002005             DATA RECORD IS CARDIN.
002006      01     CARDIN.
002007             03 CRDLOC     PIC X(6).
002008             03 CRDID      PIC X.
002009             03 CRDCITY    PIC X(30).
002010             03 CRDNAME    PIC X(25).
002011             03 CRDPHONE   PIC X(17).
002012             03 CRDCODE    PIC X.

002015      FD     OUTPUT1
002016             LABEL RECORDS ARE OMITTED
```

Figure 5–1. Example of Data Division Entries (Part 1 of 2)

**Extension to American National Standard COBOL (1968).*

CONTINUATION

SEQUENCE NUMBER 1–6	7	A 8–11	B 12–40 TEXT
002017			RECORDING MODE IS F
002018			BLOCK CONTAINS 10 RECORDS
002019			DATA RECORD IS DISC01.
002020		01	DISC01
002021			03 LOCNO PIC X(6.).
002022			03 ID PIC X.
002023			03 CITY PIC X(30).
002024			03 CNAME PIC X(25).
002025			03 PHONE PIC X(17).
002026			03 CODE PIC X.

Figure 5–1. Example of Data Division Entries (Part 2 of 2)

6. Procedure Division

6.1. GENERAL

The procedure division in the SPERRY UNIVAC Operating System/3 (OS/3) COBOL program contains the instructions or steps necessary to solve a given problem.

Format:

```
PROCEDURE DIVISION [USING unqualified-data-name-1 [unqualified-data-name-2] . . .].
[DECLARATIVES.
{section-name SECTION. declarative-sentence.
{paragraph-name. { sentence} . . . }. . . }. . .
 END DECLARATIVES.]
{[section-name SECTION [priority-number] .]
{paragraph-name. { sentence } . . . }. . . }. . .
```

6.1.1. USING* Statement

Function:

When the USING statement immediately follows the heading PROCEDURE DIVISION, it serves as an entry point declaration and can appear only if this program is a called subprogram.

Format:

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

Rules:

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

2. If the USING option 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.

**Extension to American National Standard COBOL (1968).*

6.2. DECLARATIVES SECTION

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 SECTION. declarative-sentence.
{ paragraph-name. {sentence} ...}...}...
END DECLARATIVES.
```

Rules:

1. Declarative sections are grouped at the beginning of the procedure division.
2. The keyword DECLARATIVES must immediately follow the division header PROCEDURE DIVISION on a separate line. The keywords 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 statement. The remainder of the section consists of one or more procedural paragraphs.
4. No priority number is allowed on section-names in the declaratives section.
5. See 6.6.7.4, USE statement.

6.3. SECTION

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. {sentence}...}...
```

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. 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.
4. Priority-numbers 0 through 49 are used for the fixed and the fixed overlayable segments, and priority numbers 50 through 99 designate independent segments. (See 6.7 for a complete discussion of segmentation.)

6.4. PARAGRAPH

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**} ...} ...

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:

Those statements which are neither compiler-directing statements nor conditional statements (including conditional-causing arithmetic or input-output statements), which indicate a specific action to be taken by the object program.

Format:

verb word-string.

Rules:

1. The verb must be one of those listed in 6.6, excluding the compiler-directing and conditional verbs and those input-output or arithmetic verbs for which the statement specifies one of the conditional options AT END, SIZE ERROR, or INVALID KEY (6.5.2).

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

Conditional statements specify that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent upon this truth value.

A conditional statement is:

- an IF statement, a SEARCH statement, or a PERFORM statement with the UNTIL option;
- an input/output verb that specifies an INVALID KEY or an AT END option; or
- an arithmetic verb that specifies an ON SIZE option.

6.5.3. Compiler-Directing Statements

Definition:

Statements directing 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 verbs:

   COPY, ENTER, NOTE, USE

2. A word-string consists of 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.

6.5.4. Overlapping Operands

When a sending and a receiving item in an arithmetic statement or in an EXAMINE, MOVE, or TRANSFORM statement share portions of their storage areas, the results are undefined when these statements are executed.

6.6. VERB TYPES

A verb is a reserved word, used in the procedure division, denoting an action to be performed by the processor or the compiler. There are eight general categories of verbs in OS/3 COBOL. These categories, and the verbs in each, are:

- 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
- Interprogram Communication: CALL, ENTRY

A description of the 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. The COMPUTE verb is provided to allow the programmer to specify arithmetic calculations through the use of arithmetic expressions.

Rules:

1. All data items referenced in arithmetic statements must represent numeric elementary data items previously defined in the data division. A data item following the word GIVING, or a receiving identifier of a COMPUTE verb, may be a numeric edited item.

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

3. Except for floating-point items, 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 unless the receiving data item is defined as floating point.

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 also is automatic.

5. 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 1 whenever the most significant digit of the excess is equal to or greater than 5. If the resultant identifer is a floating-point item, the ROUNDED option is meaningless.

6. If, after decimal-point alignment, the value of the 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 occurs, depending on whether the ON SIZE ERROR option has been specified:

   - If ON SIZE ERROR is not specified, and a size error condition arises, the effect is unpredictable.
   - If the ON SIZE ERROR option 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 is executed after the last resultant identifier is considered.
   - An ON SIZE ERROR option is meaningless for a floating-point receiver, except in a DIVIDE statement where the size error imperative statement is executed only on an attempt to divide by zero.

7. 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 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.
   - Both of the data items are elementary numeric items.
   - Neither d_1 nor d_2 can be a data item with level-number 66, 77, or 88.
   - 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.

8. Statements with multiple results are considered by the compiler as though they were written:

   - as a statement that 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; or
   - 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.

6.6.1.1. ADD Statement

Function:

The ADD statement adds two or more numeric operands and stores the result.

Format 1:

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

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

Format 2:

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

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

Format 3:

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

[; ON SIZE ERROR imperative-statement]
```

Rules:

1. In formats 1 and 2, each identifier must refer to an elementary numeric item, except identifiers to the right of the word GIVING, which may be numeric edited items.

2. Each literal must be a numeric literal.

3. If floating-point operands are not used, 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, and 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; 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 6.6.1, rules 5, 6, and 7.

6.6.1.2. DIVIDE Statement

Function:

The DIVIDE statement divides one numeric data item into another and sets the value of a data item equal to the results; identifier-1 may be either dividend or divisor, depending on whether INTO or BY is specified.

Format 1:

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED] [ ; ON SIZE ERROR imperative-statement]
       {literal     }
```

Format 2:

```
DIVIDE {identifier-1} INTO {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }      {literal-2   }
[ ; ON SIZE ERROR imperative-statement]
```

Format 3:

```
DIVIDE {identifier-1} BY {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }    {literal-2   }
[ ; ON SIZE ERROR imperative statement]
```

Format 4:

```
DIVIDE {identifier-1} INTO {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }      {literal-2   }
REMAINDER identifier-4 [ ; ON SIZE ERROR imperative-statement]
```

Format 5:

```
DIVIDE {identifier-1} BY {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }    {literal-2   }
REMAINDER identifier-4 [ ; ON SIZE ERROR imperative statement]
```

Rules:

1. Each identifier must refer to a numeric elementary item, except identifiers immediately 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. 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. The rule does not apply if any of the operands are floating-point items.
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 rules 5 and 6 in 6.6.1.

7. 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. When the REMAINDER option is specified, none of the operands may be floating-point.

6.6.1.3. MULTIPLY Statement

Function:

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

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 containing 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 identifer-1 or literal-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, except for floating point, is 18 decimal digits.

6. For a description of the ROUNDED and SIZE ERROR options, see rules 5 and 6 in 6.6.1.

6.6.1.4. SUBTRACT Statement

Function:

The SUBTRACT statement subtracts one or the sum or two or more numeric data items from one or more items, and sets the value of one or more items equal to the results.

Format 1:

```
SUBTRACT { identifier-1 } [, identifier-2] ...
         { literal-1    }  [, literal-2   ]

    FROM identifier-m [ROUNDED] [,identifier-n [ROUNDED]] ...

    [ ; ON SIZE ERROR imperative statement]
```

Format 2:

```
SUBTRACT { identifier-1 } [, identifier-2] ...
         { literal-1    }  [, literal-2   ]

    FROM { identifier-m } GIVING identifier-n [ROUNDED]
         { literal-m    }

    [ ; ON SIZE ERROR imperative-statement]
```

Format 3:

```
SUBTRACT { CORR          } identifier-1
         { CORRESPONDING }

    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 the total is subtracted from identifier-m, identifier-n, etc. The result of the subtraction is stored as the new value in identifier-m, identifier-n, etc.

2. Except for floating point, 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 item that follows the word GIVING, aligned on their decimal points, must not contain more than 18 digits unless floating-point operands are used.

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, the total is subtracted from literal-m or identifier-m, and 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 rules 5, 6, and 7 in 6.6.1.

6.6.1.5. COMPUTE Statement

Function:

The COMPUTE statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression.

Format:

COMPUTE identifier-1 [ROUNDED] = { arithmetic-expression / identifier-2 / literal }

[; ON SIZE ERROR imperative-statement]

Rules:

1. Literal must be a numeric literal.

2. Each identifier must refer to an elementary numeric item, except for identifier-1, which may be a numeric edited item.

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, except floating-point operands, 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 identifer-1.

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

8. Intermediate results are possible in a COMPUTE statement containing two or more 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 to division by zero and to final results. See Appendix C for a discussion of how the compiler handles intermediate results.

   - Arithmetic operators character representation:

Addition	+
Subtraction	–
Multiplication	*
Division	/
Exponentiation	**
Unary plus and minus	+, –

- 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.

- 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 –

6.6.2. 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

6.6.2.1. ALTER Statement

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 with a priority equal to or greater than 50 must not be referred to by an ALTER statement in a section with a different priority.

6.6.2.2. GO TO Statement

Function:

The GO TO statement transfers control from one part of the procedure division to another. GO TO (format 3) is used as a special exit from a USE procedure.

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 fixed-point 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. 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 0, control is passed to the sentence following this statement.
8. The maximum number of procedure-names allowed in format 2 is 64; the minimum is two.
9. Format 3 transfers control from a USE procedure to the I/O control system and is an extension to American National Standard COBOL (1968). The following rules apply to the GO TO MORE-LABELS option:
   - Format 3 can appear only within a label-processing section in the declarative section.
   - 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 procedure division.

- 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.

6.6.2.3. PERFORM Statement

Function:

This verb permits a temporary departure from the normal sequence of execution to execute one or more procedures, either a specified number of times or until a specified condition is satisfied, after which control is automatically returned to the normal sequence.

Format 1:

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

Format 2:

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

Format 3:

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

Format 4:

```
PERFORM procedure-name-1 [THRU procedure-name-2]
    VARYING { identifier-1   } FROM { identifier-2   }
            { index-name-1   }      { index-name-2   }
                                    { literal-2      }
    BY { identifier-3 } UNTIL condition-1
       { literal-3    }
   [AFTER { identifier-4   } FROM { identifier-5   }
          { index-name-4   }      { index-name-5   }
                                  { literal-5      }
    BY { identifier-6 } UNTIL condition-2
       { literal-6    }
    [AFTER { identifier-7   } FROM { identifier-8   }
           { index-name-7   }      { index-name-8   }
                                   { literal-8      }
    BY { identifier-9 } UNTIL condition-3]]
       { literal-9    }
```

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, the identifier represents a numeric item with no positions to the right of the assumed decimal point; a floating-point operand is not permitted in format 2.

3. Each literal represents a numeric literal.

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

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

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

   - If procedure-name-2 is specified and is a:

     – paragraph-name, return occurs after execution of the last statement of the paragraph,

     – section-name, return occurs after execution of 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, 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 other than a PERFORM statement, control passes through the last statement of the procedure to the following statement, regardless of use of the EXIT statement.

6. Format 1 is the basic PERFORM statement. A procedure referred to by this type of PERFORM statement 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 then is 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 0, control passes immediately to the statement following the PERFORM statement. Once the PERFORM statement is initiated, any redefinition of 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, the specified procedure is not executed and 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, references to identifier as the object of VARYING and FROM phrases also refer to index-name.

10. When one identifier is varied:

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

    - 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.
    - Condition-1 is evaluated. If true, control is passed to the statement following the PERFORM statement. If false, condition-2 is evaluated.
    - 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.
    - When condition-2 is true, identifier-4 is set to initial value (identifier-5), identifier-1 is changed by identifier-3, and condition-1 is reevaluated.
    - The PERFORM statement is completed when condition-1 is true; if false, the cycle continues until condition-1 is true.
    - Figure 6–1 illustrates the logic of the PERFORM statement when two identifiers are varied. 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.

12. When three identifiers are varied:

    - 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.
    - Figure 6–2 illustrates the logic of the PERFORM statement when three identifiers are varied. 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:

    - sections with priority numbers of less than 50; and
    - sections entirely contained in a single segment with a priority number greater than 49.

14. A PERFORM statement that appears in a section which has a priority number equal to or greater than the SEGMENT-LIMIT can have, within its range, only the following:

    - sections with the same priority number as the section containing the PERFORM statement; and
    - 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.

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 statement must itself be either totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM statement. Thus, an active PERFORM statement, the execution of which 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.

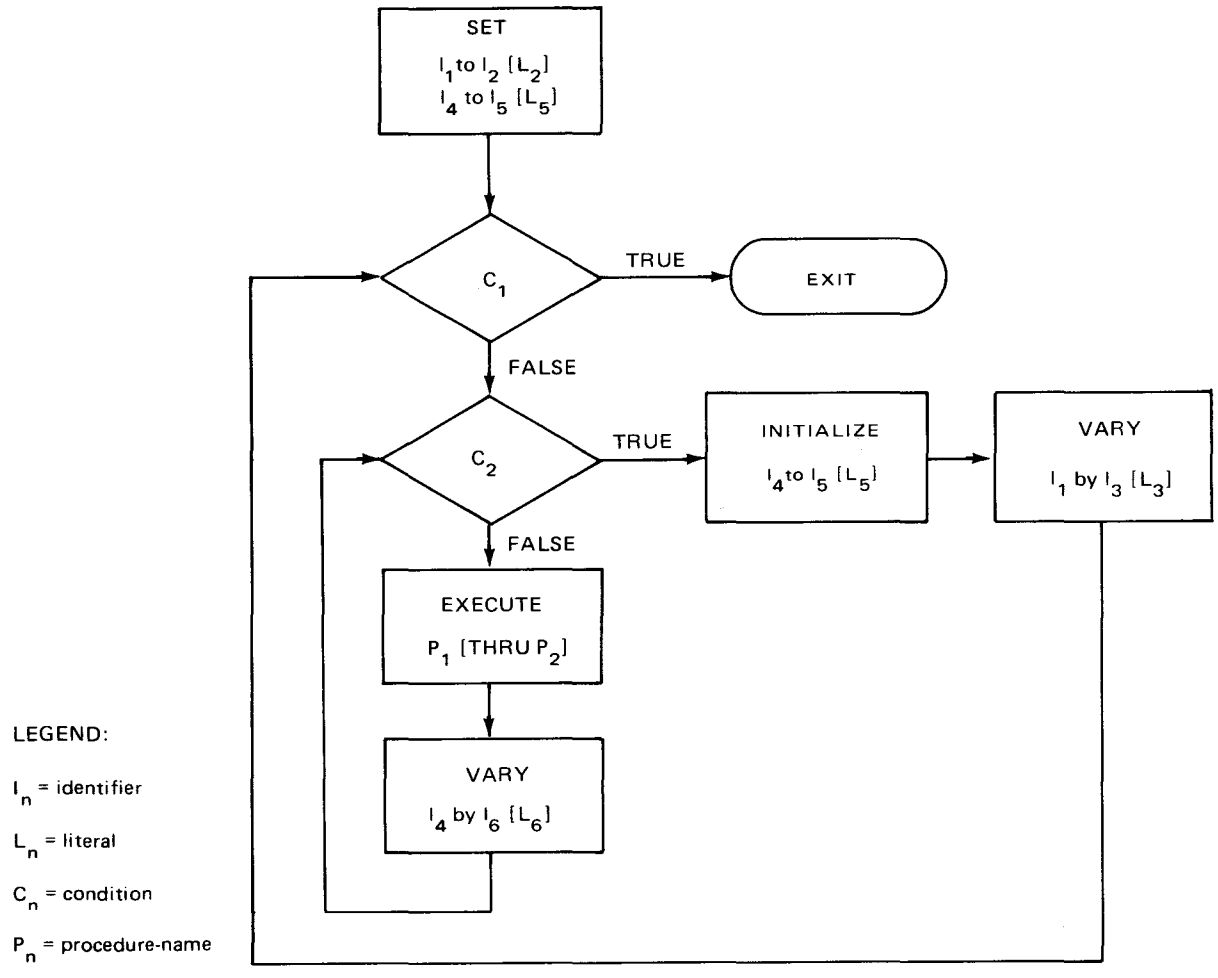
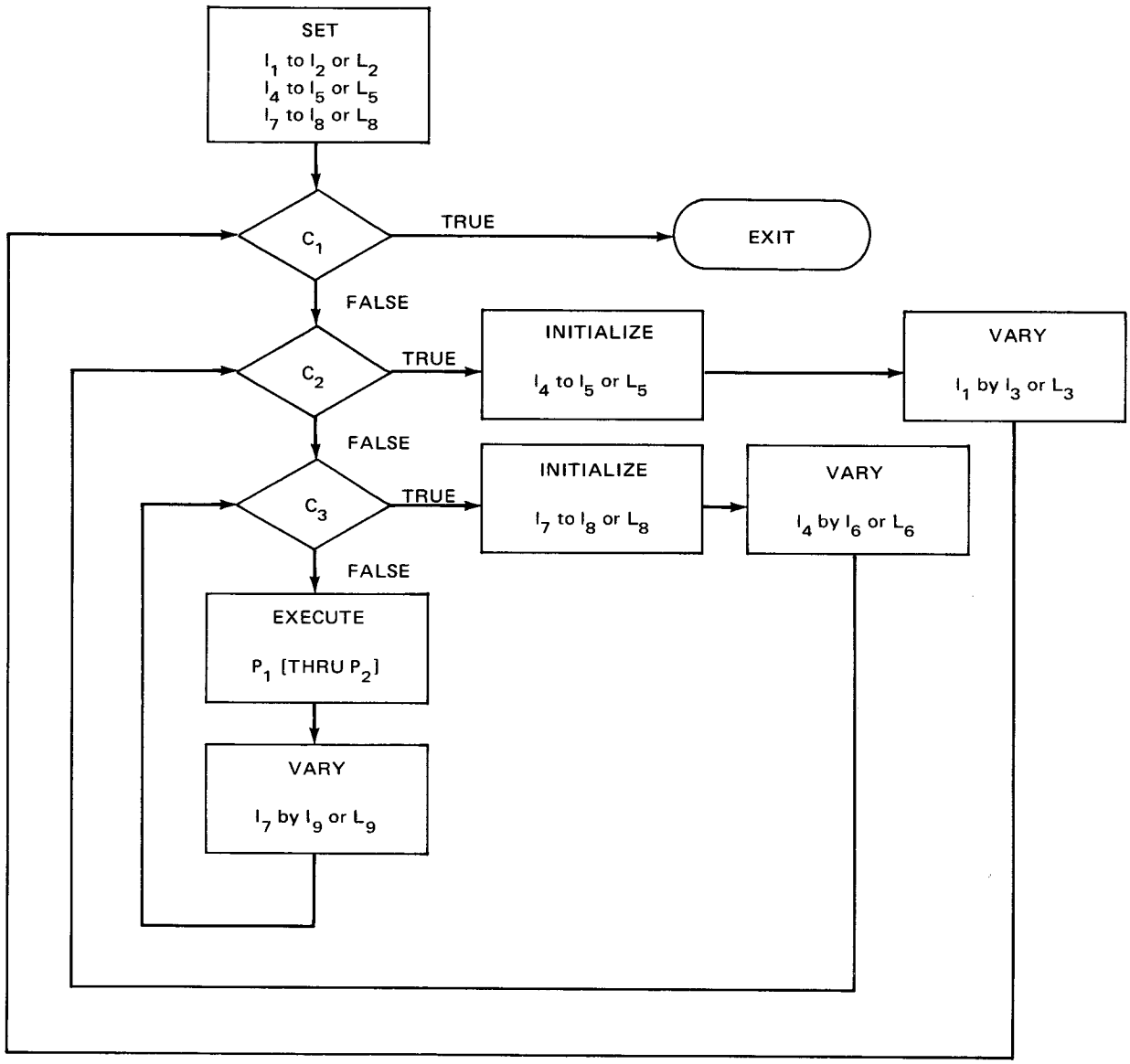


Figure 6–1. PERFORM Logic: Varying Two Identifiers



Legend:

I_n = identifier

L_n = literal

C_n = condition

P_n = procedure-name

Figure 6–2. PERFORM Logic: Varying Three Identifiers

6.6.2.4. EXIT* Statement

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 statement 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.

NOTE:

For examples of called or calling programs, see 6.8.

6.6.3. Data Movement Verbs

Four verbs are provided by OS/3 COBOL for the specific purpose of moving or manipulating data:

EXAMINE, MOVE, SET, TRANSFORM

These are verbs in addition to the several verbs which, as a secondary function, 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.

**Extension to American National Standard COBOL (1968).*

6.6.3.1. EXAMINE Statement

Function:

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

Format:

```
EXAMINE identifier { TALLYING { ALL      }                                  }
                   {           { LEADING  } literal-1 [REPLACING BY literal-2] }
                   {           { UNTIL FIRST }                              }
                   { REPLACING { ALL      }                                  }
                   {           { LEADING  } literal-3 BY literal-4           }
                   {           { [UNTIL] FIRST }                            }
```

Rules:

1. The description of the identifier must be such that USAGE IS DISPLAY (explicitly or implicitly). Floating-point display identifiers are examined as if they were nonnumeric.

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:

   - Nonnumeric examination starts at the leftmost character and proceeds to the right; each character is examined individually.

   - 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:

   - ALL option: the number of occurrences of literal-1.

   - LEADING option: the number of occurrences of literal-1 prior to encountering a character other than literal-1.

   - 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:

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

   - 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.

   - UNTIL FIRST option: the substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 is encountered.

   - FIRST option: the first occurrence of literal-3 is replaced by literal-4.

6.6.3.2. MOVE Statement

Function:

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

Format 1:

```
MOVE { identifier-1 } TO identifier-2[, identifier-3] . . .
     { literal-1    }
```

Format 2:

```
MOVE { CORR          } identifier-1 TO identifier-2
     { CORRESPONDING }
```

Rules:

1. If the CORRESPONDING option is used, selected items within identifier-1 are moved to selected items within identifier-2 according to rule 7 in 6.6.1, except that identifiers need not be numeric and may be either both elementary items, or 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 6–1 shows legal categories of sending and receiving fields.

Table 6–1. MOVE 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 floating-point item or a numeric item with an implicit decimal point not immediately to the right of the least significant digit must not be moved to an alphanumeric or alphanumeric edited data item.

4. The following rules apply to legal elementary moves:

   - When the receiving field is alphanumeric edited, alphanumeric, or alphabetic, justification and any necessary space filling takes place as defined under the JUSTIFIED option. 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.

   - When the receiving field is a numeric or numeric edited item, alignment by decimal point and any necessary zero filling 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.

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

   - When the sending field is an edited item, it is treated as an alphanumeric item.

   - 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 no conversion of data from one form of internal representation to another occurs.

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

6.6.3.3. SET Statement

Function:

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

Format 1:

```
SET { identifier-1      } [ , identifier-2      ] ... TO { identifier-3      }
    { index-data-item-1 }   { , index-data-item-2 }      { index-data-item-3 }
    { index-name-1      }   { , index-name-2      ]      { index-name-3      }
                                                         { literal-1         }
```

Format 2:

```
SET index-name-1 [, index-name-2] ... { DOWN BY } { identifier-1 }
                                      { UP BY   } { literal-1    }
```

Rules:

1. All identifiers must be either index data items or numeric elementary items described without any positions to the right of the assumed decimal point, i.e., no floating-point, except that identifier-1 in format 2 must not be an index-data-item.

2. All literals must be positive integers; floating-point literals are not permitted.

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 explain the allowable combinations of choices in the SET statement.

   - SET index-name-1 TO index-name-3

     The occurrence number value of index-name-3 computes a new displacement value for index-name-1. Also, the occurrence number value of index-name-3 replaces that of index-name-1. If the length of one occurrence is the same for both, no computation is necessary.

   - SET index-name TO index-data-item

     Same as SET index-name-1 TO index-name-2, 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.

   - SET index-name { DOWN BY } { identifier }
                    { TO   } { literal   }
                    { UP BY   }

     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, increments, or decrements the occurrence number value of index-name.

- SET index-data-item-1 TO { index-data-item-2 / index-name }

  A 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.

- 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

8 bytes

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)

6.6.3.4. TRANSFORM* Statement

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.

Format 1:

TRANSFORM identifier-3[, identifier-4] . . . CHARACTERS

FROM { figurative-constant-1 / identifier-1 / nonnumeric-literal-1 } TO { figurative-constant-2 / identifier-2 / nonnumeric-literal-2 }

Format 2:

TRANSFORM identifier-3[, identifier-4] . . . CHARACTERS

FROM { ASCII TO EBCDIC / EBCDIC TO ASCII }

*Extension to American National Standard COBOL (1968).

Format 3:

TRANSFORM identifier-3[, identifier-4] . . . CHARACTERS

$$\left\{\begin{matrix}\underline{\text{BY}}\\ \underline{\text{ON}}\end{matrix}\right\}$$ **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, decimal numeric display item without a SEPARATE SIGN clause is treated as a single character, not as 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 equal the length of identifier-1, or identifier-2 must have a length of 1 character.

4. In format 1, all figurative constants are permitted except ALL.

5. In format 1, a character must not be repeated in identifier-1 or in nonnumeric-literal-1.

6. In format 3, identifier-5 must be a length of 256 characters.

7. The following paragraphs and Table 6–2 explain the allowable combinations of choices in the TRANSFORM statement.

   - The following rules apply to these combinations in format 1:

     identifier-1 TO identifier-2

     identifier-1 TO nonnumeric-literal-2

     identifier-1 TO figurative-constant-2

     nonnumeric-literal-1 TO identifier-2

     nonnumeric-literal-1 TO nonnumeric-literal-2

     nonnumeric-literal-1 TO figurative-constant-2

     – If the FROM and the TO operands are 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 (or the single character) in the corresponding position 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.

- 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 must be one character. Any occurrence in identifier-3 of the single character in operand-1 is replaced by the single character in operand-2.

- 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.

- The following rules apply to format 3:

  — Identifier-3 may be described as having 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.

Table 6—2. Combination of FROM and TO Options in a TRANSFORM Statement (Part 1 of 2)

Operands	Rule	Identifier-3 Before	FROM	TO	Identifier-3 After
FROM figurative-constant-1 TO figurative-constant-2	All occurrences of figurative-constant-1 in the item represented by identifier-3 are replaced by figurative-constant-2. (Each operand must be a single character.)	1"2""3	QUOTE	ZERO	102003
FROM figurative-constant-1 TO nonnumeric-literal-2	All occurrences of figurative-constant-1 in the item represented by identifier-3 are replaced by nonnumeric-literal-2. (Each operand must be a single character.)	1Δ2Δ3	SPACE	"7"	17273
FROM figurative-constant-1 TO identifier-2	All occurrences of figurative-constant-1 in the item represented by identifier-3 are replaced by the item represented by identifier-2. (Each operand must be single character.)	1Δ2Δ3	SPACE	ALPHA (current value of ALPHA = B)	1B2B3
FROM nonnumeric-literal-1 TO figurative-constant-2	All occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-3 are replaced by the single-character figurative-constant-2.	AB12X7P	"1234567890"	SPACE	AB ΔΔ X ΔP
FROM nunnumeric-literal-1 TO nonnumeric-literal-2	Nonnumeric-literal-1 and nonnumeric-literal-2 must be equal in length, or nonnumeric-literal-2 must be a single character. If the operands are equal in length, any character in the item represented by identifier-3 that is equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of nonnumeric-literal-2.	ABCD12X	"ABCDEFGHIJ"	"1234567890"	123412X
	If nonnumeric-literal-2 is a single character, then all occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-3 are replaced by the single character in nonnumeric-literal-2.	AB21X73	"1234567890"	"L"	ABLLXLL

Table 6–2. Combination of FROM and TO Options in a TRANSFORM Statement (Part 2 of 2)

Operands	Rule	Identifier-3 Before	FROM	TO	Identifier-3 After
FROM nonnumeric-literal-1 TO identifier-2	The two operands must be equal in length, or identifier-2 must represent a single-character item. If the operands are equal in length, any character in the item represented by identifier-3 that is equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of the item represented by identifier-2. If identifier-2 is a single character, then all occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-3 are replaced by the character represented by identifier-2.	1Δ2ΔDEF ABC	"Δ12DEF" ADE	BETA (current value of BETA = FED21Δ.) GAMMA (current value of GAMMA = 1)	EFDF21Δ 1BC
FROM identifier-1 TO figurative-constant-2	All occurrences of any character of the item represented by identifier-1 in identifier-3 are replaced by the single character figurative-constant-2.	A12B	GAMMA (current value of GAMMA = ABC.)	QUOTE	"12"
FROM identifier-1 TO nonnumeric-literal-2	The two operands must be equal in length, or nonnumeric-literal-1 must be a single-character item. If the operands are equal in length, any character in the item represented by identifier-3 that is equal to a character in the item represented by identifier-1 is replaced by the character in the corresponding position of nonnumeric-literal-2. If nonnumeric-literal-2 is a single character, then all occurrences of any character of the item represented by identifier-1 in the item represented by identifier-3 are replaced by nonnumeric-literal-2.	ABCD ABCD	ALPHA (current value of ALPHA = A12B) DELTA (current value of DELTA = ABCDEF)	"DCBA" "6"	DACD 6666
FROM identifier-1 TO identifier-2	Any character in the item represented by identifier-3 that is equal to a character in the item represented by identifier-1 is replaced by the character in the corresponding position of the item represented by identifier-2. Both operands must be of equal length. Each of the operands may contain one or more characters.	1AB4	ITEM-A (current value of item-A = 1234.)	ITEM-B (current value of ITEM-B = ABCD.)	AABD

6.6.4. Input/Output Verbs

In any data processing application, quantities of data are 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	READ	RETURN
CLOSE	WRITE	REWRITE
DISPLAY	INSERT	SEEK
OPEN	RELEASE	SORT

6.6.4.1. ACCEPT Statement

Function:

Reads low volume data from an appropriate hardware device, system main storage location, or UPSI (user program switch indicator) byte.

Format:

ACCEPT identifier [**FROM** { **mnemonic-name** / **DATE*** / **DAY*** / **TIME*** }]

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. Data is moved, left-justified.

2. The job control stream is assumed to be the input source when the FROM option is not specified. The description of identifier determines the number of cards accepted. One card from the job control 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 system console, the following message is displayed:

   CA10 ACCEPT READY

   Program operation is suspended until a type-in occurs (CA10 indicates a COBOL ACCEPT). The maximum number of characters that can be transmitted from the system console for a single ACCEPT is 60.

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

   SYSIN

   SYSIN-96

   SYSIN-128

   SYSSWCH

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

5. The identifier must be defined implicitly or explicitly as USAGE IS DISPLAY.

6. The DATE and DAY options make the current date available in the formats yymmdd and yyddd, respectively. The TIME option makes the current time of day available in the format hhmmss00.

NOTE:

The use of ACCEPT statements is illustrated in Section 9.

**Extension to American National Standard COBOL (1968).*

6.6.4.2. CLOSE Statement

Function:

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

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 references are made to the file.

3. The REEL/UNIT option effects reel or unit swapping in a sequential file process. When specified, it terminates the current reel or unit of a multivolume file. Processing continues with the next reel or unit of the file. Unless early termination of the current reel or unit is desired, the REEL/UNIT phrase is unnecessary because swapping occurs automatically at the end of the current reel or unit. If the reel/unit is to be dismounted from the device, the LOCK option should be used. After execution of a CLOSE statement with a REEL/UNIT option, the file is still open.

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

5. The REEL, NO REWIND, and LOCK options are applicable only 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 used without a REEL option, the file is closed and the current volume is rewound and unloaded. As a result, the file cannot be reopened without operator intervention.

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.

6.6.4.3. DISPLAY Statement

Function:

The DISPLAY statement writes low volume data to an appropriate hardware device or system main storage location. It can also be used to set the UPSI switches. (See Section 9 for a detailed explanation of DISPLAY statement usage.)

Format:

$$\underline{\text{DISPLAY}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{literal-1} \end{matrix} \right\} \left[\begin{matrix} \text{, identifier-2} \\ \text{, literal-2} \end{matrix} \right] \ldots [\underline{\text{UPON}} \text{ mnemonic-name}]$$

Rules:

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

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. The special-names that may be associated with the DISPLAY statement via mnemonic-name are:

   SYSCOM

   SYSCONSOLE

   SYSLOG

   SYSSWCH

   SYSSWCH-n

   SYSLST

   See Table 4–1 for more detailed information.

4. If the system console is the hardware device, the sum of the sizes of operands in a DISPLAY statement may not exceed 250 characters. The data is displayed on the system console a line at a time. Each line is preceded by CD10Δ (CD11Δ if SYSLOG is used), followed by 55 characters of the contents of the operands.

5. COMP-3 numeric items and binary items are converted to DISPLAY decimal. For signed numeric items, a separate sign character is displayed immediately following the operand. Floating-point computational items are converted to floating-point display items.

6. The number of printer characters displayed is a multiple of 120. 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.

NOTE:

The use of DISPLAY statements is illustrated in Section 9.

6.6.4.4. OPEN Statement

Function:

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

Format:

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

Rules:

1. At least one of the options INPUT, OUTPUT, or I-O must be specified. They 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.

   The REVERSED option requires that the file be positioned at its end prior to the execution of the OPEN statement. The NO REWIND option requires that the file be positioned at its beginning prior to the execution of the OPEN statement.
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 is executed if one was specified by a USE statement (6.6.7.4).

6.6.4.5. READ Statement

Function:

For sequential file processing, the READ statement makes available the next logical record from a file and allows performance of a specified imperative-statement when end of file is detected.

For random file processing of mass storage files, the READ statement makes available a specific record from a file, 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        } imperative-statement
                                        { ; INVALID KEY   }
```

Rules:

1. An OPEN statement (INPUT or I-O) 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. The AT END or INVALID KEY clause is required for all file organizations except indexed organization, where its use is optional. The execution of the imperative statement AT END or INVALID KEY is dependent upon file organization and file usage. See Section 11 for detailed information on these conditions.

4. If an input file described with the keyword OPTIONAL is not present, the imperative-statement in the AT END option is executed on the first READ statement.

5. The READ statement performs the functions of the SEEK statement implicitly for random access files.

6. 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. Reading INTO is performed according to the rules of a group MOVE (6.6.3.2).

7. 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 or after the AT END condition is detected.

6.6.4.6. WRITE Statement

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.

Format 1:

```
WRITE record-name [FROM identifier-1]
[ { AFTER  } ADVANCING { identifier-2 LINES }  ]
[ { BEFORE }           { integer LINES      }  ]
                       { mnemonic-name      }
```

Format 2:

```
WRITE record-name [FROM identifier-1] ; INVALID KEY imperative-statement
```

Rules:

1. A file must be opened (OUTPUT or I/O) 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 a group MOVE.

4. After the WRITE statement 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.

5. The INVALID KEY clause in format 2 is used when processing direct access files and is required for RELATIVE file organization; for SEQUENTIAL and INDEXED organizations its use is optional. The conditions that cause execution of the INVALID KEY imperative statement depend upon file organization and file usage. For more detailed information, see Section 11.

6. The ADVANCING option controls the vertical positioning of each record on the printed page. If this option is omitted for a printer file, the printer automatically advances one line before printing (i.e., WRITE record-name AFTER ADVANCING 1 LINE). Any form of the ADVANCING option overrides this automatic advance.

   - The identifier represents a numeric item with no positions to the right of the assumed decimal point; a floating-point operand is not permitted.

   - The contents of identifier-2 or the value of integer must not exceed 127. A value of 0 is permissible (where overprinting is desired).

   - 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 (4.2.3).

7. The USE FOR FORM-OVERFLOW clause in the declaratives section 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 paper tape loop is specified (ADVANCING mnemonic-name), the form overflow condition does not occur.

6.6.4.7. INSERT* Statement

Function:

The INSERT statement adds a logical record to indexed organization 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 or extended and organization is indexed.
2. A file must be opened (I-O) 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 a group MOVE.
5. After the INSERT statement is executed, information in record-name is no longer available, but identifier-1 information is available.
6. The INVALID KEY clause is required for all file organizations except indexed organization, where its use is optional. See Section 11 for detailed information on these conditions.

6.6.4.8. REWRITE* Statement

Function:

The REWRITE statement releases a logical record for an output file for the purpose of updating an existing record.

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 (I-O) and a record read 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.

**Extension to American National Standard COBOL (1968).*

3. When the FROM option is used, data is moved from identifier-1 to record-name according to the rules specified for a group MOVE.

4. After the REWRITE statement is executed, information in record-name is no longer available, but identifier-1 information is available.

5. The INVALID KEY imperative statement in format 2 is used when processing direct access files. See Section 11 for detailed information on these conditions.

   The INVALID KEY clause is required for all file organizations except indexed organization, where its use is optional.

6.6.4.9. SEEK Statement

Function:

The SEEK statement initiates access of a mass storage data record for subsequent reading or writing.

Format:

SEEK file-name RECORD

Rules:

1. A SEEK statement pertains only to the disc files specified in the following chart.

Organization Type	Access Method	SEEK Allowed
Sequential	Sequential	No
Relative	Sequential	Yes
	Random	Yes
Indexed	Sequential	Yes
	Random	No
	Extended	Yes

2. The value of the identifier in the ACTUAL or RELATIVE key clause is used by the SEEK statement to determine the location of the record to be accessed when ORGANIZATION is RELATIVE. When ORGANIZATION is INDEXED, the value of the identifier in the 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 statement is negated by the execution of a second SEEK statement.

6.6.4.10. RELEASE Statement

Function:

The RELEASE statement transfers records to the initial phase of a sort operation.

Format:

RELEASE record-name [FROM identifier]

Rules:

1. Record-name must be part of a logical record described in the associated sort file description (SD).

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 the storage area associated with identifier are moved to the storage area associated with record-name; the contents of the record-name area are released to the sort-file. Moving takes place according to the rules specified for a group MOVE. The information in the record-name area is no longer available, but the information in the data area associated with identifier is available.

5. A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement for file-name.

6.6.4.11. RETURN Statement

Function:

The RETURN statement 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 be only 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 of a group MOVE.

6. After execution of the AT END phrase, no RETURN statements may be executed within the current output procedure.

6.6.4.12. SORT Statement

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 one or more output procedures or to an output file.

Format:

```
SORT file-name-1 ON { ASCENDING  } KEY { data-name-1 } ...
                    { DESCENDING }

    [ ; ON { ASCENDING  } KEY { data-name-2 } ... ] ...
           { DESCENDING }

    { 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 (5.2.2).

2. Each data-name must represent data items described in records associated with file-name-1. Nonnumeric key items must not exceed 256 characters. Floating-point display items are considered alphanumeric. 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 specified for the sort file. The size of the logical records described for file-name-2 and file-name-3 must be equal to the size of the logical records 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. More than one SORT statement may appear in the procedure division of a program, but none may appear in the declarative section, nor in the input and output procedures associated with another SORT statement. The use of the SORT feature is discussed in detail in Section 12.

6. Input procedure

   - This procedure must consist of one or more consecutive sections that do not form a part of any output procedure.
   - This procedure must include at least one RELEASE statement in order to transfer records to the sort file.
   - 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.
   - The input procedure may include any procedures needed to select, create, or modify records.
   - The input procedure must not contain a SORT statement.

7. Output procedure

   - This procedure must consist of one or more consecutive sections that do not form a part of any input procedure.
   - This procedure must include at least one RETURN statement in order to make sorted records available for processing.
   - The output procedure must not contain a SORT statement.
   - 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.
   - The output procedure may 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.

8. ALTER, GO TO, and PERFORM statements are not permitted to refer to procedure-names outside the input and output procedures. ALTER, GO TO, and PERFORM statements in the remainder of the procedure division must not refer to procedure-names within the input and output procedures.

9. When the ASCENDING option is used, the sorted sequence is from lowest value of key to highest value according to the character collating sequence shown in Appendix A. The sorted sequence is reversed when the DESCENDING option 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. Floating-point display keys are considered alphanumeric.

10. Every record description for the 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. Key items must not exceed 256 characters. When variable-length records are used, the key items must be within the length of the shortest record.

11. 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.

12. If the USING option is specified, all the records in file-name-2 are transferred to file-name-1. The SORT statement automatically performs the functions of the OPEN, READ, USE, and CLOSE statements for file-name-2. File-name-2 must be a sequential access file.

13. 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.

14. 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.

6.6.5. Ending Verb (STOP)

This statement is used to halt execution of the object program either permanently or temporarily, with or without a display of a literal.

Format:

STOP { **literal** / **RUN** }

Rules:

1. The literal may be numeric or nonnumeric, fixed- or floating-point, or any figurative constant except ALL.

2. The literal is communicated to the operator through the system console, and continuation of the program begins with 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'.
   ```

   causes the literal to be displayed as follows:

   ```
   CD10 CARDS OUT OF SEQUENCE, CORRECT SEQUENCE, REPLACE CARDS IN READER,
   ANSWER R WHEN READY.
   ```

   This is 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 last statement in a sequence of imperative statements.

6.6.6. Conditional Verbs

Conditional expressions are used in situations in which the outcome of a test determines the next logical step to be performed. The verb IF is the principal conditional verb used with conditional expressions.

The conditional verbs are IF and SEARCH.

6.6.6.1. IF Statement

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

Rules:

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

2. The ELSE NEXT SENTENCE option may be omitted if it immediately precedes the terminal period of the sentence.

3. Execution of an IF statement takes the following action:

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

   - 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 statement encountered is considered to apply to the immediately preceding IF statement that has not been already paired with an ELSE statement. The maximum number of IF statements that may be nested is 30 in OS/3 COBOL.

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 true value of a test. Five types of conditions are possible:

   - Relation condition

   - Class condition

   - Condition-name condition

**Extension to American National Standard COBOL (1968).*

- Switch-status condition
- Sign condition

These conditions are discussed in rules 7 through 11.

The logical operators used in combination with these conditions are:

OR

AND

NOT

Table 6–3 indicates the relationship between the logical operators and conditions A and B.

Table 6–3. Logical Operator/Condition Relationships

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

The ways in which conditions and logical operators may be combined are shown in Table 6–4.

Table 6–4. Logical Operator/Condition Combinations

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

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:

```
{arithmetic-expression-1}
{identifier-1           }  relational-operator  {arithmetic-expression-2}
{literal-1              }                      {identifier-2           }
                                               {literal-2              }
```

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 are:

```
                                  {[NOT] GREATER THAN}
                                  {[NOT] >           }
                                  {[NOT] LESS THAN   }
IF {arithmetic-expression-1} IS   {[NOT] <           }  {arithmetic-expression-2}
   {identifier-1           }      {[NOT] EQUAL TO    }  {identifier-2           }
   {literal-1              }      {[NOT] =           }  {literal-2              }
                                  EQUALS*
                                  UNEQUAL*
                                  EXCEEDS*
```

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

- the omission of the subject of the relation condition; or
- 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 were replaced by the last preceding stated subject, or the omitted relational-operator were replaced by the last preceding stated relational-operator.

A logic error with unexpected results may follow the use of NOT with abbreviated relational conditions. Programmers should remember that the compiler interprets NOT as a logical operator in these situations, not as a 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
```

*Extension to American National Standard COBOL (1968).

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

- 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.

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

- 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 is first. The items are equal if all pairs of characters are equal.

    The first pair of unequal characters encountered is compared for relative location in the OS/3 COBOL collating sequence. The operand which contains that character which is positioned higher in the collating sequence is determined to be the greater operand.

  – Operands of unequal size

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

- Index-names and/or index data-items

  – Two index-names

    The result is the same as if the corresponding occurrence numbers were 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] { ALPHABETIC / NUMERIC }

The operand being tested must be described, implicitly or explicitly, as USAGE IS DISPLAY or USAGE IS COMP-3.

- Numeric test

  Identifier can be described as alphanumeric or numeric with usage COMP-3 or DISPLAY, but not as floating-point display. 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. A condition-name may be associated with a range 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, rule 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 value of an 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 { arithmetic-expression / identifier } IS [NOT] { NEGATIVE / POSITIVE / ZERO }

6.6.6.2. SEARCH Statement

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.

Format 1:

```
SEARCH identifier-1 [ VARYING { identifier-2  } ]
                              { index-name-1  }

    [; 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 identifies the table to be searched and must not be subscripted or indexed. 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 option 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. It may not be a floating-point item.

3. In format 1, condition-1, condition-2, etc., may be any condition as described in 6.6.6.1.

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 statement is used, a serial type of search operation takes place, starting with the current index setting.

   - 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 option is specified, imperative-statement-1 is executed; if the AT END option is not specified, control passes to the next sentence.

- 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 in which 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 is satisfied, 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. 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 statement, 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 statement 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 the range of the table, 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, the 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 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 option 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.

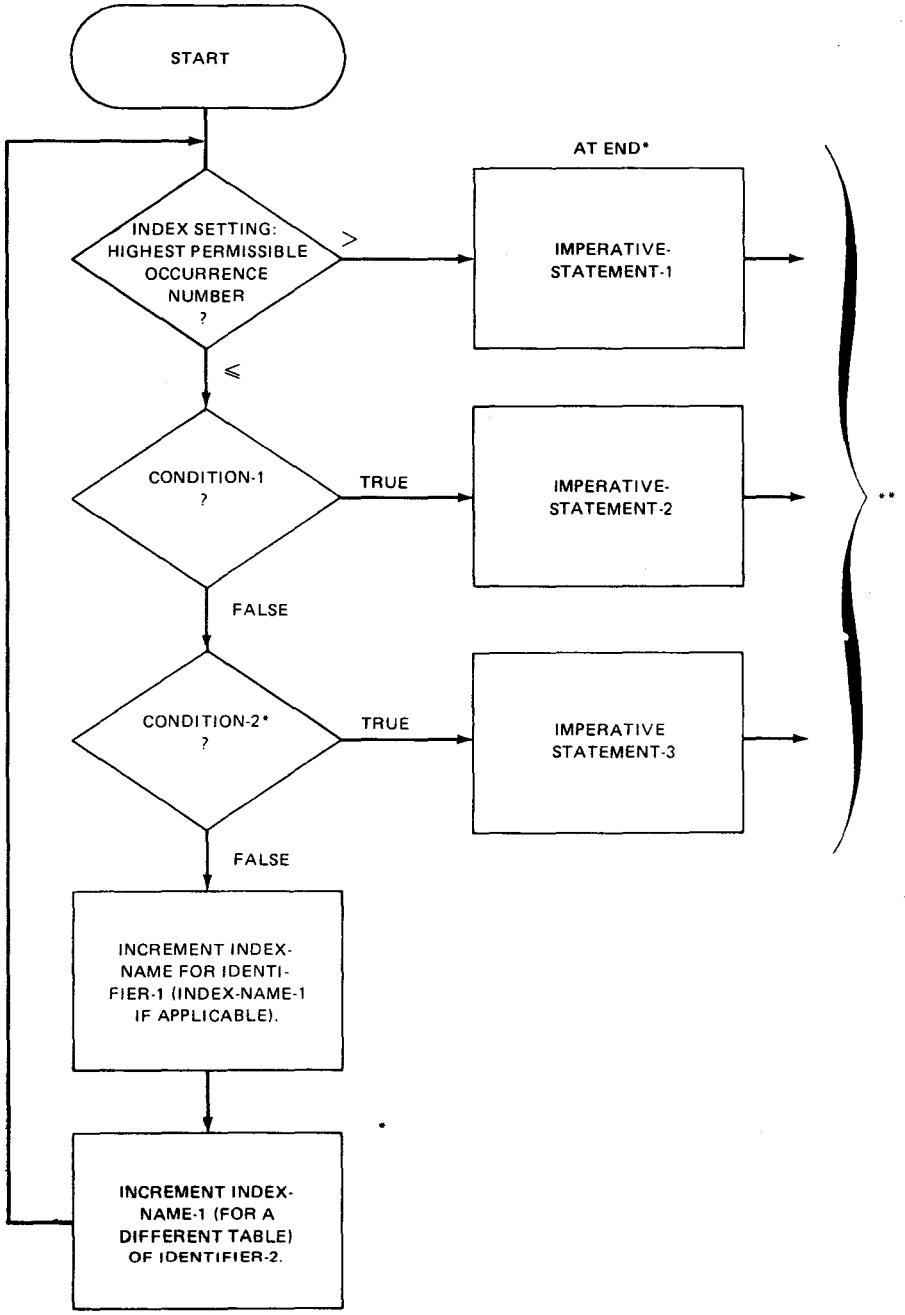
11. In format 1, if the VARYING index-name-1 option is specified, and if index-name-1 appears in the INDEXED BY phrase of identifier-1, that index-name is used for this search. If this is not the case, or if the VARYING identifier-1 option is specified, the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used for the search. In addition, the following operations will occur:

    - If the VARYING index-name-1 option 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.

    - If the VARYING identifier-2 option 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 2- or 3-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 2- or 3-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.

A diagram of the Format 1 SEARCH operation containing two WHEN phrases is shown in Figure 6–3.



*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

6.6.7. 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

6.6.7.1. COPY Statement

Function:

The COPY statement copies text from the COBOL library into the source program with a capability of word substitution as the text is copied (7.3).

Format 1:

COPY library-name.

Format 2:

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

Rules:

1. The COPY statement may appear anywhere in a COBOL program.

2. The library-name is an element name in the COBOL libraries. It may contain no more than eight characters; the name may be composed of alphanumeric characters and the hyphen, but it must contain at least one alphabetic character.

3. The remainder of the line on which a COPY statement is terminated must be blank. In other words, nothing may follow a COPY statement on the same source program line.

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 appear in the output listing, unless printing of the library text is suppressed through use of the LST=I option on the COBOL compiler PARAM statement (7.1.1).

6. The text contained in the library must not contain any COPY statements.

7. If the REPLACING option 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 option.

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 literals may be numeric or nonnumeric, fixed or floating-point, or any figurative constant except ALL.

6.6.7.2. ENTER* Statement

Function:

The ENTER statement, in conjunction with the CALL or ENTRY statements, permits run-time communications between the main COBOL program and previously compiled subprograms in OS/3 COBOL or other languages. ENTRY also may be used with the EXIT PROGRAM or RETURN options.

Format 1:

```
ENTER LINKAGE.
CALL entry-name [USING { file-name
                       { identifier
                       { procedure-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.
```

Extension to American National Standard COBOL (1968).

Rules:

1. Format 1 transfers control from one object program to another within the run unit.

   - Entry-name must be the external symbol of an entry point in the subprogram being called. Entry-name may be a nonnumeric literal.
   - Each of the identifiers in the USING clause of the CALL statement must be a reference to any level data item except an 88 level in the file, working-storage, or linkage sections of the calling program.
   - Procedure-name, file-name, and sort-name can be used only if the called subprogram is written in a language other than COBOL.
   - 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 heading), 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), the entry-name in format 1 must be the same as the entry-name in format 2.
   - 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, in the called subprogram, designates an ENTRY point; it may not appear in the declaratives portion.

   - If the calling program is written in OS/3 COBOL, entry-name in format 2 must be the same as entry-name in format 1.
   - Data-name can be neither qualified nor subscripted.
   - Data-names are the names of 01- or 77-level data items specified in the linkage section of this particular subprogram.
   - The sequence of appearance of the operands in the two USING clauses is extremely significant because 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 USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the calling program.
   - An entry name may be enclosed in quotation marks.

3. Format 3, in the called subprogram, returns control to the calling program.

   - All OS/3 COBOL subprograms must contain this clause.
   - Control returns to the point in the calling program immediately following the CALL statement.
   - The EXIT PROGRAM and RETURN options are equivalent. RETURN is included for compatibility with other COBOL implementations.

4. See 6.8 for sample calling and called programs.

6.6.7.3. NOTE Statement

Function:

The NOTE statement allows programmers to write commentary to be produced in the listing but not be compiled.

Format:

NOTE character-string.

Rules:

1. Any combination of the characters from the 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 a part of the character-string, whereas a comment line is not (Table 2–3).
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.

6.6.7.4. USE Statement

Function:

The USE statement specifies procedures for input/output label and error handling in addition to the standard procedures specified by the input/output system. Three format options are available:

- Label writing and checking
- Error checking
- Printer form-overflow

Format 3 is an extension to American National Standard COBOL (1968).

Format 1:

```
USE { AFTER  } STANDARD [ BEGINNING ] [ FILE  ]
    { BEFORE }          [ ENDING    ] [ REEL  ]
                                      [ UNIT  ]
    LABEL PROCEDURE ON { file-name-1 [,file-name-2] . . . }
                       { INPUT                             }
                       { I-O                               }
                       { OUTPUT                            }
```

Format 2:

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name-1 [,file-name-2] . . . }
                                      { INPUT                             }
                                      { I-O                               }
                                      { OUTPUT                            }
```

Format 3:

USE FOR FORM-OVERFLOW ON file-name-1

Rules:

1. A USE statement must immediately follow a section header in the declaratives section 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; the USE statement itself is never executed.

3. When format 1 is used:

   - If the file-name option is present, the FD entry for file-name-1 must contain a LABEL RECORDS ARE data-name clause (5.2.1.3).

   - If the BEGINNING or ENDING options are omitted, the designated procedures are executed for both beginning and ending labels. The ENDING option is not applicable for direct access files whose organization is other than sequential.

   - If the REEL or UNIT option 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 the FILE, UNIT, or REEL option 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.

   - 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.

   - The BEFORE option is not applicable to the OS/3 COBOL but is accepted for compatibility. The BEFORE option is processed as if AFTER were specified.

   - For files opened for input, the designated USE procedure is executed only when a user label is encountered. This label can be checked by referencing the record defined by the LABEL RECORD clause in the FD. If further labels exist, they can be accessed by issuing a GO TO MORE LABELS verb. User label processing is terminated upon execution of the last statement in the USE procedure.

   - For files opened for output, the designated USE procedure is executed after system label processing is completed. A user label is written from the record area defined by the LABEL RECORD clause after execution of the last statement in the USE procedure. A label is also written upon execution of a GO TO MORE LABELS verb and control is then transferred to the beginning of the same USE procedure.

4. When format 2 is used, the USE procedure is initiated when system standard I/O error recovery procedures are exhausted. After a format 2 USE procedure is executed, no attempt should be made to access the file in error.

5. When format 3 is used, control is transferred to the USE procedure when a printer carriage overflow condition is detected. See data management user guide, UP-8068 (current version).

   Overflow is detected during the print and space functions of the printer. If form positioning by ADVANCING mnemonic-name is specified, a form-overflow condition does not occur.

6. File-name must not represent a sort file in any format.

7. Input/output statements or the STOP verb with the literal option are not allowed inside USE procedures except for the following verbs:

   - ACCEPT (not from jobstream or system console)
   - DISPLAY
   - WRITE to a printer within a FORM-OVERFLOW procedure

   NOTE:

   At least one DISPLAY to SYSLST must be performed in the nondeclarative portion of the procedure division before any are performed with the declarative section. Accepts from the job control stream are not permitted inside a USE statement for LABEL PROCEDURE.

8. ENTRY statements are not allowed within USE procedures.

9. In a USE procedure, there must be no reference to any nondeclarative procedures. Conversely, in the nondeclarative 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 in formats 1 or 2, or to the procedures associated with such a USE declarative.

10. See 6.2, declaratives section.

6.6.8. Interprogram Communications

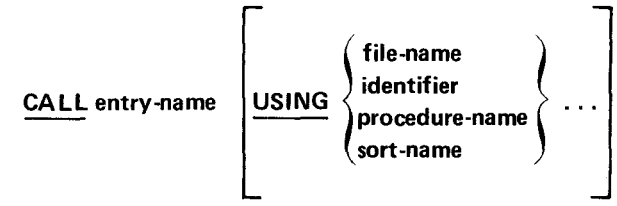
Communications between an OS/3 COBOL program (caller) and either an OS/3 COBOL or another language program (called) are established by the CALL verb. An entry point in the called program is established by the ENTRY verb.

6.6.8.1. CALL* Statement

Function:

In conjunction with the ENTER verb in the main program, communicates with subprogram entry points.

Format:



Rule:

See the ENTER verb, 6.6.7.2, for information regarding use of the CALL statement.

**Extension to American National Standard COBOL (1968).*

6.6.8.2. ENTRY* Statement

Function:

The ENTRY statement, in conjunction with the ENTER statement in a called program, establishes an entry point.

Format:

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

Rule:

See 6.6.7.2.

6.7. SEGMENTATION

Segmentation is a method of communication with the compiler to specify object program overlay requirements. Since OS/3 COBOL deals just with segmentation of procedures, only the procedure division is considered in determining segmentation requirements for an object program.

6.7.1. Program Segments

When segmentation is used, it is mandatory that the procedure division be written in sections. Each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation does not negate the need to qualify procedure-names to ensure uniqueness.

6.7.1.1. Fixed Portion

The fixed portion is that part of the object program which is logically treated as if it were always in main storage. This portion of the program is composed of two types of segments, the fixed permanent segment and the fixed overlayable segment.

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 (4.2.2). Such a segment, if called for by the program, is always made available in its last used state.

If the SEGMENT-LIMIT clause is not specified, an implicit segment-limit of 50 is in effect.

6.7.1.2. Independent Segments

An independent segment is a part of the object program that can overlay, and be overlaid by, either a fixed overlayable segment or another independent segment. An independent segment is in its initial state when control is transferred to that segment from a segment with a different priority-number.

* *Extension to American National Standard COBOL (1968).*

6.7.2. SECTION

Definition:

Segments are classified by priority numbers included in the section header.

Format:

section-name SECTION [priority number].
{paragraph-name. {sentence}... }...

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. 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-numbers 50 through 99 constitute independent segments. Sections with the same priority-number need not be grouped together in the source program.

6.7.3. Restrictions

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statements.

6.7.3.1. 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 be altered only by an ALTER statement located in the same segment as the GO TO statement.

6.7.3.2. PERFORM Statement

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:

- Sections with a priority less than 50.
- Sections entirely contained in a single segment having a priority-number greater than 49.

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:

- Sections with the same priority-number as that containing the PERFORM statement.
- Sections with a priority-number less than the implicit or explicit SEGMENT-LIMIT priority-number.

6.7.3.3. Linkage Editor Considerations

When linking a segmented COBOL program, the linkage editor control stream must have a LOADM control statement followed by an INCLUDE statement for the root section of the program (fixed-permanent segment). The module-name parameter on the INCLUDE statement must be padded on the right with zeros for a total of eight characters.

6.8. CALLING AND CALLED PROGRAMS

Run-time communication between a main OS/3 COBOL program and any other separately compiled or assembled subprogram is accomplished by the ENTER statement and its associated statements:

- CALL
- ENTRY
- EXIT PROGRAM or RETURN
- USING clause with PROCEDURE DIVISION heading

Actual transfer of control from a calling program to a called program is effected via a CALL statement with an entry-name identical with the entry-name in the ENTRY statement of the called program. Return of control to the calling program is effected by execution of an EXIT PROGRAM statement in the called program. Control is returned to the statement following the CALL statement in the calling program.

A called program need not be an OS/3 COBOL program. In such cases, the COBOL calling program may include procedure-names in its USING argument list.

For a description of register usage requirements, see the CALL, SAVE, and RETURN macro instructions in the supervisor programmer reference, UP-8241 (current version).

6.8.1. Treatment of Data Items

Data items declared in the calling program and referenced in the called program are described in the file or working-storage sections in the data division of the calling program. In the called program, the data items are described, once again, but in the linkage section. Items described in the linkage section are not allocated main 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 use 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 to be shared with the called program. The USING clause in the called program can either follow the PROCEDURE DIVISION heading 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 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.

A called program also may be a calling program sharing common data items in its data division (including linkage section items) with still another called program.

6.8.2. Linking

A sample linker job stream for calling and called programs is:

```
/$
     LOADM     CALLXX
     INCLUDE   CALLER00
     INCLUDE   CALLED00
     INCLUDE   ADDROUT
/*
```

When an object module created by the COBOL compiler is included in a load module, it must be referred to in the INCLUDE statement by the 8-character program name assigned by the compiler. The first six characters contain the program name specified in the identification division of the source program; the last two characters, decimal numbers from 00 to 99, indicate the segment number of the object module within the COBOL program. (All single segment programs are numbered 00.) If the program name specified in the source program is less then six characters, the compiler pads it with zeros before appending it with the 2-digit segment number.

6.8.3. OS/3 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 (Figure 6–4), which shares data-items and calls upon a COBOL subprogram, CALLED (Figure 6–5), and an assembly language subprogram, ADDROUT (Figure 6–6), for operations upon the shared data-items. Table 6–5 shows the relationship between these programs.

For more detailed information concerning the linking of subprograms, refer to the system service programs user guide, UP-8062 (current version).

```
LINE NO.        SOURCE STATEMENT
00001           IDENTIFICATION DIVISION.
00002           PROGRAM-ID.  CALLER.
00003           ENVIRONMENT DIVISION.
00004           CONFIGURATION SECTION.
00005           SOURCE-COMPUTER.  UNIVAC-9030.
00006           OBJECT-COMPUTER.  UNIVAC-9030.
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 6–4. 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-9030.
00006          OBJECT-COMPUTER.  UNIVAC-9030.
00007          DATA DIVISION.
00008          LINKAGE SECTION.
00009          77 DATA1      PIC 9999.
00010          77 DATA2      PIC 99.
00011          01  DATAX.
00012              02  DATA3  PIC 99.
00013              02  DATA4  PIC 99.
00014          PROCEDURE DIVISION.
00015          P0.  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 6–5. Example of Called Program

```
ADDROUT  START 0
         PRINT NOGEN
R1$      EQU   1
R2$      EQU   2
R3$      EQU   3
R4$      EQU   4
RF$      EQU   15
RE$      EQU   14
RC$      EQU   12
RD$      EQU   13
         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
```

Figure 6–6. Example of Called Assembly Subprogram (Part 1 of 2)

```
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$)      LOAD COVER REGS WITH ARG'S
         PACK  HOLD2(2),DATA2ASM(2)
         ZAP   ACCUM(3),HOLD2(2)
         PACK  HOLD2(2),DATA3ASM(2)
         AP    ACCUM(3),HOLD2(2)
         PACK  HOLD2(2),DATA4ASM(2)
         AP    ACCUM(3),HOLD2(2)
         UNPK  DATA1ASM(4),ACCUM(3)
         OI    DATA1ASM+3,X'F0'
         L     RD$,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 6–6. Example of Called Assembly Subprogram (Part 2 of 2)

Table 6–5. Program/Subprogram Relationships

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 results. 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 results 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.

PART 3. COMPILER FEATURES AND CAPABILITIES

7. Compiler Options and Library Statements

7.1. COMPILER OPTIONS

In SPERRY UNIVAC Operating System/3 (OS/3) COBOL, the optional PARAM statement provides a method of presenting parameters to the compiler to exercise specific COBOL options. The format of the PARAM statement is:

//△PARAM△parameters

When PARAM statements are used, they must be positioned immediately following the EXEC job control statement in the compilation job control stream. The PARAM statements are printed on the first page of the compiler output listing.

If a PARAM statement format error or an illegal parameter is encountered, a system console message is produced and the compilation is terminated.

If no PARAM statements are supplied, the compiler produces a source program listing and a source program diagnostic report, and generates an object module.

Only one blank may precede the P of the word PARAM.

Absence of PARAM statements implies:

//△PARAM△LST=(S)

7.1.1. 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.

If the file-name is omitted, the name YSRC is automatically supplied.

In the ambiguity mode, the definition search process is not terminated when the reference is resolved, but continues in an attempt to uncover and report duplicate definitions. When the search of the division that corresponds to the reference type is completed, the other divisions also are 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.

D Produce data division alphabetized cross-reference listing.

E Ignore printer mismatch errors during compilation.

I Inhibit listing of lines included from COPY libraries.

K Inhibit source item sequence number checking (columns 1 through 6 of the source item).

L Single-space all listings requested. If no listings were requested, a single-spaced diagnostic listing is produced.

M Produce data division storage map listing.

N Inhibit all listable output except PARAM statement listing.

O Produce object code listing.

P Produce procedure division storage map listing.

R Allow quote character to be used in a nonnumeric literal bounded by apostrophes.

S Produce source program listing.

T Allow apostrophe character to be used in a nonnumeric literal bounded by quotes.

W Inhibit listing of all precautionary diagnostics. These errors are identified by a severity code of P.

X Produce procedure division alphabetized cross-reference listing.

NOTES:

1. *When LST=(C,M), only the data division storage map has cross-references. When LST=(C,P), only the procedure storage map has cross-references.*

2. *LST=R and T are not allowed within the same program. Use of either option overrides the interchangeability of the apostrophe and quotation mark.*

7.1.2. Output Options

Format:

//△PARAM△OUT=(spec 1,...,spec n)

where:

spec 1,...,spec n
Is one or more of the following:

C Conversion mode.

E Inhibit display of ISAM file status on system console.

K All data items described as USAGE IS COMP or COMPUTATIONAL are treated as packed decimal (COMP-3 or COMPUTATIONAL-3).

L Inhibit generation of linker control statements in object module.

M Produce shared-code COBOL action program to be executed under the control of the information management system (IMS/90).

N Inhibit generation of object module.

P Disregard mismatch errors for all object program print files.

R The word QUOTE is translated as quotation marks.

S Disable object program SORT PARAM card processing. If this parameter is not specified in the compilation, during the execution of an object program SORT atatement, the SORT routine will accept parameters from the job control stream. For a list of SORT/MERGE parameters, refer to the SORT/MERGE user guide and programmer reference, UP-8054 (current version).

T Inhibit compiler generation of a transfer address in the object module. When invoked, the program cannot be executed unless it is called.

V Suppress automatic page overflow in the object program.

7.2. SOURCE AND COPY LIBRARY INPUT SPECIFICATIONS

The following PARAM statements describe the method of reading a source program either from the job control stream or from a disc library.

The formats for the source and copy library PARAM job control statements are presented in the following paragraphs.

- Source library Input

  Format:

  //△PARAM△IN=program-name/file-name

  where:

  program-name
  Is a 1- to 8-character name of source program to be compiled.

  file-name
  Is a 1- to 8-character name used to identify the file on which the source program resides. This name must appear on the LFD job control statement used to define the device to the job control program.

- Copy Library Input

  Format:

  //△PARAM△LIN=file-name

  where:

  file-name
  Is a 1- to 8-character name used to identify the file on which the COPY library resides. This name must appear on the LFD job control statement used to define the device to the job control program.

  If the file-name is omitted, the name COPY$ is automatically supplied.

  The COPY element-name is supplied in the source program via the COPY clause.

7.2.1. Object Module Version/Revision Number

Format 1:

//△PARAM△VER=vv/rr

where:

vv
Is the version number.

rr
Is the revision number.

These numbers are applied to compiler output module.

If the source program is coming from a library and this PARAM statement is not specified, the version number from that library module is used.

If the PARAM statement is not specified and the source program is coming from the job control stream, the version and revision numbers 00/00 are used.

Format 2:

//△PARAM△OBJ=file-name

where:

file-name
Is the file where the object mode generated is to be placed.

If this PARAM statement is omitted, the generated object module is placed in the temporary job run library file (YRUN).

7.2.2. Compiler Source Library Input and Copy Library Input

The source program may be read from the job control stream or a disc library. Any copy library modules referenced by the source program may be read from a disc library. Any library structures to be accessed by the compiler must have been created by the OS/3 librarian.

Any library structures to be accessed by the compiler must be defined in the job control stream, and the LFD names must appear on PARAM statements (keyword IN for the source library; LIN for the copy library). If no copy library modules are referenced by the source program, the copy library need not be defined.

Example:

Source and copy library definitions:

```
//ΔDVCΔ50          //ΔVOLΔdspxxx
//ΔLBLΔfile-id-1   //ΔLFDΔfile-name-1      } Job control statements for source input
//ΔDVCΔ50          //ΔVOLΔdspxxx
//ΔLBLΔfile-id-2   //ΔLFDΔfile-name-2      } Job control statements for copy input
```

with PARAM statements:

```
//ΔPARAMΔIN=program-name/file-name-1
//ΔPARAMΔLIN=file-name-2                   } Source file copy file
```

In the foregoing example, file-name-1 and file-name-2 are programmer-supplied names. File-id-1 and file-id-2 are file-id names used at the time the disc library was created. Program-name is the name of the source library module that contains the source program.

7.3. LIBRARY

The library module specifies text to be copied from the OS/3 COBOL library, which contains text 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. OS/3 COBOL library text is placed in the COBOL library as a function independent of the OS/3 COBOL program.

The following paragraphs contain library information applicable to the OS/3 COBOL user. For a complete discussion of the COBOL library module, see the fundamentals of COBOL — language manual, UP-7503.1 (current version).

7.3.1. Using the COPY Statement

The COBOL library contains text which, through the use of the COPY statement, may be included in a COBOL source program during compilation. The rules for the COPY statement are given in 6.6.7.1.

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 also must be present on a PARAM statement 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. Note that the source listing does not reflect the text change, as replacement is internal. Any reference to a text word which has been replaced causes a diagnostic to be issued.

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 compilation time the source listing would be:

```
 LINE NO. SOURCE STATEMENT
   .   .   .
   .   .   .
   .   .   .
00033      FILE SECTION.
00034      FD FILE01 COPY LIB-FD01 REPLACING DN-1 BY TAX-A.
00035C                LABEL RECORDS ARE STANDARD
00036C                BLOCK CONTAINS 1 RECORD
00037R                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 statements for use with the COPY statement are defined in 7.2.2.

8. RERUN Clause

8.1. GENERAL

The RERUN facility of the SPERRY UNIVAC Operating System/3 (OS/3) provides a means of recording the status and environment of an OS/3 COBOL program at a specified point in the processing of that program. Once recorded, this status and environment may be reestablished and execution of the COBOL program may be resumed from this point. The RERUN facility causes linkage between the COBOL program and the checkpoint facility. The restart ability is provided by the original job control stream with the addition of an RST job control statement placed immediately prior to the JOB job control statement.

8.2. 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 must appear in a SELECT entry. The device specified by external-name is the RERUN receiver, which receives the checkpoint records containing the status and environment of the COBOL program. File-name-1, file-name-2, etc., are RERUN controllers and dictate when the checkpoint 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 through 9,999,999.

8.3. CHECKPOINTING

Checkpoint records are issued whenever integer records occur for a RERUN controller. The RERUN controller record counter is set to 0 when the controller is opened and incremented by 1 before each READ, WRITE, or INSERT statement is issued to the controller. When the RERUN controller is opened as I-O, a WRITE statement 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 is opened automatically with the assumption that label records are standard. 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 are not 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 entry but not in an FD entry.

8.4. RESTARTING

To initiate the restart of a previously checkpointed program, an RST job control statement must immediately precede the JOB job control statement in the original job control stream; the job may then be restarted. The format of the RST job control statement is:

// RST filename, checkpoint-id, number

where:

filename
Is the name of the checkpoint file.

checkpoint-id
Is the checkpoint number identifying the checkpoint to be used to restart the job.

number
Is the job step number within the job to be restarted.

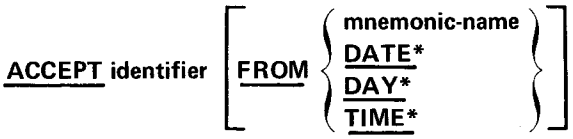
8.5. NOTES AND RESTRICTIONS

- A RERUN controller may have only one RERUN receiver and may appear in only one RERUN clause. If more than one receiver is specified for a RERUN controller, the compiler writes the checkpoint records on the first-mentioned external-name and ignores the second one.
- ACCESS and ORGANIZATION, if specified for a RERUN receiver, must be SEQUENTIAL.
- If the RERUN receiver is a magnetic tape unit, SD must not be specified. If FD is specified, the tape must have standard labels and a block size greater than or equal to 20 bytes.
- ASCII tape files are not permitted.
- The USE declarative statement does not apply to a dedicated RERUN receiver file.
- When errors occur on RERUN receiver files, diagnostic messages are displayed and processing continues. However, no further attempts are made to issue checkpoint records to that receiver.
- Checkpoints issued when a sort is active cannot be used for restarting due to the temporary nature of the sort work-files.

9. Use of ACCEPT and DISPLAY Statements

9.1. ACCEPT STATEMENT

Format:



9.1.1. Job Control Stream ACCEPT

In the SPERRY UNIVAC Operating System/3 (OS/3), COBOL programs are permitted to access their control streams to retrieve PARAM statements and data images.

9.1.1.1. 80-Column Card ACCEPT

An ACCEPT for which the FROM option is not specified or an ACCEPT for which mnemonic-name is associated with SYSIN permits retrieval of data images and PARAM statements from the job control stream. A maximum of 4095 bytes of data may be retrieved with a single ACCEPT statement. The number of bytes accepted is not required to be a multiple of 80. Two ACCEPT statements of 20 character items require two cards.

Job Control Stream Format:

//△EXEC△operand 1, operand 2, operand 3

The EXEC statement (execute) is the last statement processed by job control before the execution of the program (job step) named in the statement. PARAM statements, if any, must directly follow the EXEC 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 is not transferred to the COBOL program.

**Extension to American National Standard COBOL (1968).*

```
DATA IMAGE 1
DATA IMAGE 2
.    .
.    .
.    .
DATA IMAGE n
/*
```

The /* statement indicates the end of a data stream introduced with the job control stream. This statement is required by job control but is not transferred to the OS/3 COBOL program. An attempt to retrieve this statement results in an error condition in the COBOL program.

Job Control Stream Errors:

When the job control stream is unable to deliver an image to the COBOL program (that is, if the next sequential record in the job control stream is not a PARAM statement, or a data image), control is transferred to the object time error subroutine. The subroutine logs the following message on the system console:

CE01 ERROR-DATA FOR ACCEPT NOT AVAILABLE

If the COBOL program attempts to retrieve a /* image from the job control stream, an error condition results. Control is transferred to the object time error subroutine. The subroutine logs the following message on the system console:

CE02 ERROR-INSUFFICIENT DATA FOR ACCEPT

These errors abort the run.

ACCEPTS from the job control stream are not permitted inside a USE for LABEL PROCEDURE.

9.1.1.2. 96-Column Card ACCEPT

An ACCEPT with mnemonic-name associated with SYSIN-96 allows the COBOL program to retrieve embedded data cards from the job control stream when using 96-column cards with data extending beyond column 80. When the job control stream is punched on 96-column cards, but the embedded data is contained in only the first 80 columns, the SYSIN-96 option should not be used.

All rules regarding job control stream format and job control stream errors (9.1.1.1) apply to ACCEPT with SYSIN-96.

9.1.1.3. 8413 Diskette ACCEPT

An ACCEPT with mnemonic-name associated with SYSIN-128 allows the COBOL program to retrieve embedded data images from the job control stream when using an 8413 diskette with data extending beyond position 80. When the job control stream is recorded on 8413 diskette, but the embedded data is contained in only the first 80 columns, the SYSIN-128 option should not be used.

All rules regarding job control stream format and job control stream errors (9.1.1.1) apply to ACCEPT with SYSIN-128.

9.1.2. Console ACCEPT

An ACCEPT with mnemonic-name associated with SYSCONSOLE allows the program to receive data from the system console.

The maximum number of characters that may be entered for a single ACCEPT is 60.

When the ACCEPT statement is encountered in the COBOL program, the following message is displayed:

CA10 ACCEPT READY

The operator, when replying to a system console ACCEPT, must enter "message number" followed by the text.

When the operator types less than the number of characters expected, the remaining positions are space-filled (X'40').

The identifier must be implicitly or explicitly defined as USAGE IS DISPLAY (5.3.5).

9.1.3. Current Date ACCEPT

An ACCEPT with mnemonic-name associated with SYSDATE or an ACCEPT with the DATE option makes the date 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 (6.6.3.2).

When the date is set through the job control stream (//ΔSETΔDATE, YY/MM/DD) the date is stored in the user's job preamble. If the date is not set via the job control stream, job control moves the date from the system information block (SIB) into the user's job preamble. The date in the SIB is entered via the system console by the operator. This is accomplished by using the operator SET command to enter the current date.

By setting the date from the job control stream, the user can predate or postdate jobs.

9.1.4. Time of Day ACCEPT

An ACCEPT with mnemonic-name associated with SYSTIME or an ACCEPT with the TIME option makes the time of day available to the program in the format hhmmss00 (PIC 9(8)), where hh is the hour, mm is the minute, and ss is the second (hhmmss does not exceed 235959). This information is moved to the identifier under the rules for a COBOL MOVE (6.6.3.2).

9.1.5. Julian Date ACCEPT

An ACCEPT with the DAY option makes the date available to the program in the format yyddd (PIC 9(5)). This information is moved to the identifier under the rules for a COBOL MOVE (6.6.3.2). See 9.1.3, Current Date ACCEPT, for information on setting the date with the job control //ΔSETΔDATE command.

9.1.6. UPSI Byte ACCEPT

An ACCEPT with mnemonic-name associated with SYSSWCH permits the COBOL program to access the user program switch indicator (UPSI) byte, which is the last byte of the 12-byte communications region in the job preamble. An 8-byte item is created containing EBCDIC 0 to represent the OFF status and an EBCDIC 1 to represent the ON status of the individual UPSI bits/switches, respectively (e.g., if SYSSWCH-0 and SYSSWCH-2 are ON and all others are OFF, the ACCEPT statement makes available to the program an 8-character item containing 10100000).

9.1.7. Communications Region ACCEPT

An ACCEPT with mnemonic-name associated with 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.

9.2. DISPLAY STATEMENT

Format:

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

9.2.1. Console DISPLAY

A DISPLAY with mnemonic-name associated with SYSCONSOLE permits the COBOL program to display messages upon the system console. A display on the system console is assumed if the UPON option is omitted. The sum of the sizes of operands may not exceed 250 characters. The data is displayed a line at a time. Each line is prefixed with the code CD10 and followed by a maximum of 55 characters of the contents of the operands.

All displays are action-type messages and must be responded to by the operator with a GO command.

9.2.2. Log File DISPLAY

A DISPLAY with mnemonic-name associated with SYSLOG permits the COBOL program to display messages to the system console and the system log file. Message size is limited to 55 contiguous characters. COBOL displays are prefixed with the code CD11. This display is an informational-type message and does not require the operator to respond with a GO command (unlike SYSCONSOLE).

9.2.3. UPSI Byte DISPLAY

A DISPLAY with mnemonic-name associated with 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 the resultant eight 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 (//ΔSETΔUPSI, switch-setting).

9.2.4 UPSI Bit DISPLAY

A DISPLAY with mnemonic-name associated with 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 1-byte identifier (PIC X) is used to alter UPSI SWITCH-n. A value of 0 (X'F0') causes the switch to be turned OFF (0 value); any other value causes the switch to be turned ON (1 value).

9.2.5. Communications Region DISPLAY

A DISPLAY with mnemonic-name associated with SYSCOM allows the COBOL program to alter the contents of the communications region. The 12 bytes described by the identifier are moved into the 12-byte communications region in the job preamble.

The communications region is initialized to binary 0's prior to the first job step by job control. Through use of the SET statement (///ΔSETΔCOMREG, character-string), the communications region may be set to an initial value. Information may be passed from job step to job step in the region. The region is not changed during job steps.

9.2.6. Printer Listing DISPLAY

A DISPLAY with mnemonic-name associated with SYSLST permits the COBOL programmer to display messages on the printer. Displays are in 120-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 associated with SYSLST must be performed in the nondeclarative portion of the procedure division before any are performed within the declarative portion.

10. Table Handling

10.1. 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 available to the COBOL user in the SPERRY UNIVAC Operating System/3 (OS/3). For a complete discussion of table handling, see the fundamentals of COBOL table handling manual, UP-7503.2 (current version).

10.2. 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 1-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 (5.3.3).

Defining a 1-dimensional table within each occurrence of a table element gives rise to a 2-dimensional table. This is done by writing an OCCURS clause for a data item subordinate (i.e., with a 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 OS/3 COBOL. Each dimension must be defined by an OCCURS clause, and must be defined on a different hierarchical level.

10.3. 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. One subscript or index level for each OCCURS clause must be in the defined hierarchy containing 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.

10.4. SUBSCRIPTING

Definition:

Subscripting is a technique used to reference individual table elements within a table of like elements not assigned individual data-names. A subscript value identifies elementary items in the table.

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 subscripts in parentheses. A left parenthesis may not be followed by a space; a right parenthesis may not be preceded by a space.

4. When more than one subscript is used in a reference, each must be separated, within the parentheses, by a comma and a space.

10.5. INDEXING

Definition:

Indexing is a technique used to reference individual table elements within a table of like elements not assigned individual data-names. An index-name contains the occurrence number of a table element used for:

- direct indexing by using the index-name as a subscript; or
- relative indexing by appending to the index-name the + or − operator followed by an unsigned integer. This integer must not be 0.

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 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 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-names in parentheses.

4. When more than one index is used in a reference, each must be separated, within the parentheses, by a comma and a space. ←

5. 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 the SET statement.

10.6. SEARCHING

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 (6.6.6.2).

11. Processing Techniques for Direct Access Devices

11.1. INTRODUCTION

This section describes the techniques available to the COBOL programmer for processing files assigned to direct access devices. The technique chosen to process a particular file depends upon the file organization and the manner in which records within the file are accessed. Each file organization has its particular advantages and disadvantages. No attempt is made in this section to select one organization over another. In selecting a file organization, the user should consider factors such as device characteristics, file size, activity, growth potential, etc. This section is intended to inform the user of the capabilities, construction, and usage of the file organizations available on direct access devices.

11.2. FILE ORGANIZATION

File organization specifies the format and control of the logical file structure. Once a file is created under a specific organization, that organization cannot be altered for subsequent file processing. COBOL provides three classes of file organizations:

1. Sequential
2. Relative
3. Indexed

A file organization is specified by the ORGANIZATION clause in the SELECT entry for this file.

11.2.1. Sequential Organization

The logical file structure is such that each logical record (except the first and last) has a unique predecessor and unique successor record. The predecessor-successor relationship was established by the order of the WRITE function when the logical file structure was created. The control of placing records to, or retrieving records from, a sequentially organized file is the predecessor-successor relationship; i.e., the sequence in which records are created is the sequence in which they are retrieved. No other control information is required to access records from sequential files.

11.2.2. Relative Organization

The logical file structure is characterized by the physical relationship (location) of each record to the first record; i.e., logical record 1 occupies the first physical location in the file, record 2, the second, etc. In addition to sequential processing capabilities, records in a relative-organized file can be read or written directly by specifying the record number of the desired record. This control of the file is referred to as random access. For example, the fifteenth record of a relative-organized file may be accessed directly, whereas access to the fifteenth record of a sequentially organized file can be achieved only after retrieving the first 14 records. The ability to randomly access records provides an advantage over sequentially organized files; however, the data management techniques used with relative files restrict the format of records to fixed-length, unblocked.

11.2.3. Indexed Organization

Indexed files comprise two elements: the prime data set consisting of the logical records of the file and an index which expedites access to records in the prime data area. Each logical record of the file contains a field designated as the key. The key is the control which the access method uses in constructing the file as well as for subsequent retrievals. The access method uses a search of the index to locate the address of the record containing the requested key. The access method requires that indexed files be created in key sequence; hence, the name, indexed sequential. Records may be added to an existing indexed file; each added record is placed in overflow areas and the sequence of the file is maintained logically. Retrieval time of records increases as the number of records in overflow increases. Periodic reorganization of indexed files should be practiced to alleviate this condition.

11.3. ACCESS METHODS

Three modes of access (the manner in which records are read or written to a file) are available to the COBOL programmer: sequential access, random access, and extended access.

11.3.1. Sequential Access

Sequential processing involves the serial placement or retrieval of records to or from a file. The control in a sequential access method is the order in which records are written to or read from a file. No control information (key) need be supplied by the programmer to the access method (data management) other than the request to read or write a record. Any file organization can be accessed sequentially.

11.3.2. Random Access

Random processing assumes no serial dependency of records within a file. Each request to access a record is treated individually, without regard to prior requests. Information (key) is supplied at the time of request to designate the desired record. Random access is available only on files with relative or indexed organization.

11.3.3. Extended Access

Extended processing indicates that random and sequential access may be mixed. It is only available on files with indexed organization.

11.4. CLAUSES REQUIRED FOR FILE PROCESSING

The specification of file organization, access method, and OPEN usage (input, output, I/O) dictates the file processing technique. Each file processing technique is described, in turn, with emphasis on the COBOL clauses required to define the file, and the effects these clauses have during file processing. Refer to Table 11–8 for a summary of the following information.

11.4.1. Sequential File Processing

The following COBOL clauses are used when processing sequentially organized files:

1. ORGANIZATION IS SEQUENTIAL

   The ORGANIZATION IS SEQUENTIAL clause states that the file is organized in a serial manner. Records are accessed one after the other. Sequential organization is assumed if this clause is omitted. Keys are not allowed with sequential files.

2. ACCESS MODE IS SEQUENTIAL

   The ACCESS MODE IS SEQUENTIAL clause specifies the manner in which the records are to be written or retrieved from the file. Only sequential access is permitted indicating serial retrieval.

3. RECORDING MODE IS { F / V }

   F signifies fixed mode and V signifies variable mode. Fixed-length or variable-length records may be blocked or unblocked.

4. RESERVE { integer-n / NO } ALTERNATE [{ AREA / AREAS }]

   The RESERVE ALTERNATE AREAS clause indicates the number of additional I/O areas desired. Omission of the clause results in the allocation of one additional I/O area. If NO is specified no additional area is allocated. The only allowable integer is 1.

5. BLOCK CONTAINS integer-n { CHARACTERS / RECORDS }

   Indicates the number of records or characters per block. The actual space allocated to an I/O area is always a multiple of 256 bytes.

The following input/output statements are applicable to sequential files:

1. OPEN INPUT file-name

   The OPEN INPUT statement indicates that the file operates in a read-only mode. Standard labels are checked and user labels, if specified, are made available to the USE for beginning label procedure.

2. OPEN OUTPUT file-name

   The OPEN OUTPUT statement indicates that the file will operate in a write-only mode. Standard labels are written and user labels, if specified, are made available to the user for beginning label procedure.

3. OPEN I-O file-name

   The OPEN I-O statement indicates that the file is to be updated. Each WRITE statement must be preceded by a READ statement. Alteration of record length, insertion of new records, or deletion of existing records is not permitted.

4. READ file-name RECORD AT END imperative-statement

   The READ AT END statement causes the next sequential record in the file to be made available (after deblocking), or if the end of file is detected, performs the special imperative statement following the AT END clause.

5. WRITE record-name [; INVALID KEY imperative statement]

   The WRITE statement causes the specified record to be written in the next sequential area of the file (after blocking). An INVALID KEY condition occurs when there is insufficient space in the file to add another record.

6. CLOSE file-name

   The CLOSE statement causes orderly termination of file processing. (At the end of the file or volume, user labels are checked and created if specified.)

11.4.2. Relative File Processing

The following COBOL clauses are used in processing relative organized files:

1. ORGANIZATION IS RELATIVE

   The ORGANIZATION IS RELATIVE clause designates the file as relatively organized. The file is accessed via relative record number. The ORGANIZATION IS RELATIVE clause causes data management relative access method routines to be linked into object program. This is a required clause.

2. ACCESS MODE IS { RANDOM / SEQUENTIAL }

   The ACCESS MODE clause specifies the manner in which records are written to or retrieved from the file.

   a. The RANDOM option indicates no serial dependency of record processing. The relative record to be read, written, or sought is specified by the contents of the actual or relative key.

   b. The SEQUENTIAL option demands serial processing of records to or from the file and requires no key when accessing records. Sequential is assumed if this clause is omitted.

3. RECORDING MODE IS F

   Only fixed-length record format is available for relative organized files.

4. BLOCK CONTAINS integer-n { CHARACTERS / RECORDS }

   Relative files may not be blocked. This clause is not required. Space allocated to the I/O area is a multiple of 256 bytes.

5. { ACTUAL / RELATIVE } KEY IS data-name

   The ACTUAL or RELATIVE KEY IS clause specifies the data-name containing the relative record number to be read, written, or sought. This field is set by the programmer and/or the data management access method under the following conditions:

   a. Random access

      Programmer moves a relative record number to the field prior to every READ, WRITE, or SEEK verb. The contents of the field are unchanged after execution of the I/O command.

b. Sequential access

The contents of the actual or relative key are not required for READ or WRITE statements; therefore, the field is ignored by the data management access method. Pointers to the next sequential record are maintained by the access method while advancing through the file. After execution of a READ or WRITE statement, the contents of the actual key reflect the relative record number of the record just processed. Under sequential access, the programmer may issue a SEEK statement to position the file to a particular record. In this case, the programmer's relative record number is moved to the actual key prior to issuance of the SEEK statement. This technique of issuing a SEEK statement before each READ or WRITE statement has the effect of randomly accessing a relative file defined under sequential access.

c. File open output (either access method)

In the event that file preparation is requested on output files, the actual key should contain the relative record number on which file preparation is to begin. The file is prepped from this point to the end of the user's file extent.

NOTE:

It is the programmer's responsibility to ensure that the actual key contains the relative record number prior to opening the file.

6. APPLY FILE-PREPARATION ON file-name

The APPLY FILE-PREPARATION clause specifies the relative-organized file name on which file preparation is required. For relative-organized files, file preparation consists of writing initializing data on each track of the user's extent, starting at the relative record number contained in the actual key location and proceeding to the end of the user's extent. This initializing data, required by data management access methods, consists of an 8-byte count field plus a dummy record of length equal to the fixed size of records within the file. The dummy record consists of an X'FF' followed by all 0's up to a maximum of 255 bytes. (If the record size is greater than 256, undetermined data follows byte 256.) This file prepping guarantees that a physical record exists in every possible area of the user's extent, making it possible to access these record areas directly (randomly).

When the initial allocation of disk space is exhausted, relative files are not extended automatically. If APPLY FILE PREPARATION is specified and the relative key data item contains a record number one higher than the highest record in the file (i.e., the first record in the next extent), the file is extended by one secondary increment of disk space when the OPEN OUTPUT statement is executed.

NOTE:

For initial creation of a relative file, the programmer should set the ACTUAL KEY field to 1 prior to opening the file.

The following input/output statements are applicable to relative files:

1. OPEN INPUT file-name

The OPEN INPUT statement indicates that the file is used in a read-only mode. Standard labels are checked and user labels, if specified, are made available to the USE for BEGINNING LABEL procedure. For sequential access, the file is positioned to the first record.

2. OPEN OUTPUT file-name

The OPEN OUTPUT statement indicates that the file is used in a write-only mode. The file is formatted if the APPLY FILE-PREPARATION clause was specified starting at the record number contained in actual key and proceeding to the end of the user's extent. The USE for BEGINNING LABEL procedure is executed if specified. The file is positioned to the first record for sequential access.

3. OPEN I-O file-name

The OPEN I-O statement designates the existing file as the one to be updated. (Both READ and WRITE statements may be issued to the file.) Label processing is the same as when the file is opened for input. This type of OPEN statement affects the manner in which WRITE statements function. Each WRITE statement is dependent upon a READ or SEEK READ statement previously issued for the file. The WRITE order is issued for the relative record specified on the previous READ or SEEK statement.

4. Sequential access

a. READ file-name RECORD AT END imperative-statement

The READ AT END statement for sequential access method delivers the next logical record from an input file, or performs the specified imperative statement following the AT END clause if the end of the file is detected.

b. WRITE record-name; INVALID KEY imperative-statement

The WRITE INVALID KEY statement releases a logical record to an output file. The imperative statement following the INVALID KEY clause is executed when the end of file is detected and an attempt is made to execute a WRITE statement for that file.

c. SEEK file-name RECORD

The SEEK statement positions the file to the relative record number specified by the contents of the actual key. No error indication is available if the record is not located. Error indications are available on the succeeding READ or WRITE statements.

5. Random access

a. READ file-name RECORD INVALID KEY imperative-statement

The READ INVALID KEY statement delivers the logical record specified by the contents of the actual key, or executes the imperative statement following INVALID KEY clause if the record specified by the actual key does not exist within the user's extent.

b. $\left\{\begin{matrix}\text{WRITE}\\\text{REWRITE}\end{matrix}\right\}$ record-name INVALID KEY imperative-statement

The WRITE or REWRITE INVALID KEY statement writes the logical record to the physical area of the disc specified by the relative record number contained in the actual key. If that record does not exist in the user's extent, the INVALID KEY imperative statement in the INVALID KEY clause is executed.

c. SEEK file-name RECORD

The SEEK statement positions the file to the relative record number specified by the contents of the actual key.

6. CLOSE file-name

See CLOSE statement under sequential file processing (11.4.1).

11.4.3. Indexed File Processing

The following clauses are used in processing indexed sequential files:

1. ORGANIZATION IS INDEXED

   The ORGANIZATION IS INDEXED clause denotes file organization as indexed sequential, and causes data management indexed sequential access method (ISAM) routines to be linked into the object program.

2. ACCESS MODE IS { RANDOM / SEQUENTIAL / EXTENDED }

   The ACCESS MODE clause specifies the order in which records are written to, or read from, the file.

   a. Sequential

      The sequential access mode requires a serial processing of records to or from the file; therefore, no key need be presented when retrieving records. Indexed files can be created only under sequential access.

   b. Random

      The random access mode requires no serial dependency of record processing. The key of the record to be read or sought is specified in the SYMBOLIC KEY clause. New records can be inserted into an existing indexed file under random access.

   c. Extended

      The extended access mode combines sequential and random record processing.

3. RECORDING MODE IS { F / V }

   The RECORDING MODE IS F clause indicates fixed-length records. The RECORDING MODE IS V clause indicates variable-length records. Fixed- or variable-length blocked records are the only formats available for indexed files.

4. RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

   The RECORD CONTAINS clause indicates the size of the records. If the records are variable in length, a high/low range can be specified.

5. BLOCK CONTAINS integer-n { CHARACTERS / RECORDS }

   The BLOCK CONTAINS clause indicates the number of records or characters per block. Space allocated to the I/O area is a multiple of 256 bytes.

6. SYMBOLIC KEY IS data-name

   The SYMBOLIC KEY clause specifies the data-name containing the key of the record to be read or sought. This key field must match the size and description of the record key field.

7. RECORD KEY IS data-name

The RECORD KEY clause specifies the field within each record containing the record identification. This field is used at file creation time to build the indexes required for subsequent file processing. At retrieval time, the contents of the programmer-supplied SYMBOLIC KEY field are compared against the defined RECORD KEY field in accessing indexed records randomly. The key field must be greater than 2 and less than or equal to 249 bytes in length.

8. APPLY CYLINDER-INDEX AREA OF integer-n INDICES ON file-name

The APPLY CYLINDER-INDEX AREA clause establishes levels of indexes to expedite the retrieval of records. When an indexed file is created, data blocks containing records are loaded sequentially. Each record contains an embedded key (see RECORD KEY, item 7). As each data block is filled with records and written to disc, the key of the highest record in the block is recorded in a block index entry, along with the disc address of the block. When a track on the disc becomes filled with blocks of block index entries, an entry in the top index is created containing the highest key on the track of block index entries. Retrieval of records reverses the process. To eliminate the disc reads required to access the top index for retrieval, sufficient storage should be allocated to contain a number of top index entries. Integer-n specifies the number of top index entries to be held in storage. If all top index entries can be held in storage, then all reads to access the top index are eliminated.

The method used to calculate the value of integer-n in the APPLY CYLINDER-INDEX AREA clause is described in detail in 10.2.4 (Calculating Space for the ISAM Index Area) in the data management user guide, UP-8068, current version.

If the file already exists, use the following formula to determine the value of integer-n:

$$n = b/(s + 3)$$

where:

n

Is integer-n of the APPLY clause.

b

Is bytes that are required for main storage and that can be obtained from a display of the VTOC. The number of bytes is shown under the heading: Bytes Required for Main Storage.

s

Is size of the record key.

NOTE:

If the remainder of the divide operation in the above formula is not equal to zero, add 1 to the quotient, i.e., to n.

9. APPLY MASTER-INDEX ON file-name

The APPLY MASTER-INDEX clause is accepted for OS/4 and OS/7 compatibility. In OS/3, this clause serves for documentation only.

10. APPLY CYLINDER-OVERFLOW AREA OF integer-n PERCENT ON file-name

To keep disc head movement to a minimum in retrieving records from overflow, a percentage of each cylinder in the prime data area can be allocated to contain overflow records. If this clause is omitted, 20 percent of each cylinder is set aside to contain overflow records. If no overflow is desired, 0 percent should be specified. In this case, no new records may be inserted into the file. If specified, integer-n is an unsigned number.

11. APPLY EXTENDED-INSERTION AREA ON file-name

The APPLY EXTENDED-INSERTION AREA clause is accepted for OS/4 and OS/7 compatibility. In OS/3, this clause serves for documentation only.

12. APPLY VERIFY ON file-name

The APPLY VERIFY clause requests verification (READ after WRITE) of disc records after they have been written. If this clause is omitted, no verification of records is performed.

13. RESERVE {integer-n / NO} ALTERNATE [AREA / AREAS]

The RESERVE ALTERNATE clause indicates the number of additional I/O areas desired. The key word NO causes no additional I/O areas to be reserved; integer-n (which must be a one) reserves one additional I/O area. If this clause is omitted, no additional I/O areas are allocated.

14. LABEL {RECORD IS / RECORDS ARE} STANDARD

The reserved word STANDARD specifies that system file labels are to be checked (or created) and that the labels conform to OS/3 label specification.

The following input/output statements are used when processing indexed files:

1. OPEN OUTPUT file-name

The OPEN OUTPUT statement indicates the file is to be loaded or extended. The creation of a file (load) with standard labels is assumed unless the file already exists, in which case file extension is implied. This statement can only be specified for sequential access or extended access.

2. OPEN INPUT file-name

The OPEN INPUT statement indicates that the file is to be used in a read-only mode. Standard labels are checked. For sequential and extended access, the file is positioned to the first record. This statement can also be specified for random access.

3. OPEN I-O file-name

The OPEN I-O statement indicates that the file is to be used in a read and write mode. Standard labels are checked. For sequential and extended access, the file is positioned to the first record. This statement can also be specified for random access.

4. SEEK file-name RECORD

For sequential file processing, the SEEK statement causes the programmer-supplied value in the SYMBOLIC KEY item to specify the RECORD KEY value of the logical record within the file which is to be positioned for subsequent sequential retrieval. If no logical record is found with that key, positioning is made to the record with the next higher key.

The SEEK statement can be used only under sequential or extended access mode when opened for INPUT or I-O.

5. READ file-name RECORD [INTO identifier] [; {AT END / INVALID KEY} imperative-statement]

For sequential file processing, the READ statement makes available the next logical record from a file and allows performance of a specified imperative-statement when the end of the file is detected. The logical record retrieved is determined by the preceding input/output statements as shown in Table 11–1.

Table 11–1. Logical Record Retrieval by Sequential Read

Preceding Input/Output Statement	Logical Record Retrieved by Sequential Read
OPEN	First record of file
SEEK	Record with SEEK key or, if key does not exist, record with next higher record key
READ	Record with next higher record key after last retrieved record
WRITE/REWRITE/INSERT	Does not affect positioning for sequential read

For random file processing, the READ statement makes available the record specified by SYMBOLIC KEY, and allows performance of a specified imperative-statement if a logical record with that key does not exist.

When AT END is specified, the READ statement is treated as a sequential read and the access mode must be sequential or extended. When INVALID KEY is specified, the READ statement is treated as a random read and the access mode must be random or extended. If neither AT END nor INVALID KEY is specified, the type of read is determined by the accesss mode. If access is sequential, the read is a sequential read. If access is extended or random, the read is a random read. The file must be opened for INPUT or I-O for the READ to be valid.

6. WRITE record-name [FROM identifier-1] [; INVALID KEY imperative-statement]

The WRITE statement releases a logical record for an output file.

- File loading, extending

  When loading or extending a file, the WRITE statement is used to add logical records sequentially in the prime data area of the file and to create the necessary index entries for later retrieval of the logical records. The logical records must be presented for loading in ascending record key sequence. If the file is being extended, the RECORD KEY value of the first logical record written must be higher than the highest RECORD KEY value currently in the file. The WRITE statement allows performance of a specified imperative-statement if the RECORD KEY is equal to, or out of key sequence with, the last RECORD KEY.

  The WRITE statement can be used only for file loading or extension under sequential or extended access when opened for OUTPUT.

- Record update

  When updating an existing record, the WRITE statement must be preceded by a successful READ statement. The WRITE statement causes the updated record to be rewritten into its original physical area. Neither the length nor the RECORD KEY value can be changed. The WRITE statement allows performance of a specified imperative-statement if the length or key value have been modified.

  The WRITE statement can be used only for record updating under sequential, random, or extended access when opened for I-O.

- Record insertion

  When inserting a new record into an indexed file, the WRITE statement causes a new logical record to be added to the file at the logical position designated by its RECORD KEY value. No other logical record may exist in the file with the same RECORD KEY value. The WRITE statement allows performance of a specified imperative-statement if a logical record with the RECORD KEY already exists.

  The WRITE statement can be used only for record insertion under random or extended access when opened for I-O.

7. REWRITE record-name [FROM identifier] [; INVALID KEY imperative statement]

   The REWRITE statement can be used in place of the WRITE statement for record update. The same rules used for record update for the WRITE statement apply.

8. INSERT record-name [FROM identifier-1] [; INVALID KEY imperative-statement]

   The INSERT statement can be used in place of the WRITE statement for record insertion. The same rules used for record insertion for the WRITE statement apply.

9. CLOSE file-name

   See CLOSE statement for sequential file processing (11.4.1).

A summary of input/output statements permitted for each access method and open mode follows.

- ORGANIZATION is INDEXED, ACCESS is SEQUENTIAL

Sequential Output Processing	Sequential Input Processing	Sequential I-O Processing
OPEN OUTPUT	OPEN INPUT	OPEN I-O
WRITE [INVALID KEY]①	READ [AT END]②	READ [AT END]②
CLOSE	SEEK	SEEK
	CLOSE	WRITE [INVALID KEY]①
		REWRITE [INVALID KEY]
		CLOSE

NOTES:

① *When access is sequential and the file is opened for OUTPUT, the WRITE statement is a request for loading or extending the file. When opened for I-O, the WRITE statement is a request for an update of an existing record.*

② *When access is sequential, a READ statement is always treated as a sequential read.*

- ORGANIZATION is INDEXED, ACCESS is RANDOM

Random Output Processing	Random Input Processing	Random I-O Processing
Invalid OPEN mode	OPEN INPUT	OPEN I-O
	READ [INVALID KEY] ②	READ [INVALID KEY]②
	CLOSE	WRITE [INVALID KEY]①
		REWRITE [INVALID KEY]
		INSERT [INVALID KEY]
		CLOSE

NOTES:

① *When access is random and the file is opened for I-O, the WRITE statement is either a request for an update of an existing record or else a request for insertion of a new record.*

② *When access is random, a READ statement is always treated as a random read.*

- ORGANIZATION is INDEXED, ACCESS is EXTENDED

Extended Output Processing	Extended Input Processing	Extended I-O Processing
OPEN OUTPUT	OPEN INPUT	OPEN I-O
WRITE [INVALID KEY] ①	READ [{AT END / INVALID KEY}] ②	READ [{AT END / INVALID KEY}] ②
CLOSE	SEEK	SEEK
	CLOSE	WRITE [INVALID KEY] ①
		REWRITE [INVALID KEY]
		INSERT [INVALID KEY]
		CLOSE

NOTES:

① *When access is extended and the file is opened for OUTPUT, the WRITE statement is a request for loading or extending the file. When opened for I-O, the WRITE statement is a request for either an update of an existing record or else a request for inserting a new record.*

② *When access is extended and the file is opened for INPUT or I-O, if neither AT END or INVALID KEY is specified for a READ statement, the READ statement is treated as a random read. If AT END is specified, the READ is treated as a sequential read. If INVALID KEY is specified, the READ is treated as a random read.*

11.4.4. Summary of Imperative Statements and Error Conditions

The use of the AT END/INVALID KEY imperative-statement with the ORGANIZATION clause, system error messages, and COBOL disc processing techniques are summarized in the following paragraphs.

11.4.4.1. ORGANIZATION IS SEQUENTIAL Clause

The AT END imperative-statement is executed when the logical end of file is detected.

The INVALID KEY imperative-statement is executed when no space is left on the file for the record to be written.

11.4.4.2. ORGANIZATION IS RELATIVE Clause

The AT END imperative-statement is executed when an access to a record beyond the file is attempted.

The INVALID KEY imperative-statement is executed when the relative-record number is beyond the file extents.

11.4.4.3. ORGANIZATION IS INDEXED Clause

The AT END/INVALID KEY imperative-statement clauses are executed according to the explanation given in 11.4.3. See also Table 11–3 for a list of the AT END/INVALID KEY exception conditions.

Exception conditions for indexed files are handled in the following manner:

- Warning Exceptions

  When a warning exception condition arises during COBOL verb processing for indexed files, control is returned immediately following the input/output verb with the appropriate SYSERR setting. The warning exception condition is shown in Table 11–2.

Table 11–2. Warning Exception Conditions for Indexed File Processing

Warning Exception Condition	COBOL Verb
End of file detected when positioning unit for subsequent sequential retrieval	SEEK

- End-of-File/Invalid Key Exceptions

  When an end-of-file condition or invalid key condition arises during COBOL verb processing for indexed files, SYSERR is appropriately set and the AT END/INVALID KEY imperative-statement is executed. If no AT END/INVALID KEY imperative-statement clause is specified when this condition occurs, control is transferred to the appropriate USE AFTER ERROR procedure. If this latter procedure is not specified, the COBOL ERROR procedure is called and results in an end-of-job sequence. The AT END/INVALID KEY exception conditions are shown in Table 11–3.

Table 11–3. AT END/INVALID KEY Exception Conditions for Indexed File Processing

AT END/INVALID KEY Exception Conditions	COBOL Verb
End of file detected (AT END condition)	Sequential READ
During file creation or extension, a record-key value is found out of key sequence (INVALID KEY condition).	Load WRITE
A duplicate record-key value is detected (INVALID KEY condition).	Load WRITE Insert WRITE INSERT
A specified record-key value cannot be formed (INVALID KEY condition).	Random READ
A record-key value or length value for a record update has been modified (INVALID KEY condition).	Update WRITE WRITE

- Unrecoverable File Errors

When unrecoverable file errors occur during COBOL verb processing for indexed files, control is transferred to the applicable USE AFTER ERROR procedure with the appropriate SYSERR message set. If a USE AFTER ERROR procedure is not provided, the COBOL ERROR procedure is called and results in an end-of-job sequence. The unrecoverable file error conditions are shown in Table 11–4.

Table 11–4. Unrecoverable File Error Conditions for Indexed File Processing (Part 1 of 2)

Unrecoverable File Error Conditions	COBOL Verb
General OPEN errors	OPEN
General CLOSE errors	CLOSE
Invalid use of COBOL verb: – COBOL verb not valid for open mode – OPEN issued to file currently opened – Verb other than OPEN issued to file not currently opened – Update not preceded by a successful READ – Because of previous errors, only CLOSE verb permitted	All COBOL verbs
Insufficient file space	CLOSE Load WRITE Insert WRITE

Table 11–4. Unrecoverable File Error Conditions for Indexed File Processing (Part 2 of 2)

Unrecoverable File Error Conditions	COBOL Verb
No AT END/INVALID KEY imperative-statement specified for COBOL verb and exception condition occurred when processing the verb	Sequential READ Random READ Load WRITE Update WRITE Insert WRITE REWRITE INSERT
Hardware error	All COBOL verbs
Invalid record size	Load WRITE Insert WRITE INSERT
Data portion of track index destroyed (invalid ID) or invalid index search	OPEN SEEK Sequential READ Random READ Insert WRITE INSERT

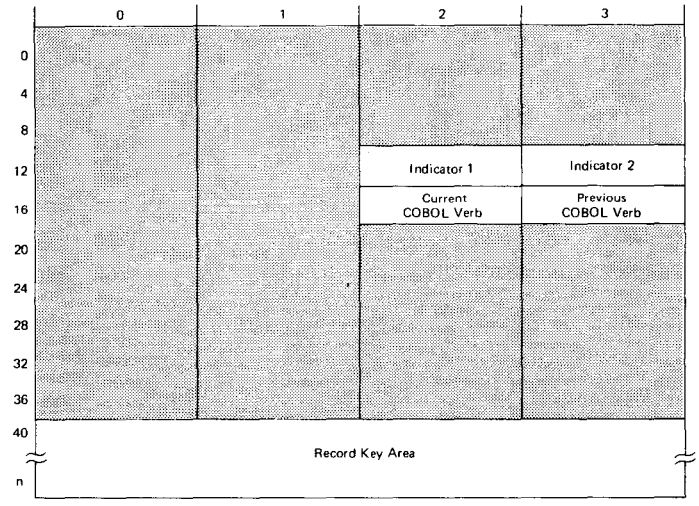
- Storage Dump

If an unrecoverable file error occurs and control is transferred to the COBOL ERROR procedure, this procedure takes a dump of the job region before job termination. The following information is available:

– Register values

Register	Value
0	SYSERR setting
1	Address of DTF of file in error
2	Address of prefix if file in error
14	Address of return locations in program if error had not occurred

– File prefix format



The shaded area is the prefix to the DTF module.

Name	Byte	Bits	Description
Indicator 1	14	0–1	Used by COBOL internally
		2–3	Access mode
			00 – sequential 10 – random 01 – extended
		4–7	Used by COBOL internally
Indicator 2	15	0–2	Open mode
			100 – input 010 – output 001 – I-O
		3–7	Used by COBOL internally
Current COBOL Verb	18	0–7	Code for COBOL verb processed for indexed file when exception condition occurred
Previous COBOL Verb	19	0–7	Code for COBOL verb processed for indexed file that preceded current COBOL verb

Code	COBOL Verb
00	OPEN
01	CLOSE
02	SEEK
03	READ (sequential)
04	READ (random)
05	WRITE (load)
06	WRITE/REWRITE (update)
07	Not used
08	WRITE/INSERT (insert)

RECORD KEY Area	40–20	–	Record-key used for sequential retrieval positioning

Table 11–5 summarizes the exception conditions for each input/output COBOL verb used for processing indexed files.

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 1 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
OPEN	00	General OPEN error	SYSERR-2, SYSERR-4	OPEN not completed. File processing cannot continue.	USE AFTER ERROR procedure
		File currently opened	SYSERR-6	OPEN not completed. File processing cannot continue.	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	OPEN not completed. File processing cannot continue.	USE AFTER ERROR procedure
		Invalid I-O when positioning to beginning of file (opened INPUT, I-O; access SEQUENTIAL or EXTENDED)	SYSERR-1	OPEN not completed. File processing cannot continue. File may not be valid.	USE AFTER ERROR procedure
		Invalid index search when positioning to beginning of file (opened INPUT, I-O; access SEQUENTIAL or EXTENDED)	SYSERR-28	OPEN not completed. File processing cannot continue. File may not be valid.	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 2 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
CLOSE	01	General CLOSE error	SYSERR-5	CLOSE not completed. File may not be valid.	USE AFTER ERROR procedure
		File not currently opened	SYSERR-6	CLOSE not completed. File still valid.	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	CLOSE not completed. File may not be valid.	USE AFTER ERROR procedure
		File not successfully loaded because of insufficient space	SYSERR-26	CLOSE not completed. File not valid and must be reloaded.	USE AFTER ERROR procedure
SEEK	02	File not currently opened	SYSERR-6	SEEK not completed	USE AFTER ERROR procedure
		SEEK not valid for open OUTPUT	SYSERR-6	SEEK not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 3 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
SEEK (cont)	02 (cont)	Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	SEEK not completed. File may not be valid.	USE AFTER ERROR procedure
		No record with key equal or greater than SEEK key found (end of file detected)	SYSERR-11 and SYSERR-25 (both always set) and SYSERR-3 not set	SEEK completed. If READ issued, AT END path will be executed. Normal file processing may continue.	Immediately after SEEK
		Invalid ID	SYSERR-1	SEEK not completed. File may not be valid.	USE AFTER ERROR procedure
		Invalid index search	SYSERR-28	SEEK not completed. File may not be valid.	USE AFTER ERROR procedure
		Because of preceding error(s), only CLOSE verb permitted	SYSERR-27	SEEK not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 4 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Sequential READ	03	File not currently opened	SYSERR-6	READ not completed	USE AFTER ERROR procedure
		READ not valid for open OUTPUT	SYSERR-6	READ not completed	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	READ not completed	USE AFTER ERROR procedure
		End of file detected	SYSERR-25	READ not completed. Before any further sequential retrieval can continue, it is necessary to reposition in the file. Normal file processing can continue.	If specified, AT END path; if not specified, USE AFTER ERROR procedure
		Invalid ID	SYSERR-1	READ not completed. File may not be valid.	USE AFTER ERROR procedure
		Invalid index search	SYSERR-28	READ not completed. File may not be valid.	USE AFTER ERROR procedure
		Because of preceding error(s), only CLOSE verb permitted	SYSERR-27	READ not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 5 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Random READ	04	File not currently opened	SYSERR-6	READ not completed	USE AFTER ERROR procedure
		READ not valid for open OUTPUT	SYSERR-6	READ not completed	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	READ not completed	USE AFTER ERROR procedure
		Specified record-key. Value cannot be found because a record with that key value was never added to the file.	SYSERR-11 and not SYSERR-3	READ not completed. Record not retrieved, but normal file processing may continue.	If specified, INVALID KEY path; if not specified, USE AFTER ERROR procedure
		Invalid ID	SYSERR-1	READ not completed. File may not be valid.	USE AFTER ERROR procedure
		Invalid index search	SYSERR-28	READ not completed. File may not be valid;	USE AFTER ERROR procedure
		Because of preceding errors, only CLOSE verb is permitted	SYSERR-27	READ not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 6 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Load WRITE	05	File not currently opened	SYSERR-6	WRITE not completed	USE AFTER ERROR procedure
		WRITE not valid for open INPUT	SYSERR-6	WRITE not completed	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	WRITE not completed	USE AFTER ERROR procedure
		Invalid record size	SYSERR-24	WRITE not completed	USE AFTER ERROR procedure
		Prime data area full or index area full	SYSERR-1	WRITE not completed. Record not written because of inedequate space. File should be closed.	USE AFTER ERROR procedure.
		Duplicate record key	SYSERR-30 SYSERR-29 (both always set)	WRITE not completed. because key already exists in file. Normal file processing can continue.	If specified, INVALID KEY path; if not specified, USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 7 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Load WRITE (cont)	05 (cont)	Record-key sequence error	SYSERR-29	WRITE not completed because key is not greater than last key in file. Normal file processing can continue.	If specified, INVALID KEY path; if not specified, USE AFTER ERROR procedure
		Because of preceding error(s), only CLOSE verb permitted	SYSERR-27	WRITE not completed	USE AFTER ERROR procedure
Update WRITE/ REWRITE	06	File not currently opened	SYSERR-6	Update not completed	USE AFTER ERROR procedure
		Update not valid for open INPUT or OUTPUT	SYSERR-6	Update not completed	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	Update not completed	USE AFTER ERROR procedure
		Update not preceded by random or sequential READ	SYSERR-6	Update not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 8 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Update WRITE/ REWRITE (cont)	06 (cont)	End of file detected for preceding sequential READ	SYSERR-6	Update not completed	USE AFTER ERROR procedure
		Record not found detected for preceding random READ	SYSERR-6	Update not completed	USE AFTER ERROR procedure
		A record-key value or length value for a record update has been modified.	None	Update not completed. Processing can continue.	If specified, INVALID KEY path; if not specified, USE AFTER ERROR procedure
		Because of preceding error(s), only CLOSE verb permitted.	SYSERR-27	Update not completed	USE AFTER ERROR procedure
Insert WRITE/ INSERT	08	File not currently opened	SYSERR-6	Insert not completed	USE AFTER ERROR procedure
		Insert not valid for open INPUT or OUTPUT	SYSERR-6	Insert not completed	USE AFTER ERROR procedure
		Hardware error: When one occurs, SYSERR-3 is always set along with one or more of the following: Unrecoverable error Unique unit error Record not found (hardware search) Unit exception Wrong length found Command rejection Intervention required Output parity check Equipment check Data check Overrun STOP state Device check	SYSERR-3 SYSERR-9 SYSERR-10 SYSERR-11 SYSERR-12 SYSERR-13 SYSERR-16 SYSERR-17 SYSERR-18 SYSERR-19 SYSERR-20 SYSERR-21 SYSERR-22 SYSERR-23	Insert not completed	USE AFTER ERROR procedure

Table 11–5. Exception Handling for COBOL Verbs Used for Indexed File Processing (Part 9 of 9)

COBOL Verb for Indexed File	Prefix Verb Code	Exception Condition	SYSERR Setting	File Processing Status	Transfer of Control
Insert WRITE/ REWRITE (cont)	08 (cont)	Invalid record size	SYSERR-24	Insert not completed	USE AFTER ERROR procedure
		Overflow area full	SYSERR-26	Insert not completed. Record not written because of inadequate space. Processing can continue.	USE AFTER ERROR procedure
Insert WRITE/ INSERT	08	ADD rejected because of error on preceding insert	SYSERR-31	Insert not completed. Processing can continue.	USE AFTER ERROR procedure
		Duplicate record key	SYSERR-29 SYSERR-30	Insert not completed because key already exists in file. Normal file processing can continue.	If specified, INVALID KEY path; if not specified, USE AFTER ERROR procedure
		Invalid ID	SYSERR-1	Insert not completed. File may not be valid.	USE AFTER ERROR procedure
		Invalid index search	SYSERR-28	Insert not completed. File may not be valid.	USE AFTER ERROR procedure
		Because of preceding error(s), only CLOSE verb permitted	SYSERR-27	Insert not completed	USE AFTER ERROR procedure
		Zero percent overflow allocated	SYSERR-31	Insert not completed	USE AFTER ERROR procedure

11.4.4.4. SYSERR Messages

Table 11–6 contains the definitions of the 32 SYSERR messages for ORGANIZATION INDEXED and ORGANIZATION RELATIVE. SYSERR is set whenever data management indicates an error has occurred. If no error occurs, all SYSERR settings will be off.

Table 11–6. System Error Messages (SYSERR) for INDEXED and RELATIVE Files

Message	Definition	Message	Definition
SYSERR-0	Last block on track accessed	SYSERR-17	Intervention required
SYSERR-1	Invalid ID	SYSERR-18	Output parity check
SYSERR-2	Invalid DTF (Indexed) Invalid PCA/DTF (Relative)	SYSERR-19	Equipment check
SYSERR-3	Hardware error	SYSERR-20	Data check
SYSERR-4	Error found in OPEN	SYSERR-21	Overrun
SYSERR-5	Error found in CLOSE	SYSERR-22	STOP state
SYSERR-6	Invalid macro sequence	SYSERR-23	Device check
SYSERR-7	Reserved (Indexed) WAITF required (Relative)	SYSERR-24	Invalid record size
SYSERR-8	I/O complete	SYSERR-25	Logical end of file
SYSERR-9	Unrecoverable error	SYSERR-26	File space exhausted (Indexed) Logical end of volume (Relative)
SYSERR-10	Unique unit error	SYSERR-27	Processing inhibited
SYSERR-11	Record not found	SYSERR-28	Invalid index (Indexed) Reserved (Relative)
SYSERR-12	Unit exception	SYSERR-29	Key sequence error (Indexed) Reserved (Relative)
SYSERR-13	Wrong length found	SYSERR-30	Duplicate key error (Indexed) Reserved (Relative)
SYSERR-14	End of track	SYSERR-31	ADD rejected (Indexed) Reserved (Relative)
SYSERR-15	End of cylinder		
SYSERR-16	Command rejection		

Additional information regarding error conditions can be found in the OS/3 data management user guide, UP-8068 (current version).

11.4.4.5. COBOL Disc Processing Techniques

Table 11–7 contains a summary of COBOL disc processing techniques.

Table 11–7. Summary of COBOL Disc Processing Techniques

Processing Technique		Addressing Technique	Required Key Clauses	Record Format ③	Open Verb	Allowable I/O Statements	Required Clauses	Optional Clauses	Restricted Clauses
Organization	Access								
SEQUENTIAL OR OMITTED	SEQUENTIAL OR OMITTED		NONE ALLOWED	F	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 CYLINDER-OVERFLOW
					OUTPUT	WRITE INVALID KEY			
				V	I-O	READ AT END WRITE INVALID KEY			
RELATIVE ①	SEQUENTIAL OR OMITTED	RELATIVE RECORD ②	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 RESERVE NO ALTERNATE AREA, READ INTO, WRITE FROM, INSERT FROM	RESERVE INTEGER, OPTIONAL, BLOCK CONTAINS >1 RECORD, USE ENDING LABEL
					OUTPUT	WRITE INVALID KEY, SEEK			
					I-O	READ AT END, WRITE ④ INVALID KEY, SEEK ⑤			
RELATIVE OR OMITTED	RANDOM	RELATIVE RECORD ②	ACTUAL OR RELATIVE	F	INPUT	READ INVALID KEY, SEEK			
					OUTPUT	WRITE INVALID KEY, SEEK			
					I-O	READ INVALID KEY, WRITE ④ INVALID KEY, SEEK ⑤			
INDEXED ①	SEQUENTIAL OR OMITTED		RECORD AND [SYMBOLIC] ⑥	F	INPUT	READ [AT END], SEEK	SELECT/ASSIGN LABEL RECORDS ARE STANDARD CLOSE	FOR MULTIPLE UNIT, RESERVE NO ALTERNATE AREA, RESERVE INTEGER ALTERNATE AREA, FILE LIMIT, PROCESSING MODE IS SEQUENTIAL, RERUN ON, SAME (RECORD) AREA, APPLY VERIFY, APPLY MASTER INDEX ON, APPLY CYLINDER OVERFLOW ON, APPLY CYLINDER INDEX AREA OF, APPLY EXTENDED-INSERTION AREA ON, BLOCK CONTAINS, RECORD CONTAINS, VALUE OF, DATA RECORDS ARE, USE ERROR INTO, FROM	FOR MULTIPLE REEL, MULTIPLE FILE TAPE, APPLY RESTRICTED SEARCH, APPLY BLOCK COUNT ON, APPLY FILE PREPARATION ON, APPLY ASCII, LABEL RECORDS ARE OMITTED OR DATA NAME, USE LABELS, OPTIONAL
				V	OUTPUT	WRITE [INVALID KEY]			
					I-O	READ [AT END], SEEK WRITE [INVALID KEY], REWRITE [INVALID KEY]			
INDEXED ①	RANDOM		RECORD AND [SYMBOLIC] ⑥	F	INPUT	READ [INVALID KEY]			
				V	I-O	READ [INVALID KEY], WRITE [INVALID KEY], REWRITE [INVALID KEY], INSERT [INVALID KEY]			
INDEXED ①	EXTENDED		RECORD AND [SYMBOLIC] ⑥	F	INPUT	READ [AT END / INVALID KEY] ⑦ SEEK			
				V	OUTPUT	WRITE [INVALID KEY]			
					I-O	READ [AT END / INVALID KEY] ⑦ SEEK, WRITE [INVALID KEY], REWRITE [INVALID KEY], INSERT [INVALID KEY]			

① American National Standard language element extension

② Requires preformatting of entire file prior to creation

③ Default RECORD FORMAT is underlined.

④ REWRITE accepted as synonym for WRITE.

⑤ SEEK not permitted between READ and WRITE

⑥ ACTUAL KEY may be used in place of SYMBOLIC KEY for UNIVAC 9300 System compatibility.

⑦ If AT END is specified, READ is treated as a sequential read.

12. Sorting

12.1. GENERAL

In the SPERRY UNIVAC Operating System/3 (OS/3) the 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.

12.2. 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 (SD) (5.2.2). 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 that file which is 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 return the sorted record from the sort file to the COBOL program. 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 or output procedure is in control, all transfers of control must refer to procedures contained within that input 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 the input or output procedure. Neither an input nor an output procedure may contain a SORT statement.

For a detailed discussion of COBOL sorting, consult the fundamentals of COBOL sorting manual, UP-7503.3 (current version).

12.3. SORT STATEMENT FORMATS

The following paragraphs summarize the entries used in OS/3 COBOL sorts.

12.3.1. Sort File SELECT Entry

Function:

The SELECT entry is used to name the sort file and to identify the hardware storage medium used during the sorting process.

Format:

SELECT file-name ASSIGN TO [external-name] [integer-1] implementor-name-1 [OR implementor-name-2]

Rules:

1. The SELECT entry is discussed in detail in 4.3.1.
2. The external name is not required in the sort file SELECT entry, as fixed external names are used.
3. Tape or disc subsystems 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.

12.3.2. SAME AREA Clause

Function:

The SAME AREA clause of the I-O-CONTROL paragraph is used to specify that two or more files are to use the same main storage area during processing.

Format:

SAME { RECORD / SORT } AREA FOR file-name-1 [, file-name-2] ...

Rules:

This clause, and the effect of the SORT option, are discussed in detail in 4.3.2, rule 3.

12.3.3. 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]
[; RECORDING MODE* IS { D | F | V }]
[; DATA { RECORD IS | RECORDS ARE } data-name-1 [, data-name-2] ...]
```

Rules:

Paragraph 5.2.2 lists the rules applicable to this statement.

12.3.4. RELEASE Statement

Function:

The RELEASE statement 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 6.6.4.10.

12.3.5. RETURN Statement

Function:

The RETURN statement 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 6.6.4.11.

12.3.6. SORT Statement

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.

**Extension to American National Standard COBOL (1968).*

Format:

```
SORT file-name-1 ON { ASCENDING  } KEY { data-name-1 } ...
                    { DESCENDING }     {           }

   [ ; ON { ASCENDING  } KEY { data-name-2 } ... ] ...
          { DESCENDING }     {           }

  { 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 6.6.4.12.

12.3.7. Use of the Sort Feature

The OS/3 Extended COBOL compiler generates linkage code to the OS/3 subroutine sort merge for all SORT operations. Tape-only, disc-only, or internal-only sorts are possible, depending on record volume and environment. These sorts are specified through job control device assignments and, optionally, through PARAM statements in the job control stream. (See the sort/merge user guide/programmer reference manual, UP-8074 (current version).)

Other considerations are:

- Record size

  The maximum COBOL record size of 4092 characters may be sorted.

- Record format

  Record format may be fixed (F), ASCII (D), or variable (V). When variable-length records are to be sorted, the BIN size (subrecord size used for internal sort purposes) provided to OS/3 sort merge by the compiler will be the size of the smallest record described in the sort file description (SD).

  NOTE:

  If the USING/GIVING options of the SORT statement are used, the record format of the USING/GIVING files must agree with the SD record format.

- Storage allocation

  The compiler ensures that the object program obtains the minimum storage required for sorting by including a RES linker control statement in the generated output module. 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 module by an external reference (EXTRN) to label KE$ALP.

  NOTE:

  If the programmer inhibits the compiler generation of linker control statements (optional OUT=L (7.1.2)), he must construct linker control specifications to satisfy this area requirement.

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, all storage from KE$ALP to the end of the program storage is used by the sort/merge processor for internal processing. Additional storage can greatly increase the efficiency of the sort operation and should be allocated when possible.

- 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 or disc devices, but not both, may be assigned for this purpose through job control statements. Tapes are assigned using fixed sort file-names of SM01, SM02, . . ., SM06. If tapes are assigned, a minimum of three is required, and a maximum of six may be used. Disc devices (maximum of eight), which must contain system scratch area, are assigned using fixed sort file-names of DM01, DM02, . . .,DM08.

- Job control stream parameters

  When the sort is executed by the object program, the job control stream is examined for the presence of SORT // PARAM statements. Use of SORT job stream parameters allows the programmer to override or add to the parameters specified in the object program.

- Multiple sorts

  The OS/3 extended COBOL compiler does not restrict the programmer from using two or more SORT statements which refer to the same SD (sort file description), or from using a number of SORT statements which refer to different sort file descriptions. However, only one SORT statement may be active at any one time; multicycle sorting is not supported. 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 previously initiated sort operation, a system console message is displayed, and processing is terminated. (For a listing of system console messages, see the error message programmer/operator reference manual, UP-8076 (current version).)

- Merging

  American National Standard COBOL (1968) does not support a merge facility; consequently such a feature is not supported in the UNIVAC OS/3 COBOL compilers.

- Checkpointing

  Checkpoints will not be issued if a sort is active. (See 8.5.)

13. ASCII Tape Processing

13.1. GENERAL

When the user requests it, the SPERRY UNIVAC Operating System/3 (OS/3) COBOL compiler processes and produces ASCII tapes. Data management automatically translates the tapes to EBCDIC when reading and to ASCII when writing.

13.2. DECLARATION OF ASCII FILES

ASCII files must be declared to the compiler by the APPLY ASCII* ON file-name clause. A mix of ASCII and non-ASCII files is permitted in the COBOL program.

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 tape file that contains or receives ASCII data (4.3.2).

2. 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.

3. The BLOCK-LENGTH-CHECK option applies only to files with a RECORDING MODE IS D clause. When specified, input data blocks are assumed to possess a 4-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 4-character buffer offset area.

** Extension to American National Standard COBOL (1968).*

13.3. RECORDING MODE* CLAUSE

Format:

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

Rules:

1. The RECORDING MODE clause is expanded to include the specification of D-type records (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 because, for ASCII files, D and V are synonymous. The D mode is provided for compatibility with other implementors.

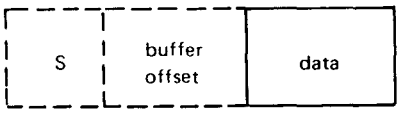
4. The RECORDING MODE IS D clause may be specified for ASCII tape files which contain variable-length records. An option within the APPLY ASCII ON file-name clause allows the specification of a buffer offset for any tape input file or the activation of the block length check feature on tape files with RECORDING MODE D.

NOTE:

Figure 13–1 and Table 13–1 show the physical tape formats and characteristics.
Table 13–2 lists the ASCII/EBCDIC conversions.

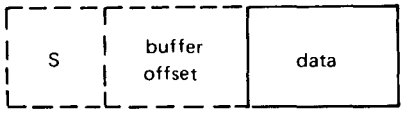
* *Extension to American National Standard COBOL (1968).*

U FORMAT RECORDS

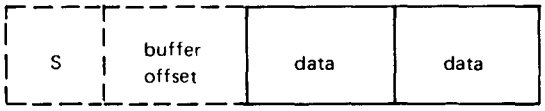


F FORMAT RECORDS

UNBLOCKED



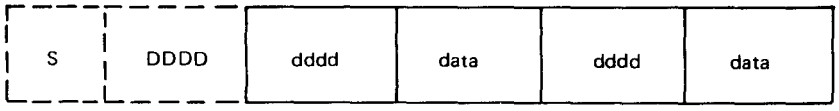
BLOCKED



D FORMAT RECORDS

UNBLOCKED

BLOCKED



S — Optional 1-character block sequence indicator whose presence is specified by the APPLY BLOCK-COUNT clause.

buffer offset — Optional field at the front of each input data block. Offset may be 0 to 99 characters in length. This area cannot be referenced by program nor can it be created on output files; presence specified by the APPLY ASCII WITH BUFFER-OFFSET OF integer CHARACTERS clause.

DDDD — Optional block length field in an implified buffer offset area of four characters. Block length is created and validated by data management programs. This option is specified by the APPLY ASCII BUFFER- OFFSET FOR BLOCK-LENGTH-CHECK clause.

dddd — Record length.

S, DDDD, dddd are all in ASCII decimal format.

Figure 13–1. ASCII Physical Tape Formats

Table 13–1. Characteristics of Tape Files Available to COBOL Users

Recording Mode Is	File Declared As	Label Records Specifications	Apply Buffer-Offset		Apply Block-Length-Check	Apply Block-Count
			Input	Output		
D blocked or unblocked	EBCDIC					
	ASCII	STANDARD (2) data-name (3)	0 to 99	(4)	Optional (4)	Optional (5)
F blocked or unblocked	EBCDIC	OMITTED STANDARD (1) data-name (3)				Optional
	ASCII	STANDARD (2) data-name (3)	0 to 99			Optional (5)
U	EBCDIC	OMITTED STANDARD (1) data-name (3)				Optional
	ASCII	STANDARD (2) data-name (3)	0 to 99			Optional (5)
V blocked or unblocked	EBCDIC	OMITTED STANDARD (1) data-name (3)			Automatic	Optional
	ASCII					

NOTES:

(1) De facto standard as defined by the data management system user guide, UP-8068 (current version)

(2) American National Standard COBOL (1968)

(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 1-character cyclic block sequence indicator (input files only)

Table 13–2. ASCII/EBCDIC Conversion (Part 1 of 3)

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 13–2. ASCII/EBCDIC Conversion (Part 2 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 13–2. ASCII/EBCDIC Conversion (Part 3 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.

PART 4. DEBUGGING AIDS

14. Debugging Language

14.1. GENERAL

The source program debugging statements, READY TRACE, RESET TRACE, EXHIBIT, and *DEBUG in SPERRY UNIVAC Operating System/3 (OS/3) COBOL are extensions to American National Standard COBOL (1968).

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 1-line paper advance. ←

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 (14.5).

14.2. 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.

14.3. RESET TRACE

Function:

The execution of the RESET TRACE statement terminates the functions initiated by READY TRACE and produces the following output:

TRACE OFF AT line-number

Format:

RESET TRACE.

Rule:

This statement may appear anywhere in the procedure division or in a debugging packet.

14.4. EXHIBIT

Function:

The execution of the EXHIBIT statement results in a formatted display of identifiers or nonnumeric literals listed in the statement.

Format:

```
EXHIBIT { CHANGED            } { identifier-1    }
        { CHANGED NAMED      } { nonnumeric-literal-1 }
        { NAMED              }
        [, { identifier-n    } ... ].
           { nonnumeric-literal-n }
```

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:

   - Identifier

     identifying-name△equal-sign△identifier-value△

     The identifying-name includes qualifiers and subscripts. A maximum of 130 characters is 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.

- Nonnumeric-literal

  nonnumeric-literal

7. The CHANGED NAMED option produces a noncolumnar display of 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 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 has changed, one line of space is displayed. The operands are displayed in the order in which they appear in the statement and in the following format:

   - Identifier

     identifier-valueΔ

     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.

   - Nonnumeric literals

     nonnumeric-literalΔ

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 the 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 options.

14.5. 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 with the following format:

1	8
*DEBUG	location

Location refers to a section or paragraph name which starts anywhere within margin A; a period is not permitted immediately following location. 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, which 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, 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 control stream. If *DEBUG cards are in the job control stream, they are processed as if they were contained at the end of the library module. If no *DEBUG cards are present in the job control stream, the process of reading COBOL input to the compiler is terminated.

Example:

```
// EXEC COBOL,library-name
// PARAM IN = PROGNAME/LIBIN
// PARAM LST = (O,C,S)
/$
*DEBUG ————————————————
  ————————————————————
  ————————————————————
*DEBUG ————————————————
  ————————————————————
  ————————————————————
/*
.
.
.
```

PART 5. APPENDIXES

Appendix A. Character Set

DECIMAL	HEXA-DECIMAL	EBCDIC	80-COLUMN CARD CODE	CONSOLE KEYBOARD CODE (EBCDIC)	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		
18	12		11-9-2		
19	13		11-9-3		BIT BIT
20	14	RES	11-9-4		
21	15	NL	11-9-5	CARR. RET(CR)	
22	16	BS	11-9-6		
23	17	IL	11-9-7		POSITION POSITION
24	18		11-9-8		
25	19		11-9-8-1		
26	1A		11-9-8-2		
27	1B		11-9-8-3		
28	1C		11-9-8-4		
29	1D		11-9-8-5		
30	1E		11-9-8-6		
31	1F		11-9-8-7		
32	20	DS	11-0-9-8-1		
33	21	SOS	0-9-1		0, 4,
34	22	FS	0-9-2		1, 5,
35	23		0-9-3		2, 6,
36	24	BYP	0-9-4		3. 7
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	Ⓢ (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.

DECIMAL	HEXA-DECIMAL	EBCDIC ②	80-COLUMN CARD CODE	CONSOLE KEYBOARD CODE (EBCDIC)
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	.
76	4C	<	12-8-4	<
77	4D	(	12-8-5	(
78	4E	+	12-8-6	+
79	4F	!	12-8-7	!
80	50	&	12	&
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	]
91	5B	$	11-8-3	$
92	5C	*	11-8-4	*
93	5D	)	11-8-5	)
94	5E	;	11-8-6	;
95	5F	^	11-8-7	^
96	60	-	11	-
97	61	/	0-1	/
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	| (Vert. Bar)	12-11	| (Vert. Bar)
107	6B	,(Comma)	0-8-3	,(Comma)
108	6C	%	0-8-4	%
109	6D	_(Underscore)	0-8-5	_(Underscore)
110	6E	>	0-8-6	>
111	6F	?	0-8-7	?
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	:
123	7B	#	8-3	#
124	7C	@	8-4	@
125	7D	'(Prime or Apos.)	8-5	'(Prime or Apos.)
126	7E	=	8-6	=
127	7F	"(Quotes)	8-7	"(Quotes)
128	80		12-0-8-1	
129	81	a	12-0-1	

② Lowercase letters are an industry standard and are not printable on the SPERRY UNIVAC Series 90 Printers without special print options.

NOTE:

Some graphic, card code, and hexadecimal assignments may differ depending upon the device, application, or installation policy.

DECIMAL	HEXA-DECIMAL	EBCDIC②	80-COLUMN CARD CODE	CONSOLE KEYBOARD SET (EBCDIC)
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 Lowercase letters are an industry standard and are not printable on the SPERRY UNIVAC Series 90 Printers without special print options.

DECIMAL	HEXA-DECIMAL	EBCDIC	80-COLUMN CARD CODE	CONSOLE KEYBOARD SET (EBCDIC)
192	C0	PZ	12-0	
193	C1	A	12-1	A
194	C2	B	12-2	B
195	C3	C	12-3	C
196	C4	D	12-4	D
197	C5	E	12-5	E
198	C6	F	12-6	F
199	C7	G	12-7	G
200	C8	H	12-8	H
201	C9	I	12-9	I
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	
209	D1	J	11-1	J
210	D2	K	11-2	K
211	D3	L	11-3	L
212	D4	M	11-4	M
213	D5	N	11-5	N
214	D6	O	11-6	O
215	D7	P	11-7	P
216	D8	Q	11-8	Q
217	D9	R	11-9	R
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	
225	E1		11-0-9-1	
226	E2	S	0-2	S
227	E3	T	0-3	T
228	E4	U	0-4	U
229	E5	V	0-5	V
230	E6	W	0-6	W
231	E7	X	0-7	X
232	E8	Y	0-8	Y
233	E9	Z	0-9	Z
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
241	F1	1	1	1
242	F2	2	2	2
243	F3	3	3	3
244	F4	4	4	4
245	F5	5	5	5
246	F6	6	6	6
247	F7	7	7	7
248	F8	8	8	8
249	F9	9	9	9
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	

Appendix B. Reserved Words

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*
CHARACTER*
CHARACTERS
CHANGED*
CLOSE
COBOL
COMMA
COMP
COMP-1*
COMP-2*
COMP-3*
COMP-4*
COMPUTATIONAL
COMPUTATIONAL-1*
COMPUTATIONAL-2*
COMPUTATIONAL-3*
COMPUTATIONAL-4*
COMPUTE
CONFIGURATION
CONTAINS
COPY
CORR
CORRESPONDING
CURRENCY
CYLINDER-INDEX*
CYLINDER-OVERFLOW*
DATA
DATE-COMPILED
DATE-WRITTEN
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DIRECT*
DISC*
DISC-8411*
DISC-8414*
DISC-8415*
DISC-8416*
DISC-8418*
DISC-8430*
DISC-8433*
DISPLAY
DIVIDE
DIVISION
DOWN
EBCDIC*
ELSE
END
ENDING
ENTER
ENTRY*
ENVIRONMENT
EQUAL
EQUALS*
ERROR
EVERY
EXAMINE
EXCEEDS*
EXHIBIT*
EXIT
EXTENDED
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

**Extensions to American National Standard COBOL (1968).*

OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION*
OTHERWISE*
→ OUK-90-250*
OUK-90-300*
→ OUK-90-400*
OUK-90-600*
OUK-90-700*
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
RESERVE
RESET
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-1*
SYSCHAN-2*
SYSCHAN-3*
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-5*
SYSERR-6*
SYSERR-7*
SYSERR-8*
SYSERR-9*
SYSERR-10*
SYSERR-11*
SYSERR-12*
SYSERR-13*
SYSERR-14*
SYSERR-15*
SYSERR-16*
SYSERR-17*
SYSERR-18*
SYSERR-19*
SYSERR-20*
SYSERR-21*
SYSERR-22*
SYSERR-23*
SYSERR-24*
SYSERR-25*
SYSERR-26*
SYSERR-27*
SYSERR-28*
SYSERR-29*
SYSERR-30*
SYSERR-31*
SYSIN*
SYSIN-96*
SYSIN-128*
SYSLOG*
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-9000*
UNIVAC-9025*
UNIVAC-9030*
UNIVAC-9040*
UNIVAC-9060*
UNIVAC-9070*
UNIVAC-9200II*
UNIVAC-9300*
UNIVAC-9300II*
UNIVAC-9400*
UNIVAC-9480*
UNIVAC-9700*
UNTIL
UP
UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
VERIFY*
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

Extension to American National Standard COBOL (1968).

Appendix C. Intermediate Results in Arithmetic Operations

C.1. GENERAL

For certain arithmetic statements in the SPERRY UNIVAC Operating System/3 (OS/3) COBOL, the COBOL compiler generates code that uses internal work areas for storage of intermediate results. Intermediate results may be required in the following types of statements:

- Add, where more than one operand precedes TO or GIVING.
- SUBTRACT, where more than one operand precedes FROM or GIVING.
- 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 use in the operation or series of operations for which it is required. The description can be expressed as a numeric PICTURE; however, an intermediate result used in the evaluation of an expression may contain as many as 30 digits.

If at least one floating-point (COMP-1 or COMP-2) or floating-point display or floating-point literal operand is used, the range of intermediate results is $\pm 5.4 * 10^{-79}$ to $\pm 7.2 * 10^{75}$; the remainder of this appendix is applicable only to nonfloating-point operands.

C.2. ADD AND SUBTRACT STATEMENTS

The description of the intermediate result area is determined by forming the composite of operands (6.6.1.1) and appending one additional digit in the most significant position to contain overflow when 10 or fewer operands immediately follow the verb, or two digits for more than 10 operands.

C.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 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 $10^{mag}-10^{-pl}-1$

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
+,–	$pl_{ir} = \max(pl_{OP1}, pl_{OP2})$ $L_{ir} = \max(mag_{OP1}, mag_{OP2}) + pl_{ir} + 1$
*	$pl_{ir} = pl_{OP1} + pl_{OP2}$ $L_{ir} = mag_{OP1} + mag_{OP2} + pl_{ir}$
/	$pl_{ir} = pl_{comp}$ $L_{ir} = pl_{OP2} - pl_{OP1} + L_{OP1} + pl_{ir}$
**	$pl_{ir} = 12$ $L_{ir} = 30$

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 readjusted. In these cases, $L_{ir} = 30$.

Appendix D. Compiler Diagnostics

D.1. GENERAL

The SPERRY UNIVAC Operating System/3 (OS/3) extended COBOL compiler generates system console and diagnostic messages during compilation. System console messages relate to the compilation environment and are displayed when the error condition is encountered. The job is terminated and the error condition must be corrected before the job can be rerun. The diagnostic messages flag errors encountered in the source program during compilation. A list of all diagnostic messages generated is output after all other printer options are satisfied.

D.2. DIAGNOSTIC MESSAGES

The diagnostic listing is produced as its last printed output. 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 diagnostic severity code definitions are:

P (precautionary)
: No source language error was detected, 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.

Table D–1 explains the error messages and related recovery procedures. The messages are listed in ascending order based on the message number.

Table D–1. Diagnostic Messages (Part 1 of 30)

Message Number	Severity Code	Diagnostic Message	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 was found in area A (columns 8 to 11) when continuation was 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 was 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 were detected following a left parenthesis.	In OS/3 COBOL, spaces must not separate left or right parentheses from that which they enclose.	Processing continues as if the space had not occurred.
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 that enclose a nonnumeric literal is not followed by a space or punctuation and a space.	The terminating quote or apostrophe enclosing a nonnumeric literal must be followed by a space or punctuation and a space.	Processing continues as if a space had occurred. The first 30 characters of the nonnumeric literal are noted in the diagnostic.

Table D–1. Diagnostic Messages (Part 2 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
007	C	EXCESSIVE CHARACTER STRING char-string.	A character string which is greater than its maximum legal size was detected.	Maximum legal sizes are: 132 characters for non-numeric literals, 20 characters for numeric literals (including sign and decimal point), 30 characters for nonliterals.	Processing continues after the excessive characters are discarded. The first 30 characters of the string are noted in the diagnostic.
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, e.g., P'CTURE.	The entire string is deleted.
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, e.g., #.	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	HYPHEN 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 non-numeric 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 RIGHT PARENTHESIS.	One or more spaces have been detected preceding right parenthesis.	In OS/3 COBOL, spaces must not separate left or right parentheses from that which they enclose.	Processing continues as if the space had not occurred.

Table D–1. Diagnostic Messages (Part 3 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
014	U	SYNTAX REQUIRES clause, 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 applicable language formats in this manual.	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. 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-1, COMP-2, COMP-3, COMP-4, DISPLAY, INDEX, SIZE, MAP, level-number WORKING-STORAGE, LINKAGE, PROCEDURE, Procedure-name in Area A, any verb.

Table D–1. Diagnostic Messages (Part 4 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
015	S	COMPILER ERROR	This diagnostic is issued only as the result of a compiler/system error.		The occurrence of this diagnostic should be reported using the SUR procedure.
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, 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 was not 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 eight characters of the SELECT file-name.	The RERUN entry is deleted.
018	U	clause PREVIOUSLY SPECIFIED FOR filename.	An entry, such as APPLY BLOCK-COUNT, was 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 source 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.
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.

Table D–1. Diagnostic Messages (Part 5 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
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 one alternate area, or none.	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, i.e., 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 E P R S V X Z space * , + – . ; ( ) or ".	The clause is deleted and the currency sign remains a $.
026	P	EXTERNAL-NAME external-name TRUNCATED	The external-name contains more than eight characters.	Only the first eight 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.

Table D–1. Diagnostic Messages (Part 6 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
028	C	CLAUSE CONFLICTS WITH ACCESS METHOD SPECIFICATION.	OPTIONAL and RESERVE are applicable only to disc files with ACCESS SEQUENTIAL and ORGANIZATION SEQUENTIAL.	See Section 11.	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.
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 main storage to process all of the SELECT entries.	Each SELECT entry requires 26 bytes of main 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 main 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.

Table D–1. Diagnostic Messages (Part 7 of 30)

Message Number	Severity Code	Diagnostic Message	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.	An invalid combination of ACCESS, ORGANIZATION, and KEY clauses has been specified.	The combinations of ACCESS, ORGANIZATION, and KEY clauses are invalid. See Section 11.	The file is classified as ACCESS SEQUENTIAL, ORGANIZATION SEQUENTIAL.
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.

Table D–1. Diagnostic Messages (Part 8 of 30)

Message Number	Severity Code	Diagnostic Message	Reason	Explanation 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, 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, 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 3, and 5 through 7, the clause is deleted. For rule 4, the first value is accepted; all others are deleted.
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.

*FD, RENAMES, and 66 available in extended compiler.

Table D–1. Diagnostic Messages (Part 9 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
043	U	clause NOT SUPPORTED	An obsolete COBOL clause has been encountered.	The SIZE clause is not within the OS/3 COBOL language.	The SIZE clause is deleted.
044	C	LEVEL NUMBER number MUST BEGIN 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, character string INVALID.	A COPY verb was not followed by a library name.	A library name: ■ is composed of no more than eight characters of the set A through Z, 0 through 9, and the hyphen (-). ■ has at least one alphabetic U character. ■ does not have a hyphen as the first or last character. ■ is not a COBOL reserved word.	The first eight characters of the string provided are used as a library name.
046	C	OCCURS CLAUSE INTEGER INVALID.	An OCCURS clause integer is 0 or greater than 65,535.	The minimum OCCURS value is 1. The maximum OCCURS value is 65,535. (In Format 2 of the OCCURS clause, integer-1 may be 0.)	If 0 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.
047	C	LIBRARY NAME character string EXCEEDS EIGHT CHARACTERS.	The library name following the COPY verb was found to be longer than eight characters.	The name of a library structure may be a maximum of eight characters.	The first eight 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 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.

Table D–1. Diagnostic Messages (Part 10 of 30)

Message Number	Severity Code	Diagnostic Message	Reason	Explanation Rule	Recovery
050	C	APPLY CLAUSE OR SEGMENT-LIMIT INTEGER INVALID.	Cylinder overflow of disc was specified as being greater than 80 percent. The buffer offset value is not from 0 to 99, or the SEGMENT-LIMIT value is not from 1 to 49.	Cylinder overflow percent may not be greater than 80 percent.	The overflow percent is set to 80 percent. The buffer offset is set at 99, or the SEGMENT-LIMIT is set at 49.
051	C	BLOCKING SPECIFIED WITH RECORDING MODE U.	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.
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; i.e., 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 was 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 was encountered prior to detection of a DATA DIVISION section 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.

Table D–1. Diagnostic Messages (Part 11 of 30)

Message Number	Severity Code	Diagnostic Message	Reason	Explanation Rule	Recovery
056	U	LANGUAGE ELEMENT NOT IMPLEMENTED.	A COBOL language feature not supported by the compiler has been encountered.	The following language elements are not available: I-O verbs in USE ERROR or LABEL procedure and ENTRY within a 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.
059	U	LEVEL 66 data-name MUST APPEAR ONLY AT END OF A HIERARCHY.	The level number 66 entry was not followed by one of the following: a level number 01 entry, an FD or SD entry, a level number 77 entry, a level number 66 entry, or a PROCEDURE DIVISION header.	See rules for RENAMES.	A level number 01 named FILLER is created to follow the level number 66 entry.

Table D–1. Diagnostic Messages (Part 12 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
060	U	OCCURS DEPENDING ASSOCIATED WITH data-name.	The data-name with the DEPENDING option of the OCCURS clause is not the last group entry in a 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 number between 02 and 49 follows a level number 77 or DATA DIVISION header.	See rules for level number.	A level number 01 named FILLER is created to precede the data entry.
062	U	CONSISTENCY ERROR: clause-1 INVALID WHEN USED WITH clause-2.	Conflict between description cluases of the data entry, e.g., USAGE COMP-3 and ALPHANUMERIC PICTURE.	See Section 5 for rules on clauses in conflict.	Clause-1 is deleted.
063	P	GO TO DEPENDING OPTION CONTAINS ONLY ONE PROCEDURE NAME.	At least two procedure names are required in a GO TO statement with the DEPENDING option.	See Format 2 of GO TO statement.	Control is transferred to procedure name if value of identifier is 1. Otherwise, control is passed to the next sentence.
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.
065	U	IMS ENVIRONMENT PROHIBITS USE OF LANGUAGE ELEMENT element.	The specified element is not allowed under IMS processing mode.	IMS mode requirement	The specified element is deleted.
066	U	PROCEDURE DIVISION USING-REQUIRED IN IMS ENVIRONMENT.	Procedure division USING must be present in the IMS environment.	The procedure division USING is the only allowable entry point in a COBOL program in the IMS environment.	No action is taken by the compiler.

Table D–1. Diagnostic Messages (Part 13 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
069	C	SAME SORT OR SAME RECORD AREA CONFLICTS WITH SAME AREA CLAUSE.	Some, but not all, filenames in a SAME AREA clause appear in a SAME RECORD or SAME SORT AREA clause.	If one or more filenames of a SAME AREA clause appears in a SAME RECORD or SAME SORT AREA clause; all the filenames in that SAME AREA clause must appear in the SAME SORT or SAME RECORD AREA clause.	No action is taken by the compiler.
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 group entry's usage as proper usage.
075	U	THE OCCURS CLAUSE ON data-name INVALID, 4 DIMENSIONAL 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. There must be a data record description for each file.	No action is taken by the compiler.

Table D–1. Diagnostic Messages (Part 14 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
077	C	BLOCK-LENGTH-CHECK CONFLICTS WITH RECORDING MODE FOR character-string.	BLOCK-LENGTH-CHECK is not allowed with all recording modes.	BLOCK-LENGTH-CHECK is appropriate with recording mode V or D only.	The BLOCK-LENGTH-CHECK is disregarded.
078	S	ADDITIONAL MEMORY REQUIRED FOR LABEL RECORDS PROCESSING.	There is not enough main storage available to hold all the label name definitions for this file.	N/A	Compiler assumes that label name definitions that will not fit into main storage do not exist. Main storage is required to hold the SELECTS and label name definitions. To allow processing of more label names, allocate more main storage, shorten the size of the SELECTS, or define fewer label names.
079	U	BLOCK CONTAINS CHARACTERS NOT A MULTIPLE OF RECORD SIZE FOR FILE filename.	A file with organization relative with an inconsistent blocking factor was encountered (blocking from BLOCK CONTAINS clause).	N/A	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.
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 with ORGANIZATION RELATIVE, and RECORDING MODE was V or U.	Device restriction (card reader) access method restriction (DISC, DISC-8414)	Compiler assumes recording mode F for this file.

Table D–1. Diagnostic Messages (Part 15 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
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 was encountered with a BLOCK CONTAINS clause specifying two or more records.	Device restriction.	Compiler assumes BLOCK CONTAINS one 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.
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.	See BLOCK CONTAINS.	The compiler assumes the minimum length for the BLOCK CONTAINS clause.
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 but the LABEL RECORDS clause specifies 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 with 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.

Table D–1. Diagnostic Messages (Part 16 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
091	U	COPY SYNTAX REQUIRES character-string, character-string INVALID.	The character-string listed as invalid has produced a syntax error. The required type of character-string is indicated.	See 6.6.7.1 for COPY verb rules.	The item in error and all items which follow it in the COPY clause are deleted.
092	S	REPLACING character-string OVERFLOW CAUSED BY character-string	The main storage area used to save replacing items has been exhausted or the number of qualifiers associated with an identifier has exceeded internal storage.	Compiler restriction.	The compiler ignores the balance of the clause which causes overflow. Recompile with additional main storage allocated to the compiler or reduce the number of items, amount of qualification, or size of names in the REPLACING clause.
093	C	Sign condition test requires figurative constant ZERO; literal 0 is invalid.	Literal 0 invalid unless preceded by a relational operator.	When testing the condition of a data item for a 0 condition. Syntax requires the use of figurative constant ZERO when no conditional operator is present in the test.	Literal constant 0 is treated as figurative constant ZERO with code being generated as if statement was written: IF DATA-NAME ZERO.
094	C	CHARACTER NUMBER literal IS INVALID IN type PICTURE picture-string.	An invalid PICTURE character, a PICTURE character inconsistent with the PICTURE type, or a violation of the PICTURE precedence rules was detected.	See Section 5 for the allowable PICTURE symbols and the rules for their usage.	In order 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 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 invalid PICTURE character, a PICTURE character inconsistent with the PICTURE type, or a violation of the PICTURE precedence rules was 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.

Table D–1. Diagnostic Messages (Part 17 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
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.
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 0 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 0 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 were detected 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.
104	P	LITERAL literal-string TRUNCATED DURING MOVE.	The 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 field.	The literal is moved and truncated.
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 than can fit into the data item.	The excess characters are truncated.

Table D–1. Diagnostic Messages (Part 18 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
106	U	INVALID POSITIONING OF KEY data-name IN HIERARCHY.	There must not be any item with an OCCURS clause between the table item and its keys.	See rules for KEY 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.	Not enough main storage is available to contain all entries subordinate to the 01 data entry, and too many entries for the 01 hierarchy for the amount of main storage allocated.		The compiler does not process the data entries not contained in main storage. To compensate, shorten the hierarchy, shorten names in data entries, or assign more main storage 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 contain a REDEFINES clause.
109	C	data-name HAS 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.
110	S	ADDITIONAL MEMORY REQUIRED TO PROCESS RENAMES QUALIFIER.	Insufficient main storage is available to contain the RENAMES qualifier because of 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 detected.	See data definition.	The compiler assumes the length to be 4092 for the elementary item.

Table D–1. Diagnostic Messages (Part 19 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
115	C	SIZE OF WORKING-STORAGE GROUP ITEM data-name EXCEEDS MAXIMUM OF 65,535.	A group entry in working-storage is 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.
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 a level number on a data entry directly subordinate to the 01, e.g., 01 A LEVEL 02 MISSING 05 B 02 C INVALID LEVEL STRUCTURE
118	C	THE FIRST OBJECT OF THE 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.)
120	C	USAGE INDEX INVALID FOR CONDITIONAL VARIABLE data-name.	A condition name entry is defined for a data entry with a USAGE IS INDEX clause.	See rules for condition name.	The compiler assumes the alphanumeric usage for the conditional variable.
121	C	RECORD data-name SIZE UNEQUAL TO 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.	OS/3 label specification has a length of 80 for labels.	The compiler assumes the length of label records to be 80.

Table D–1. Diagnostic Messages (Part 20 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
123	U	data-name NOT ALIGNED.	The data-name is the subject of a REDEFINES clause and requires alignment due to a SYNC clause. However, the object of the REDEFINES is not aligned.	See rules for SYNCHRONIZED.	The SYNCHRONIZED clause is ignored.
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.
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 to which the file is assigned.	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.

Table D–1. Diagnostic Messages (Part 21 of 30)

Message Number	Severity Code	Diagnostic Message	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 REDEFINES.	The compiler assumes the REDEFINES clause does not exist.
132	U	REDEFINES OBJECT, data-name, AND SUBJECT, data-name, UNEQUAL LEVEL NUMBER.	The object and subject 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 256th specified index-name.		The compiler starts index-name main storage assignment over and reassigns the main storage 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 hierarchy as the specified object of the RENAMES clause.
136	C	OBJECT OF RENAMES data-name HAS ILLEGAL LEVEL NUMBER.	The object of the RENAMES clause has an 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 ENTRY 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 exit.
138	S	ADDITIONAL MEMORY REQUIRED FOR PROCEDURE NAME PROCESSING.	The compiler needs more main storage in order to process the rest of the section and paragraph names.	Each procedure-name definition requires 16 bytes of storage plus one 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 main storage assigned to the compiler.	This procedure-name definition and all others that follow are deleted.

Table D–1. Diagnostic Messages (Part 22 of 30)

Message Number	Severity Code	Diagnostic Message	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.
143	U	STRUCTURE OF CONDITIONAL SENTENCE INVALID, UNPAIRED ELSE ENCOUNTERED.	ELSE encountered in IF statement with no preceding IF verb to match it.	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. The statement following the EXIT sentence is executed before the EXITs statements.
146	C	THE BEFORE OPTION OF THE USE STATEMENT IS NOT APPLICABLE IN SYSTEM.	The BEFORE option is not allowed in SPERRY UNIVAC OS/3 COBOL.	The BEFORE option is not applicable to SPERRY UNIVAC OS/3 COBOL, but is accepted for compatibility	The AFTER option is assumed.

Table D–1. Diagnostic Messages (Part 23 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
147	C	THE PROGRAM NAME IN CALL STATEMENT EXCEEDS EIGHT CHARACTERS.	Program name exceeds eight characters in length.	A maximum of eight characters is allowed in subprogram names.	The program name in the CALL statement is truncated to eight characters.
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 nonunique 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.
151	U	REFERENCE TO name OF name CANNOT BE RESOLVED DUE TO DEFINITION AT LINE literal. name of name UNRESOLVED DUE TO DEF 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 implies 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.

Table D–1. Diagnostic Messages (Part 24 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
152	C	REFERENCE TO name AMBIGUOUS DUE TO DEF AT LINE literal, DEF 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.
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.
157	P	name STATEMENT OPERAND name IS IMPROPERLY INDEXED	Index name used to address table element is not associated with the table but is associated with another table which has the same element size.	When an item is indexed by an index name, that index name must be associated with that table name.	Precautionary warning. No corrective action is taken.

Table D–1. Diagnostic Messages (Part 25 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
158	P	verb CONTAINS WORKING-STORAGE OPERAND data-name WHICH SHOULD NOT BE MODIFIED	Due to the shared nature of programs operating under IMS mode, errors could occur if working-storage elements are modified at object program execution time.	Do not modify WORKING STORAGE operands in the IMS environment.	No action. Precautionary warning.
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 subscripts, too few, or an improper type of subscript.	References to items in a table must have the correct number of subscripts or indexes, subnumeric integers, subscripts must be unsigned, subscripts and indexes must not be moved 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.
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.

Table D–1. Diagnostic Messages (Part 26 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
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 in the 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.
166	U	verb STATEMENT OPERAND data-name IS NOT LEVEL NUMBER 01 OR 77.	An operand with a level number other than 01 or 77, has been detected in an ENTRY or procedure division USING statement.	Data items in an 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 verb.	This statement exceeds the internal main storage area available to process statements with multiple operands.	The main 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 main storage available for statement processing is a function of the total main 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 main storage should be assigned to the compiler or the statement must be rewritten as multiple statements.
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.

Table D–1. Diagnostic Messages (Part 27 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
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.
171	U	PERIOD ELSE OR WHEN MUST FOLLOW NEXT SENTENCE	NEXT SENTENCE must be followed by ELSE, period, or WHEN.	In an IF, NEXT SENTENCE must be followed by ELSE or a period. In a SEARCH, NEXT SENTENCE must be followed by WHEN, ELSE, or a period.	The NEXT SENTENCE phrase is ignored.
172	P	PERFORM STATEMENT REFERENCES A NON-DECLARATIVE PROCEDURE	A PERFORM within the declarative section referenced a procedure outside of the declarative section.	Within a USE procedure, there must not be any reference to any non-declarative procedures.	No action. Precautionary warning.
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.
175	P	COMPARISON FOR EQUALITY MAY BE MEANINGLESS FOR A FLOATING POINT OPERAND.	A floating-point operand in a relational condition may cause the two operands not to be exactly equal.	No rule has been violated. Message is strictly informative.	Expected results may not occur at execution time.
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.

Table D–1. Diagnostic Messages (Part 28 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
177	U	verb STATEMENT CONFLICTS WITH SEGMENTATION RULES.	A procedure-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	U	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 nonnumeric literal must be consistent with the class of the receiving item.	The statement is deleted.
181	P	data-name TRUNCATED DURING MOVE.	The data-name 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 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 statement is deleted.

Table D–1. Diagnostic Messages (Part 29 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
183	P	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.
185	U	FILE-NAME IN insert STATEMENT REQUIRES SYMBOLIC KEY	The file-name referenced by the verb requires the SYMBOLIC KEY clause under the SELECT clause.	For ORGANIZATION INDEXED files, if ACCESS is SEQUENTIAL or EXTENDED, a symbolic key is required for the SEEK verb. If ACCESS is EXTENDED, a symbolic key is required for a READ that does not have the AT END clause.	The record key is used.
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 three in the conversion mode.	The advancing line count is set to 1.

Table D–1. Diagnostic Messages (Part 30 of 30)

Message Number	Severity Code	Diagnostic Message	Explanation		
			Reason	Rule	Recovery
188	U	FILE AT LINE literal NOT {CLOSED / OPENED} 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.
189	U	verb STATEMENT PROHIBITED WITHIN USE PROCEDURE.	The only I-O verbs allowed in a USE procedure are: ACCEPT (not from system console or job control stream) DISPLAY WRITE (to a printer only in USE FOR FORM OVERFLOW)	See rules for USE verb.	The I-O verb is dropped.
190	S	ADDITIONAL MAIN STORAGE REQUIRED TO PRODUCE OBJECT CODE LISTING	The compiler does not have sufficient main storage to produce the object code listing.		The object module is produced. Recompilation is necessary with more main storage assigned to the compiler.
191	S	ADDITIONAL MEMORY REQUIRED TO PRODUCE OBJECT PROGRAM.	The compiler does not have sufficient main storage to maintain the compile time tables necessary to create the object module output for this program.		A recompilation is necessary with more main 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.
193	P	TRUNCATION OF FLOATING POINT OPERAND literal MAY OCCUR.	In any move from a floating-point operand to a nonfloating-point operand, the floating-point value may not be able to be represented exactly in fixed-point format.	No rule has been violated. Message is strictly informative.	Truncation may occur.

D.3. SYSTEM CONSOLE MESSAGES

During compilation, COBOL source programs may encounter an error condition as indicated by a system console message. All operator system console messages are listed and described in the error messages programmer/operator reference manual, UP-8076 (current version). The programmer system console messages, those that are directed to the programmer, are given in Table D–2. The messages are listed in ascending order based on the message number and include the meaning and the corrective action to be taken.

Table D–2. System Console Messages (Part 1 of 2)

Message Number	Diagnostic Message	Meaning	Corrective Action
CC01	INSUFFICIENT MEMORY	Insufficient main storage provided to accommodate the processor. The job step is terminated.	Allocate sufficient main storage and rerun the job.
CC02	LOAD ERROR	An error occurred while attempting to locate and load a job phase in the LOAD library. The job step is terminated.	Check the LOAD library to make sure that the phase is entered. If not, enter it and rerun. If the phase is in the library, contact your Sperry Univac customer representative.
CC04	PATCH s aaaa IGNORED, SIZE INVALID	A patch card format error has occurred in the control stream. The job step is terminated. where: s Is the segment number. aaaa Is the address where the error occurred.	Correct the card format and rerun.
CC05	PATCH s aaaa IGNORED, NO DELIMITER	A patch card format error has occurred in the control stream. The job step is terminated.	Correct the card format and rerun.
CC06	SNAP s aaaa IGNORED	A snap card format error has occurred in the control stream. The job step is terminated.	Correct the card format and rerun.
CC07	NO SOURCE PROGRAM	An error occurred when the end-of-file card was read prior to the first source card in the control stream. The job step is terminated.	Correct the control stream and rerun the job.
CC08	PARAM CARD ERROR	An error was detected in the PARAM card which specifies job options. The job step is terminated.	Correct the PARAM card and rerun.
CC10	SOURCE PROGRAM NOT FOUND	A program designated as existing on a library file cannot be found. The job step is terminated.	Mount the correct library file and rerun the job.
CC11	SOURCE LIBRARY FILE NOT ALLOCATED	COBOL compiler cannot access the library file designated as containing the COBOL source program. Job step is terminated.	Correct volume mounting or control stream error and rerun the job.

Table D–2. System Console Messages (Part 2 of 2)

Message Number	Diagnostic Message	Meaning	Corrective Action
CC12	I/O ERROR ON {filename / JOB-STRM / CORSCARD}. CODE=nnnn	For filename: An I/O error occurred on a work file, source, copy, or object module file. The 4-digit code is a copy of the error status field settings. For JOB-STRM: An I/O error occurred during job control stream processing. The 4-digit code is a copy of the control stream error code. For CORSCARD: An error was detected in the library update correction cards. CODE=0006 indicates error in the correction cards. CODE=0007 indicates error in the SEQ card. In all cases, the job step is terminated.	Rerun the job. If the error persists, contact your Sperry Univac customer representative.
CC13	COMPILER ERROR phase indication	An error has occurred while attempting to position a file, or attempting to process a phase. The job step is terminated. A storage dump is provided.	Submit a Software User Report (SUR).
C14	COPY LIBRARY MODULE module-name NOT FOUND	The source COBOL program has requested that a module be included from the copy library, and the module cannot be found. The job step is terminated.	Mount the correct library or correct the module-name reference and rerun the job.
CC15	COPY LIBRARY FILE filename NOT ALLOCATED	The source COBOL program has requested that a module be included from the copy library and the compiler cannot access the designated library file. The job step is terminated.	Correct the volume mounting or control stream error and rerun the job.
CC17	PRINTER NOT ASSIGNED	An error has occurred while attempting to open the print file PRNTR. The DVC and LFD statements are missing or incorrect. The job step is terminated.	Correct the control stream and rerun the job.
→ CC19	EXTENDED COBOL REQUIRES MICROLOGIC EXPANSION	The extended COBOL compiler requires the micrologic expansion feature (2K COS), but the standard (1K COS) microcode has been loaded. The job step is terminated.	Load the expanded microcode (2K COS) or use the Basic COBOL compiler.

Appendix E. Compiler Listings

E.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 when the supervisor is loaded. The page heading of the source code listing locates the following information:

(1) LINE NO.

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.

(2) C or R

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 letter R is used to indicate lines on which word substitution was made.

(3) SEQ

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.

(4) SOURCE STATEMENT

The text (card columns 7 to 72) of the COBOL source program is listed.

(5) IDEN.

Under IDEN., program identification information (card columns 73 to 80) is listed. This is an optional entry made by the programmer to provide identification or card deck information. The compiler takes no action upon it.

(6) PAGE

Page number associated with compilation listing.

A sample source code listing is shown in Figure E–1.

```
COMPILED BY UNIVAC OS/3E COBOL COMPILER VERSION   07.00/02   DATE 79/07/26   TIME 01.05.43

// PARAM IN=TBL3T

// PARAM LST=(S,L,O,C,D,X,A)

SOURCE CREATION DATE   00/01/74  TIME  10.03
(1)LINE NO. (3)SEQ.        (4)SOURCE  STATEMENT                              (5)IDEN.          (6)PAGE  00001

    00001           IDENTIFICATION DIVISION.
    00002           PROGRAM-ID. TBL3T.
    00003           REMARKS. THIS IS A TEST PROGRAM TO VALIDATE COMPILER PROCESSING
    00004               OF VARIABLE TABLES, VARIABLE GROUP ITEMS AND THE SEARCH VERB.
    00005           ENVIRONMENT DIVISION.
    00006           CONFIGURATION SECTION.
    00007           SOURCE-COMPUTER. UNIVAC-9400.
    00008           OBJECT-COMPUTER. UNIVAC-9400.
    00009           SPECIAL-NAMES. SYSLST IS P.
    00010           INPUT-OUTPUT SECTION.
    00011           FILE-CONTROL.
    00012               SELECT A ASSIGN TO TPFIL1 TAPE.
    00013               SELECT B ASSIGN TO TPFIL2 TAPE.
    00014           I-O-CONTROL.
    00015               SAME RECORD AREA FOR A, B.
    00016           DATA DIVISION.
    00017           FILE SECTION.
    00018           FD  A
    00019               RECORDING MODE V
    00020               BLOCK CONTAINS 2000 CHARACTERS
    00021               LABEL RECORDS ARE OMITTED
    00022               DATA RECORDS ARE AA, AB.
    00023           01  AA.
    00024               03  AAA  PIC S999 COMP.
    00025               03  AAB.
    00026                   05  AABA OCCURS 0 TO 100 TIMES DEPENDING ON AAA,
    00027                       INDEXED BY AABAX1, AABAX2, AABAX3,
    00028                       ASCENDING KEY AABAB,
    00029                       DESCENDING KEY AABAA IN AAB OF AA,
    00030                       ASCENDING KEY AABAC.
    00031                        07  AABAA  PIC S99.
    00032                        07  AABAB  PIC S99.
    00033                        07  AABAC  PIC S99.
    00034           01  AB.
    00035               03  ABA  PIC S999 COMP.
    00036               03  ABB.
    00037                   05  ABBA OCCURS 100 TIMES
    00038                       INDEXED BY ABBAX1.
    00039                        07  ABBAA  PIC S99.
    00040                        07  ABBAB  PIC S99.
    00041                        07  ABBAC  PIC S99.
    00042           FD  B
    00043               RECORDING MODE U
    00044               LABEL RECORDS ARE OMITTED
    00045               DATA RECORD IS BA.
    00046           01  BA.
```

Figure E–1. Example of Source Code Listing

E.2. DATA DIVISION STORAGE MAP AND CROSS-REFERENCE LISTING

The storage map heading line contains the PROGRAM-ID name, the compiler version, and the date and time of compilation.

The page heading locates the following information:

① LINE

The compiler-generated line number on which the data item is defined.

② LEVEL

The level indicator or level number assigned to the item. An * indicates that the item was generated by the compiler, as with TALLY.

③ DATA-NAME

The name of the item.

④ 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.

⑤ DISP

The displacement of the item relative to the address contained in the item's cover register. The number is expressed in hexadecimal.

⑥ 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.

⑦ LENGTH

The length in bytes of the item.

⑧ TYPE

The class or type of the item where:

GP	Group
A/N	Alphanumeric
A	Alphabetic
NUP	Numeric unpacked
IDN	Index-data-name
IDX	Index-name
AE	Alphabetic edited
NE	Numeric edited
NP	Numeric packed
VGP	Variable group
B	Binary (USAGE COMP, USAGE COMP-4)
FPD	Floating-point display
LFP	Long floating point (USAGE COMP-2)
SFP	Short floating point (USAGE COMP-1)

⑨ 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, e.g., –5 for PIC PP999 or PIC 9.99999 or PIC 99V99999.

+ integer indicates the number of trailing P's in the PICTURE, e.g., +5 for PIC 99P(5)

⑩ OCC

The number of occurrences of the item as specified by the OCCURS clause.

⑪ LINE NUMBERS OF REFERENCES

If the cross-reference list has been specified, the line numbers where one or more procedural references to the item were made are listed here.

Figure E–2 is a sample data division storage map. The data division storage map is listed in ascending order according to line number. The information that is presented in the data division storage map may also be listed in alphabetical order based on the data names (7.1.1, the alphabetized cross-reference listings).

E.3. PROCEDURE DIVISION STORAGE MAP AND CROSS-REFERENCE LISTING

The storage map heading line will contain the PROGRAM-ID name, the compiler version, and the date and time of compilation.

The page heading locates the following information:

① LINE
The compiler-generated line number on which the item is defined.

② SECTION
If the item is a section-name, it is listed here.

③ PARAGRAPH
If the item is not a section-name, it is listed here.

→ ④ PRIOR
The priority number of the section-name.

⑤ 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.

⑥ GO TO
An E indicates that the procedure is the object of a GO TO.

⑦ PERFORM

⑦a ENTRY
An E indicates that the procedure is the object of a PERFORM.

⑦b EXIT
An X indicates that the procedure contains a PERFORM exit point.

```
PROGRAM-ID. TBL3T     COMPILED  BY  UNIVAC  OS/3E COBOL  COMPILER       VERSION  07.00/02 DATE  79/07/26   TIME  01.05.43

                                        DATA DIVISION MEMORY MAP                                          PAGE  00068

LINE  LEVEL          DATA NAME          REG DISP   ADDR   LENGTH  TYPE PTLOC  OCC      LINE NUMBERS OF REFERENCES
 (1)   (2)             (3)              (4)  (5)    (6)     (7)    (8)   (9)   (10)                        (11)

      * * * SPECIAL NAMES * * *
00009 77     P                                                                        00452 00454 00462 00464 00473
                                                                                      00475

      * * * FILE SECTION * * *
*****   *    TALLY                      *   000C  000190      3   NP

00018 FD     A                                                                        00152 00420 00421 00426 00431
00023 01     AA                         5   0004  001890    602   VGP               00022 00429
00024    03  AAA                        5   0004  001890      2   BIN               00026 00158 00166 00174 00182
                                                                                      00191 00199 00209 00218 00226
                                                                                      00238 00248 00251 00258 00261
                                                                                      00272 00282 00293 00302 00307
                                                                                      00315 00321 00361 00368 00375
                                                                                      00386 00398 00407 00419
00025    03  AAB                        5   0006  001892    600   VGP               00160 00168 00176 00184 00193
                                                                                      00201 00211 00220 00227 00239
                                                                                      00242 00249 00252 00259 00262
                                                                                      00273 00274 00275 00277 00284
                                                                                      00285 00286 00287 00295 00297
                                                                                      00305 00306 00308 00316 00322
                                                                                      00323 00363 00370 00371 00377
                                                                                      00380 00381 00388 00400 00402
                                                                                      00403 00409
00026    05  AABA                       5   0006  001892      6   GP          100   00161 00169 00177 00185 00202
                                                                                      00212 00221 00228 00243 00253
                                                                                      00263 00292 00313 00379 00389
                                                                                      00423
00026    *   AABAX1                     *   0018  0001A8      8   IDX               00423 00425 00440
00026    *   AABAX2                     *   002C  000180      8   IDX
00026    *   AABAX3                     *   0028  0001B6      8   IDX
00031    07  AABAA                      5   0006  001892      2   NUP
00032    07  AABAB                      5   0008  001894      2   NUP
00033    07  AABAC                      5   000A  001896      2   NUP
00034 01     AB                         5   0004  001890    602   GP                00022
00035    03  ABA                        5   0004  001890      2   BIN
00036    03  ABB                        5   0006  001892    600   GP                00155 00165 00173 00181 00190
                                                                                      00198 00208 00217 00225 00362
                                                                                      00369 00376 00387 00408 00410
                                                                                      00418 00422 00433 00437
00037    05  ABBA                       5   0006  001892      6   GP          100
00037    *   ABBAX1                     *   003C  0001C0      8   IDX
00039    07  ABBAA                      5   0006  001892      2   NUP
00040    07  ABBAB                      5   0008  001894      2   NUP
00041    07  ABBAC                      5   000A  001896      2   NUP

00042 FD     B                                                                        00152 00435 00436 00443 00448
00046 01     BA                         6   0000  001888    602   VGP               00045 00446
00047    03  BAA                        6   000C  001888      2   BIN               00050 00434
```

Figure E–2. Example of Data Division Storage Map and Cross-Reference Listing

(8) ALTER

An A indicates that the procedure is altered.

(9) SORT

(9a) ENTRY

An E indicates that the procedure is the entry point of a SORT procedure.

(9b) EXIT

An X indicates that the procedure contains a SORT procedure exit point.

(10) 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. A sample procedure division storage map is shown Figure E–3.

The procedure division storage map is listed in ascending order according to priority number above SEGMENT-LIMIT, and within priority number in ascending order according to line number. The information that is presented in the procedure division storage map may also be listed in alphabetic order based on the procedure names (7.1.1, the alphabetized cross-reference listings).

E.4. OBJECT CODE LISTING AND EXTERNAL REFERENCES

The object code listing heading line contains the compiler version number and the date and time of the compilation.

Following the report heading 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 2-character ESID number (External Symbol Identification) 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 first entry in the list is the program name and its ESID number of 02. The program name is the PROGRAM-ID name from the identification division. If the COBOL program is segmented, the segment names follow. The 8-character segment name is composed of the first six characters of the program-name and a 2-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 identifies various external programs required in the execution of the COBOL program, such as the data management modules and special COBOL object time subroutines.

The symbols in the last group are names that appear in CALL statements.

The object code listing page heading 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, either no cover is needed for the line of code, or the cover register assignment at object time varies and cannot be defined.

```
                                           PROCEDURE DIVISION MEMORY MAP                                      PAGE  00071

 (1)      (2)                               (4)    (5)  - - - - - VERB REFERENCES - - - - -
 LINE     SECTION NAME                      PRIOR  ADDR GO   PERFORM     ALTER   SORT (9)  DEBUG
            PARAGRAPH NAME                            (6) TO ENTER EXIT   (8)   ENTER EXIT  (10)      LINE NUMBERS OF REFERENCES
               (3)                                            (7a)  (7b)         (9a)  (9b)
         * * * * * DEPENDING SECTION  * * * * *
00150      DP$00000                                002738      E    X
00150      DP$00001                                002752      E    X
00150      DP$00002                                00276C      E    X
         * * * * END DEPENDING SECTION  * * * *
00151      0010                                    002786
00154      1000-MOVE-V-TO-F-PADDED                 0027A4
00164      1010-MOVE-V-TO-F-TRUNC                  00284C
00172      1020-MOVE-V-TO-F-JUST-PAD               0028F6
00180      1030-MOVE-V-TO-F-JUST-TRUNC             00299E
00189      1040-ZERO-LENGTH-TO-F                   002A48
00197      1050-MOVE-F-TO-V-PADDED                 002AE6
00207      1060-MOVE-F-TO-V-TRUNC                  002BAC
00216      1070-MOVE-F-TO-ZERO-LENGTH-V            002C5E
00224      1080-MOVE-FIGURATIVE-TO-V               002CFC
00232      1090-DECISION                           002D9C
00237      1100-MOVE-V-TO-V-PADDED                 002DB8
00247      1110-MOVE-V-TO-V-TRUNC                  002ED6
00257      1120-MOVE-V-TO-V-ZERO-LENGTH            002FF4
00266      2000-DECISION                           003106
00271      2010-IF-V-VS-LONGER-F                   003122
00281      2020-IF-V-VS-SHORTER-F                  003246
00291      2030-IF-ZERO-LENGTH-VS-F                00336A
00301      2040-IF-V-VS-V                          003442
00312      2050-IF-Z-VS-Z                          0035CA
00319      2060-IF-V-VS-FIGURATIVE                 003664
00330      2070-IF-ZERO-LENGTH-V-VS-FCON           0037A8
00337      2080-IF-V-ALPHABETIC                    00387C
00346      2090-IF-V-NUMERIC                       003952
00355      3000-DECISION                           003A28
00360      3010-EXAMINE-V-TALLY-ALL                003A44
00367      3020-EXAMINE-V-TALLY-REP-ALL            003AC4
00374      3030-EXAMINE-V-TALLY-REP-TIL            003B7A
00385      3040-EXAMINE-ZERO-LENGTH-V              003C84
00392      4000-DECISION                           003DCE
00397      4010-TRANSFORM-V                        003D2A
00406      4020-TRANSFORM-ZERO-LENGTH-V            003EC2
00417      5010-WRITE-V-REC-MODE-V                 003F54
00428      5020                                    00406A      E    X                                     00419
00430      5030                                    0040B0      E    X                                     00423
00432      5040-WRITE-V-REC-MODE-U                 004106 E                                               00427
00445      5050                                    00421E      E    X                                     00434
00447      5060                                    00425C      E    X                                     00438
00449      5070                                    0042B4 E                                               00444
00451      9000-SUMMARY                            0042B4 E                                               00477
00460      9000-DISPLAY                            004390      E    X                                     00457
00465      9010-PASS                               004408      E    X                                     00162 00170 00178 00187 00195
                                                                                                          00205 00214 00222 00230 00245
                                                                                                          00255 00264 00278 00288 00298
                                                                                                          00309 00317 00327 00335 00343
```

Figure E–3. Example of Procedure Division Storage Map and Cross-Reference Listing

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, displayed under COMMENTS, are used to locate the indicated items and areas.

(1) Intersegment GO TO Subroutine

Used when control is passed from one segment of a segmented program to another.

(2) Intersegment 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 half words.

(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.

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 TOs in priority segements.

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 DTFs 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 VCONs.

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: the value of TALLY is also stored in this area.

㉗ PERFORM n TIMES Counters

A table containing the counters for the PERFORM verb.

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 TOs 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.

A sample object code listing is shown in Figure E–4.

E.5. DIAGNOSTIC ERROR LISTING

The diagnostic listing header line contains the program-ID name, the compiler version, the date, and the time of compilation. The page headings locate the following information:

① LINE #
Compiler-generated number which identifies the particular line of source code with which it appears.

② SVC
Severity code letter

③ ERROR
Diagnostic number

④ DIAGNOSTIC MESSAGE
Brief explanation of the error condition

Explanation of the text of the diagnostic error listing is in Appendix D.

A sample diagnostic listing is shown in Figure E–5.

```
PROGRAM-ID. TBL3T   COMPILED BY UNIVAC OS/3E COBOL COMPILER  VERSION 07.00/02  DATE 79/07/26  TIME 01.05.43   PAGE 00010
                                    EXTERNAL REFERENCES

T3L3T000 02    CC@EXM94 03    CC@BJERR 04    CB@CAD   05    CS@ALPH  06    CS@NUMU  07    CS@ANRP  08    CB@ANR   09
CM@VMOVE 0A    DD$T111  0B    CC@BJER3 0C    CF@DCP   0D    CW@TRANS 0E    CW@EB2AS 0F    CB@OPCL3 10

     LINE #     BASE/DISPL    ADDRESS      CONTENTS OF MEMORY     OPERAND ADDRESSES      OPCODE      COMMENTS

                              000000       05 F0                                         BALR
                  F  000      000002       45 E0 F 006                 000008            BAL
                  F  004      000006       07 FC                                         BCR
                  F  006      000008       98 AD F 012                 000014            LM
                  F  00A      00000C       98 59 A CFC                 000124            LM
                  F  00E      000010       04 A0                                         SPM
                  F  010      000012       07 FE                                         BCR
                  F  012      000014       00000028 02                                   DC A
                  F  016      000018       00001028 02                                   DC A
                  F  01A      00001C       00002786 02                                   DC A
                  F  01E      000020       000026E8 02                                   DC A
                  F  022      000024       00000000
                  A  000      000028       58 F0 1 000                                   L           (03)
                  A  004      00002C       41 00 0 C00                                   LA
                  A  008      000030       19 F0                                         CR
                  A  00A      000032       47 80 E 000                                   BC
                  A  00E      000036       58 C0 1 004                                   L
                  A  012      00003A       95 00 1 000                                   CLI
                  A  016      00003E       90 00 1 000                                   STM
                  A  01A      000042       47 D0 F C00                                   BC
                  A  01E      000046       45 E0 A C12                 00003A            BAL
                  A  022      00004A       0000
                  A  024      00004C       96 0F A 037                 00005F            OI          (04)
                  A  028      000050       4F 10 A 030                 000058            CVB
                  A  02C      000054       07 FE                                         BCR
                  A  02E      000056       0000
                  A  030      000058       0000000C0000000C                              DC X        (15)
                  A  038      000060                                                     DS    48    (16)
                  A  068      000090       000003E8                                      DC X        Y
                  A  06C      000094       00002734 02                                   DC A        (17)
                  A  070      000098       00002738 02                                   DC A        (21)
                  A  074      00009C       00002752 02                                   DC A
                  A  078      0000A0       0000276C 02                                   DC A
                  A  07C      0000A4       00004408 02                                   DC A
                  A  080      0000A8       0000442A 02                                   DC A
                  A  084      0000AC       000044A2 02                                   DC A
                  A  088      0000B0       0000406A 02                                   DC A
                  A  08C      0000B4       00004080 02                                   DC A
                  A  090      0000B8       00004108 02                                   DC A
                  A  094      0000BC       0000421E 02                                   DC A
                  A  098      0000C0       0000425C 02                                   DC A
                  A  09C      0000C4       00004264 02                                   DC A
                  A  0A0      0000C8       0000444C 02                                   DC A
                  A  0A4      0000CC       00004390 02                                   DC A
                  A  0A8      0000D0       00004284 02                                   DC A
                  A  0AC      0000D4       00000210 02                                   DC A        (22)
                  A  0B0      0000D8       00000210 02                                   DC A
                  A  0B4      0000DC       00000308 02                                   DC A
```

Figure E–4. Example of Object Code Listing and External References

```
PROGRAM-ID TBL3T    COMPILED BY UNIVAC OS/3E COBOL COMPILER VERSION 07.00/02 DATE 79/07/26  TIME  01.05.43
 (1)  (2) (3)   (4)
 LINE SVC ERROR   DIAGNOSTIC MESSAGE                                                          PAGE  00077

 00160 P  181  AAB TRUNCATED DURING MOVE.
 00168 P  181  AAB TRUNCATED DURING MOVE.
 00176 P  181  AAB TRUNCATED DURING MOVE.
 00184 P  181  AAB TRUNCATED DURING MOVE.
 00193 P  181  AAB TRUNCATED DURING MOVE.
 00274 P  181  AAB TRUNCATED DURING MOVE.
          *****ERRORS   U-  0000    S-  0000    C-  0000    P-  0006   *****
          OS/3E COBOL COMPILATION COMPLETE TBL3T    START   01.05.43  END   01.07.59
```

Figure E–5. Example of Diagnostic Listing

E.6. ALPHABETICALLY ORDERED DATA DIVISION CROSS-REFERENCE LISTING

This listing presents the same information as the data division storage map (Figure E—2), but the items are presented in ascending sequence by data-names.

Figure E—6 is a sample alphabetically ordered data division cross-reference listing.

E.7. ALPHABETICALLY ORDERED PROCEDURE DIVISION CROSS-REFERENCE LISTING

This listing presents the same information as the procedure division storage map (Figure E—3), but the items are listed in ascending sequence by procedure-names.

Figure E—7 shows a sample alphabetically ordered procedure division cross-reference listing.

```
PROGRAM-ID. TBL3T     COMPILED BY UNIVAC OS/3E COBOL COMPILER     VERSION 07.00/02 DATE 79/07/26   TIME 01.05.43

                                DATA DIVISION CROSS REFERENCE                                    PAGE 00073

        DATA NAME          LEVEL  LINE   REG  DISP  ADDR   LENGTH  TYPE PTLOC OCCURS    LINE NUMBERS OF REFERENCES

A                           FD   00018
AA                          01   00023    5  0004  001890   602   VGP              00152 00420 00421 00426 00431
AAA                         03   00024    5  0004  001890     2   BIN              00022 00429
                                                                                   00026 00158 00166 00174 00182
                                                                                   00191 00199 00209 00218 00226
                                                                                   00238 00248 00251 00258 00261
                                                                                   00272 00282 00293 00302 00307
                                                                                   00315 00321 00361 00368 00375
                                                                                   00386 00398 00407 00419
AAB                         03   00025    5  0006  001892   600   VGP              00160 00168 00176 00184 00193
                                                                                   00201 00211 00220 00227 00239
                                                                                   00242 00249 00252 00259 00262
                                                                                   00273 00274 00275 00277 00284
                                                                                   00285 00286 00287 00295 00297
                                                                                   00305 00306 00308 00316 00322
                                                                                   00323 00363 00370 00371 00377
                                                                                   00380 00381 00388 00400 00402
                                                                                   00403 00409
AABA                        05   00026    5  0006  001892     6   GP       100    00161 00169 00177 00185 00202
                                                                                   00212 00221 00228 00243 00253
                                                                                   00263 00292 00313 00379 00389
                                                                                   00423
AABAA                       07   00031    5  0006  001892     2   NUP
AABAB                       07   00032    5  0008  001894     2   NUP
AABAC                       07   00033    5  000A  001896     2   NUP
AABAX1                      *    00026    *  0018  0001A8     8   IDX              00423 00425 00440
AABAX2                      *    00026    *  0020  0001B0     8   IDX
AABAX3                      *    00026    *  0028  0001B8     8   IDX
AB                          01   00034    5  0004  001890   602   GP               00022
ABA                         03   00035    5  0004  001890     2   BIN
ABB                         03   00036    5  0006  001892   600   GP               00155 00165 00173 00181 00190
                                                                                   00198 00208 00217 00225 00362
                                                                                   00369 00376 00387 00408 00410
                                                                                   00418 00422 00433 00437
ABBA                        05   00037    5  0006  001892     6   GP       100
ABBAA                       07   00039    5  0006  001892     2   NUP
ABBAB                       07   00040    5  0008  001894     2   NUP
ABBAC                       07   00041    5  000A  001896     2   NUP
ABBAX1                      *    00037    *  0030  0001C0     8   IDX
B                           FD   00042                                              00152 00435 00436 00443 00448
BA                          01   00046    6  0000  001888   602   VGP              00045 00446
BAA                         03   00047    6  0000  001888     2   BIN              00050 00434
BAB                         03   00048    6  0002  00188A   600   VGP
BABA                        05   00049    6  0002  00188A     6   A/N      100    00438
BABAX1                      *    00049    *  0038  0001C8     8   IDX              00438
FAILURE                     88   00148    7  0A20  002510     1   NUP              00233 00267 00356 00393
P                           77   00009                                            00452 00454 00462 00464 00473
                                                                                   00475
TALLY                       *    *****    *  0000  000190     3   NP
WA                          01   00053    7  0000  001AF0   602   VGP
WAA                         03   00054    7  0000  001AF0     2   BIN              00056 00238 00241 00248 00258
```

Figure E–6. Example of Alphabetically Ordered Data Division Cross-Reference Listing

```
                                PROCEDURE DIVISION CROSS REFERENCE                                PAGE  00075

      PROCEDURE NAME          TYPE LINE      ADDR GO  PERFORM  ALTER   SORT   DEBUG
                                    PRI        TO ENTER EXIT    ENTER EXIT          LINE NUMBERS OF REFERENCES

DP$00000                      PAR  00150     002736     E    X
DP$00001                      PAR  00150     002782     E    X
DP$00002                      PAR  00150     00276C     E    X
0010                          PAR  00151     002786
1000-MOVE-V-TO-F-PADDED       PAR  00154     0027A4
1010-MOVE-V-TO-F-TRUNC        PAR  00164     00284C
1020-MOVE-V-TO-F-JUST-PAD     PAR  00172     0028F6
1030-MOVE-V-TO-F-JUST-TRUNC   PAR  00180     00299E
1040-ZERO-LENGTH-TO-F         PAR  00189     002A48
1050-MOVE-F-TO-V-PADDED       PAR  00197     002AE6
1060-MOVE-F-TO-V-TRUNC        PAR  00207     002BAC
1070-MOVE-F-TO-ZERO-LENGTH-V  PAR  00216     002C5E
1080-MOVE-FIGURATIVE-TO-V     PAR  00224     002CFC
1090-DECISION                 PAR  00232     002D9C
1100-MOVE-V-TO-V-PADDED       PAR  00237     002DB8
1110-MOVE-V-TO-V-TRUNC        PAR  00247     002ED6
1120-MOVE-V-TO-V-ZERO-LENGTH  PAR  00257     002FF4
2000-DECISION                 PAR  00266     003106
2010-IF-V-VS-LONGER-F         PAR  00271     003122
2020-IF-V-VS-SHORTER-F        PAR  00281     003246
2030-IF-ZERO-LENGTH-VS-F      PAR  00291     00336A
2040-IF-V-VS-V                PAR  00301     003442
2050-IF-Z-VS-Z                PAR  00312     00350A
2060-IF-V-VS-FIGURATIVE       PAR  00319     003664
2070-IF-ZERO-LENGTH-V-VS-FCON PAR  00330     00376A
2080-IF-V-ALPHABETIC          PAR  00337     00387C
2090-IF-V-NUMERIC             PAR  00346     003952
3000-DECISION                 PAR  00355     003A26
3010-EXAMINE-V-TALLY-ALL      PAR  00360     003A44
3020-EXAMINE-V-TALLY-REP-ALL  PAR  00367     003AC4
3030-EXAMINE-V-TALLY-REP-TIL  PAR  00374     003B7A
3040-EXAMINE-ZERO-LENGTH-V    PAR  00385     003C84
4000-DECISION                 PAR  00392     003D0E
4010-TRANSFORM-V              PAR  00397     003D2A
4020-TRANSFORM-ZERO-LENGTH-V  PAR  00406     003EC2
5010-WRITE-V-REC-MODE-V       PAR  00417     003F54
5020                          PAR  00428     00406A     E    X                00419
5030                          PAR  00430     0040B0     E    X                00423
5040-WRITE-V-REC-MODE-U       PAR  00432     0041C6 E                         00427
5050                          PAR  00445     00421E     E    X                00434
5060                          PAR  00447     00425C     E    X                00438
5070                          PAR  00449     00428A E                         00444
9000-DISPLAY                  PAR  00460     00439D     E    X                00457
9000-SUMMARY                  PAR  00451     00428A E                         00477
9010-PASS                     PAR  00465     004408     E    X                00162 00170 00178 00187 00195
                                                                              00205 00214 00222 00230 00245
                                                                              00255 00264 00278 00288 00298
                                                                              00309 00317 00327 00335 00343
                                                                              00352 00365 00372 00382 00390
                                                                              00404 00411 00425 00441
9020-FAIL                     PAR  00468     00442A     E    X                00163 00171 00179 00188 00196
```

Figure E–7. Example of Alphabetically Ordered Procedure Division Cross-Reference Listing

Appendix F. Conversion Mode

F.1. GENERAL

To facilitate the conversion of IBM/360 DOS COBOL level D to SPERRY UNIVAC Operating System/3(OS/3) COBOL, a conversion facility has been built into the OS/3 extended COBOL compiler. This facility, called the conversion mode (C-mode) accepts COBOL source code and alters it to American National Standard specifications, or issues diagnostics so the programmer is made aware of the need for changes.

F.2. CONVERSION MODE OPERATION

A PARAM statement option is available to energize the conversion mode of the OS/3 COBOL compiler.

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 compile successfully 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.

The conversion mode may sometimes assume the presence of a particular OS/3 COBOL clause in order to ensure the proper processing of a COBOL-D clause. An example of this technique is the fabrication of a SYNCHRONIZED clause for each appearance of a COBOL-D COMP, COMP-1, or COMP-2 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 90/30 System hardware limitation.

This appendix describes the known differences that exist between COBOL-D and UNIVAC OS/3 COBOL. It also defines the language differences that the conversion mode renders transparent. Those language differences for which no automatic software support is possible also are identified here, along with the appropriate source program change requirement.

When functioning in the conversion mode, many of the compiler American National Standard 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 OS/3 COBOL language capabilities without first being totally converted to American National Standard COBOL.

In the normal American National Standard 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 American National Standard mode; that is, control character print files are unique to the conversion mode.

F.3. CONVERSION MODE SYNTAX

The differences between COBOL-D and OS/3 COBOL are described in the following paragraphs within each program division.

F.3.1. Identification Division

- PROGRAM-ID. program-name.

  COBOL-D
  Program-name is one to eight characters enclosed in quotation marks.

  OS/3 COBOL
  Program-name is 1 to 30 characters not enclosed in quotation marks. Only the first 6 characters, excluding hyphens, are used to identify the object program.

  C-mode
  OS/3 accepts a 1- to 8-character name enclosed in quotation marks. Only the first 6 characters, excluding hyphens, are used to identify the object program.

F.3.2. Environment Division

- CONFIGURATION SECTION heading.

  COBOL-D
  Optional

  OS/3 COBOL
  Required

  C-mode
  Optional

- SOURCE/OBJECT COMPUTER clause.

  COBOL-D
  IBM-360 model-number

  OS/3 COBOL
  UNIVAC-9030

  C-mode
  The compiler accepts any SOURCE/OBJECT COMPUTER entries 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.

  OS/3 COBOL

  Reversal of decimal point and comma controlled by SPECIAL-NAMES entry.

  C-mode

  No automatic support. The converter must 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 automatically produces a special-names entry, internally, for the program. CONSOLE is equated to SYSCONSOLE, and SYSLST is equated to SYSLST.

- SELECT/ASSIGN clause

  COBOL-D

  ASSIGN TO 'external-name' { **DIRECT-ACCESS** / **UNIT-RECORD** / **UTILITY** }

  device-number UNIT(s)

  OS/3 COBOL

  ASSIGN TO 'external-name' integer implementor-name

  C-mode

  No automatic support. The COBOL-D SELECT statement, with respective ASSIGN clauses, must be replaced by the appropriate SELECT/ASSIGN clauses before compilation.

- ACCESS clause

  COBOL-D

  The word 'IS' is optional.

  OS/3 COBOL

  The word 'IS' is required.

  C-mode

  The word 'IS' is optional.

- KEY clauses

  COBOL-D

  The word 'IS' is optional.

  OS/3 COBOL

  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.

  OS/3 COBOL

  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 prior to compilation or diagnostics will result.

- 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 which 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.

  OS/3 COBOL

  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/3 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.

  OS/3 COBOL

  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.

  OS/3 COBOL
  The word 'ON' is required.

  C-mode
  The word 'ON' is optional.

- COPY library-name

  COBOL-D
  Library names are enclosed in quotation marks.

  OS/3 COBOL
  Library names are not enclosed in quotation marks.

  C-mode
  Library names are enclosed in quotation marks. All libraries are expected to be in OS/3 format.

F.3.3. Data Division

- 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-level items in the linkage section.

  OS/3 COBOL
  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
  The sterling currency feature may be specified by extensions to the PICTURE clause.

  OS/3 COBOL
  The sterling currency feature is not supported in UNIVAC -9030 COBOL.

  C-mode
  The sterling currency feature is not supported.

- COPY specifications

COBOL-D

The COPY statement is allowed on level-number 77 items in the working-storage and linkage sections.

OS/3 COBOL

The COPY statement is not allowed on level-number 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 OS/3 COBOL formats.

F.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.

OS/3 COBOL

A maximum of 60 characters may be accepted from the system console.

When the FROM option is not used, a maximum of 4095 characters (52 card images) is 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 OS/3 job control stream file.

The compiler creates an internal special-name definition to equate CONSOLE to SYSCONSOLE.

- DISPLAY statement

COBOL-D

When 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, e.g., -32 displayed as 3K.

OS/3 COBOL

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, e.g., -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'.

OS/3 COBOL

A class test may be performed on an item whose usage is either DISPLAY or COMP-3, 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

No automatic support. The item to be tested should be defined as signed.

- INCLUDE Statement/COPY Function

COBOL-D

An INCLUDE statement in the procedure division implies a COPY function.

OS/3 COBOL

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 OS/3 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'.

OS/3 COBOL

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.

  OS/3 COBOL
  This debugging statement is not supported.

  C-mode
  No automatic support. This clause is not supported.

- READ statement

  See F.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.

  OS/3 COBOL
  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.

  OS/3 COBOL
  The word 'PROCEDURE' is required.

  C-mode
  The word 'PROCEDURE' is optional.

- USE FOR LABEL PROCEDURE

  COBOL-D

  USE FOR { **CHECKING** / **CREATING** } { **BEGINNING** / **ENDING** } **LABELS**
  ON { **INPUT** / **OUTPUT** } **file-name**

  OS/3 COBOL

  USE { **AFTER** / **BEFORE** } **STANDARD** { **BEGINNING** / **ENDING** } { **FILE** / **REEL** / **UNIT** } **LABEL**
  PROCEDURE ON { **file-name** / **INPUT** / **I-O** / **OUTPUT** }

C-mode

No automatic support. The USE statement for label procedures must be rewritten in accordance with OS/3 COBOL format.

- WRITE statement

See F.4 for printer considerations, and F.5 for disc considerations.

- *DEBUG card

COBOL-D

*DEBUG packets precede the source deck.

OS/3 COBOL

*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.

F.3.5. Reserved Words

C-mode

The following OS/3 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 OS/3 extended 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
DISC-8415
DISC-8416
DISC-8418
DISC-8430
DISC-8433
DOWN

EQUALS
EXTENDED
EXTENDED-INSERTION
EBCDIC

FILE-LIMIT
FILE-LIMITS
FILE-PREPARATION

INDICES
INDEX
INSERT

JUST

LINE

MAP
MASTER-INDEX
MEMORY
MODULE
MORE-LABELS
MULTIPLE

OFF
OPTIONAL
OUK-90-250
OUK-90-300
OUK-90-400
OUK-90-600
OUK-90-700

PERCENT
PIC
POSITION
PRINTER
PROGRAM

RELEASE
REMAINDER
RENAMES

SEARCH
SEGMENT-LIMIT
SEPARATE
SEEK
SET
SIGN
SORT
SPECIAL-NAMES
STATUS
SYNC
SYNCHRONIZED
SYSCHAN-1 ←
SYSCHAN-2
SYSCHAN-3 ←
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
SYSERR-16
SYSERR-17
SYSERR-18
SYSERR-19
SYSERR-20
SYSERR-21
SYSERR-22
SYSERR-23
SYSERR-24
SYSERR-25
SYSERR-26
SYSERR-27
SYSERR-28
SYSERR-29
SYSERR-30
SYSERR-31
SYSIN
SYSIN-96
SYSIN-128
SYSLOG
SYSSWCH
SYSSWCH-C
SYSSWCH-1
SYSSWCH-2
SYSSWCH-3
SYSSWCH-4
SYSSWCH-5
SYSSWCH-6
SYSSWCH-7
SYSTIME

TAPE
TAPES
TAPE-6
THROUGH
TIME
TRACKS
TRAILING
UNIVAC-9000
UNIVAC-9025
UNIVAC-9030
UNIVAC-9040
UNIVAC-9060
UNIVAC-9070
UNIVAC-9200II
UNIVAC-9300
UNIVAC-9300II
UNIVAC-9400
UNIVAC-9480
UNIVAC-9700
UP

VALUES
VERIFY

WORDS
WHEN

F.4. PRINTER FILE SUPPORT

Support is available for printer files in the conversion mode of the compiler; the aim is to be as compatible as possible with COBOL-D printer file processing within the limits of hardware differences.

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 printer subsystem. All printer files must be defined and referenced according to COBOL-D rules. COBOL-D control characters must be used; consequently, neither a BEFORE ADVANCING nor an ADVANCING mnemonic-name is 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**

Rules:

1. The printer file must have fixed recording mode.

2. Each logical record defined in the printer file must have the first character position reserved for a control character. The control character is used to control printer spacing, but is not actually printed. The legal control characters are as follows:

blank	Print and space 1 line.
0	Print and space 2 lines.
–	Print and space 3 lines.
+	Print and space 0 lines.
1 through 9	Print and skip to channel.
A through C	Print and skip to channel.

3. When the FROM phrase is used, the identifier specified in the FROM phrase must also reserve the first character position to contain a control character.

4. When the AFTER phrase is used, the identifier specified in the AFTER phrase must be a 1-character alphanumeric item that contains a control character.

5. When a literal is specified in the AFTER phrase, the literal must be numeric, with only the following being valid:

0	Print and skip to home paper.
1	Print and space 1 line.
2	Print and space 2 lines.
3	Print and space 3 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 specfied, one line is printed after the overflow punch in the carriage control loop is detected. Because of the manner in which the logical write commands are converted into physical commands, three lines are printed after overflow is detected.

To overcome the problem of three lines being printed, 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 and manual conversion is required.

No action is taken when form overflow is detected unless specified by a USE FOR FORM-OVERFLOW procedure.

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 is to leave 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 SPERRY UNIVAC printer subsystems support various carriage control channels, depending on the printer subsystem on line. The COBOL-D carriage control references are translated as follows:

COBOL-D Control Character	Carriage Control Punch		
	0773	0770	0768
1 (Home paper)	7	7	14, 15
2	2	2	12
3	3	3	13
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	15
8	2	8	8
9	1	9	9
A	3	10	10
B	4	11	11
C (Form overflow)	1	12	9

F.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.

F.5.1. Sequential Organization

- SELECT/ASSIGN clause

  The SELECT/ASSIGN clause requires a source program change to meet the format requirements of OS/3 COBOL.

- APPLY VERIFY clause (not available 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 INVLAID KEY phrase specified.

F.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 OS/3 COBOL.

- 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)

  In OS/3, this clause serves for documentation only.

  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 in 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)

  IN OS/3, this clause serves for documentation only.

- RECORD KEY description

  In C-mode, the record key size must not be less than 3 or greater than 249 bytes.

- SYMBOLIC KEY description

  In C-mode, the symbolic key size must not be less than 3 or greater than 249 bytes.

- OPEN verb

  In C-mode, the file is positioned to the logical record specified in the SYMBOLIC KEY item, or, if none is specified, the file is positioned to the first record.

F.5.3. Direct Organization

No conversion mode support is provided for ORGANIZATION IS DIRECT.

F.5.4. Error Testing in USE AFTER ERROR Procedures

When testing in USE AFTER ERROR procedures, the programmer should replace any calls on DTF interrogation subprograms by tests of SYSERRs (4.2.3) to determine error status. SYSERRs are described in Section 11.

Appendix G. Job Control Stream Requirements

G.1. INTRODUCTION

Any COBOL program you write must be compiled before it can be run. A language translator converts the instructions in your program into a form (an object module) understandable to the computer. The facilities of OS/3 job control are used to relay information to the operating system regarding the requirements for compiling your program. There are two ways to do this:

- Code and keypunch all the job control statements needed to execute the COBOL compiler. See the OS/3 job control user guide, UP-8065 (current version) for details on coding these statements.
- Use a single job control procedure call statement (jproc call) provided by Sperry Univac.

A jproc call generates all the job control statements needed to execute the COBOL compiler. When you specify the proper options for the keyword parameters, you tailor the generated control stream to meet the individual needs of your job. The jproc calls give you the ability to compile your source program (COBOL); compile and link-edit the generated object module to create a load module (COBOLL); or compile, link-edit, and immediately execute this load module (COBOLLG).

G.2. PROCEDURE CALL STATEMENT (COBOL)

Function:

This procedure call statement generates the necessary job control statements to run the COBOL language processor. Optionally, it can generate the job control statements that specify the following:

- Input-source library
- Output-object library
- Copy library
- PARAM control statements to define the format of the compiler listing

Format:

```
// [symbol] { COBOL   } [ PRNTR= { N( lun )          } ] [ ,IN= { (vol-ser-no,label) } ]
            { COBOLL  }           { (  20  [,vol-ser-no]) }     { (RES)            }
            { COBOLLG }           { N                   }     { (RES,label)      }
                                                              { (RUN,label)      }
                                                              { (*,label)        }

[ ,OBJ= { (vol-ser-no,label) } ]
        { (RES,label)        }
        { (RUN,label)        }
        { (*,label)          }
        { (RUN,$Y$RUN)       }

[ ,LIN= { (vol-ser-no,label) } ]
        { (RES,label)        }
        { (RUN,label)        }
        { (*,label)          }
        { (RES,$Y$SRC)       }

[,OUT= (p-1,...,p-n)] [,LST=(p-1,...,p-n)]

[ ,SCR1= { vol-ser-no } ] [ ,SCR2= { vol-ser-no } ]
         { RES        }           { RES        }

[ ,SCR3= { vol-ser-no } ]
         { RUN        }

[ ,ALTLOD= { (vol-ser-no,label) } ]
           { (RES,label)        }
           { (RUN,label)        }
           { (*,label)          }
           { (RES,$Y$RUN)       }
```

Label:

symbol
Specifies the 1- to 6-character source module name; only needed when the IN parameter is used.

Operation:

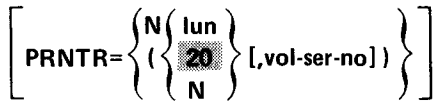
COBOL
This form of the procedure call statement is used to compile a COBOL source program.

COBOLL
This form of the procedure call statement is used to compile a COBOL source program and link-edit the object modules.

COBOLLG
This form of the procedure call statement is used to compile a COBOL source program, link-edit the object modules, and execute the load module.*

**The COBOLLG procedure call statement cannot be used when operating with the shared code data management feature. Instead, use the COBOLL procedure call statement and provide a separate EXEC statement to execute the load module.*

PRNTR Keyword Parameter:



Specifies the logical unit number of the printer. N specifies that the device assignment set for the printer is to be manually inserted in the control stream.

IN Keyword Parameter:

This parameter specifies the input file definition and generates a PARAM IN control statement. The options are:

IN=(vol-ser-no,label)
Specifies the file identifier (label) and the volume serial number (vol-ser-no) where the source input is located.

IN=(RES)
Specifies that the source input is located on the SYSRES device in YSRC.

IN=(RES,label)
This is used if the source input is located on the SYSRES device, but the file identifier (label) is of user-own specification, not YSRC.

IN=(RUN,label)
Specifies that the source input is located on the volume containing the job YRUN file, with the file identifier (label) of user-own specification.

IN=(*,label)
Specifies that the source input is located on a catalog file identified by the file identifier (label).

If omitted, the source input is in the form of embedded data cards (/$, source deck, /*).

Note that when this parameter is specified, it is assumed that the first embedded data set following the PARAM statements contains changes for the source program (G.4), the second contains input to the linkage editor (COBOLL or COBOLLG jproc only), and the third is control stream input to the COBOL source program (COBOLLG jproc only). It may be necessary to insert dummy data sets (/$ followed immediately by /*) into the job stream to ensure that the embedded data sets remain in the sequence just described. For example, if only the third data set is needed, two dummy data sets must be inserted between the last PARAM statement and the third data set. For more information, see example 3c in this section.

OBJ Keyword Parameter:

This parameter specifies the output file definition and generates a PARAM OBJ control statement. The options are:

OBJ=(vol-ser-no,label)
Specifies the file identifier (label) and the volume serial number (vol-ser-no) where the object module is located.

OBJ=(RES,label)
Specifies that the object module is located on the SYSRES device, with the file identifier specified by the label parameter.

OBJ=(RUN,label)

Specifies that the object module is located on the volume containing the job YRUN file, with a file identifier (label) of user-own specification.

OBJ=(*,label)

Specifies that the object module is located on a catalog file identified by the file identifier (label).

If omitted, the object module is located on the job YRUN file.

NOTE:

The OBJ keyword parameter must not be used with COBOLL or COBOLLG.

LIN Keyword Parameter:

LIN=(vol-ser-no,label)

Defines the volume serial number (vol-ser-no) and the file identifier (label) where the copy modules are located. The LFD name is COPY$.

LIN=(RES,label)

Specifies that the copy modules are located on the job's SYSRES device, in the file identified by the file identifier (label).

LIN=(RUN,label)

Specifies that the copy modules are located on the job's YRUN file with the file identifier (label) specified by the user.

LIN=(*,label)

Specifies that the copy modules are located on a catalog file identified by the file identifier (label).

If omitted, the copy modules are located on the SYSRES device in YSRC.

OUT Keyword Parameter:

OUT=(p-1,...,p-n)

Specifies the parameter definitions for the COBOL compiler. This parameter generates a PARAM OUT control statement. See 7.1.2.

LST Keyword Parameter:

LST=(p-1,...,p-n)

Specifies the format of the compiler listing. Generates a PARAM LST control statement. See 7.1.1.

SCR1 Keyword Parameter:

SCR1= { **vol-ser-no** / **RES** }

Specifies the volume serial number of the work file with an identifier of $SCR1.

SCR2 Keyword Parameter:

SCR2= { **vol-ser-no** / **RES** }

Specifies the volume serial number of the work file with an identifier of $SCR2.

SCR3 Keyword Parameter:

SCR3= { **vol-ser-no** / **RUN** }

Specifies the volume serial number of the work file with an identifier of $SCR3.

ALTLOD Keyword Parameter:

ALTLOD=(vol-ser-no,label)

Specifies the location of the compiler to be used, if other than YLOD.

ALTLOD=(RES,label)

Specifies that the alternate load library is located on the job's SYSRES device, in the file identified by the file identifier (label).

ALTLOD=(RUN,label)

Specifies that the alternate load library is located on the job's YRUN file with the file identifier (label) specified by the user.

ALTLOD=(*,label)

Specifies that the alternate load library is located on a catalog file identified by the file identifier (label).

If omitted, the compiler is loaded from YRUN.

Example 1a:

The following illustrates the use of the COBOL procedure call statement in its basic form:

	LABEL	△OPERATION△	OPERAND	△
	1	10	16	
1	//JOB COBOL1A			
2	//COBOL			
3	/$			
4	.			
5	source	deck		
6	.			
7	/*			

Line	Explanation
1	Indicates that the name of the job is COBOL1A
2	Indicates the name of the procedure being called (COBOL). There are no keyword parameters specifying special options for this compilation.
3	Indicates start of data
4–6	Represents the source deck to be compiled
7	Indicates end of data

As coded, this example can be the first step in a job to be followed by the link-edit jproc call, or it can be an entire job in itself by specifying a /& (end-of-job) statement and a // FIN (terminate card reader operations) statement on lines 8 and 9, respectively. The latter case could be used to test-compile a new program or an updated version of an existing program.

Example 1b:

The basic form given in example 1a generates the following control stream:

	LABEL	OPERATION	OPERAND
	1	10	16
1	// JOB COBOL1B		
2	// DVC 20	// LFD PRNTR	
3	// DVC RES		
4	// EXIT ST,C,3,CYL,1		
5	// LBL $SCR1	// LFD $SCR1	
6	// DVC RES		
7	// EXIT ST,C,3,CYL,1		
8	// LBL $SCR2	// LFD $SCR2	
9	// DVC RUN		
10	// EXIT ST,C,3,CYL,1		
11	// LBL $SCR3	// LFD $SCR3	
12	// EXEC COBOL		
13	/$		
14	.		
15	source deck		
16	.		
17	/*		

Line	Explanation
1	Indicates that the name of the job is COBOL1B
2	Indicates the default logical unit number and LFD name of the printer
3–5	Indicates that the first work file needed for compiling is, by default, on the SYSRES device, has both a file identifier and LFD name of $SCR1, and uses the sequential access technique; that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation.
6–8	Identifies the second work file needed for compiling. The only difference between this work file and the first work file is that the file identifier and LFD name are $SCR2 rather than $SCR1.
9–11	Indicates that the third work file needed for compiling is, by default, on the device containing the job's YRUN file. Both the file identifier and the LFD name are $SCR3, and the file extent specification is the same as the first and second work files.
12	Loads the COBOL compiler from YLOD
13	Indicates start of data
14–16	Represents the source deck to be compiled
17	Indicates end of data

As with example 1a, this example can be the first step in a job, or it can be the entire job in itself by specifying the /& statement and the // FIN statement on lines 18 and 19, respectively.

Example 2a:

The following example illustrates the use of a COBOL procedure call statement that defines many of the keyword parameters:

```
      LABEL      ΔOPERATIONΔ          OPERAND                    Δ
      1          10         16                                    72
1     // JOB COBOL2A
2     //PROGNM COBOL PRNTR=21,IN=(RES,U$$RC),                     X
3     //1          OBJ=(DSC2,U$OBJ),                            X
4     //2          LIN=(DSC1,COPYLIB1),                         X
5     //3          SCR2=DSC4,SCR3=DSC1,LST=(S,0)
6     /&
7     // FIN
```

Line	Explanation
1	Indicates that the name of the job is COBOL2A
2	Indicates the name of the procedure being called (COBOL). The source module name is PROGNM. The logical unit number of the printer is 21, and the input file is on the SYSRES device, with a file identifier of U$SRC.
3	Indicates that the output file volume serial number is DSC2, with a file identifier of U$OBJ
4	Indicates that the copy module volume serial number is DSC1, with a file identifier of COPYLIB1
5	Indicates that the second work file needed for compiling is on the device with a volume serial number of DSC4, and the third work file is on the device with a volume serial number of DSC1. By default, the device for the first work file is the SYSRES device. The format of the compiler listing is supplied by the LST parameter.
6	End of job
7	Terminates card reader operations

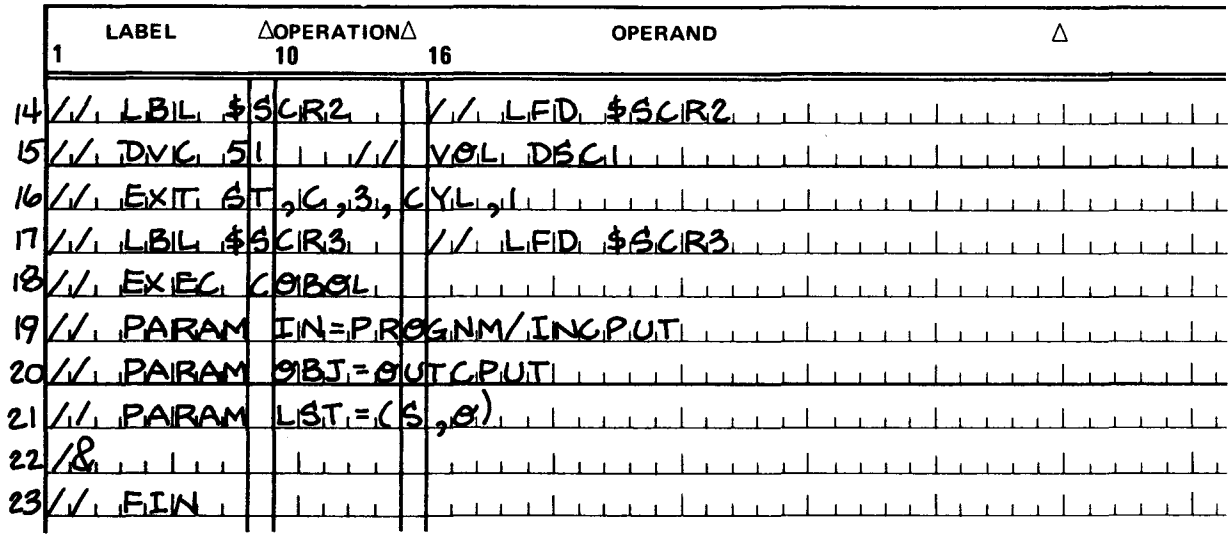
As written, this example is a one-step job that compiles your source program. It produces a nonexecutable object module. Before your program could be executed, a job step would have to be inserted in the control stream that would link-edit the object module to produce an executable load module.

Example 2b:

Based on the keyword parameters specified in example 2a, the following control stream is generated:

	LABEL (1)	△OPERATION△ (10)	OPERAND (16)	△
1	// JOB COBOL2B			
2	// DVC 21	//	LFD PRNTR	
3	// DVC RES			
4	// LBL U$SRC		// LFD INCPUT	
5	// DVC 50	//	VOL DSC2	
6	// LBL U$OBJ		// LFD OUTCPUT	
7	// DVC 51	//	VOL DSC1	
8	// LBL COPYLIB1	//	LFD COPY$	
9	// DVC RES			
10	// EXT ST,C,3,CYL,1			
11	// LBL $$CR1		// LFD $$CR1	
12	// DVC 52	//	VOL DSC4	
13	// EXT ST,C,3,CYL,1			

(continued)



```
      LABEL      OPERATION      OPERAND
      1          10             16
14    //  LBL  $SCR2          //  LFD  $SCR2
15    //  DVC  51             //  VOL  DSC1
16    //  EXIT  ST,C,3,CYL,1
17    //  LBL  $SCR3          //  LFD  $SCR3
18    //  EXEC  COBOL
19    //  PARAM  IN=PROGNM/INCPUT
20    //  PARAM  OBJ=OUTCPUT
21    //  PARAM  LST=(S,0)
22    /&
23    //  FIN
```

Line	Explanation
1	Indicates that the name of the job is COBOL2B
2	Indicates that the printer is to be assigned to the logical unit number 21, with an LFD name of PRNTR. This was obtained from line 2 in example 2a.
3	Indicates that the input file is on the device containing the SYSRES volume. This was obtained from the IN parameter on line 2 in example 2a.
4	Indicates that the input file has a file identifier of U$SRC, with an LFD name of INCPUT. This was obtained from the IN parameter on line 2 in example 2a.
5	Indicates that the output file volume serial number is DSC2. This was obtained from the OBJ parameter on line 3 in example 2a. It is assigned to the device with a logical unit number of 50, which was the first available number in the range of 50–54.
6	Indicates that the output file has a file identifier of U$OBJ, with an LFD name of OUTCPUT. This was obtained from the OBJ parameter on line 3 in example 2a.
7	Indicates that the copy library has a volume serial number of DSC1. It is assigned to the device with a logical unit number of 51, which was the next available number in the range of 50–54. Logical unit number 50 was already assigned to the device with a volume serial number of DSC2 (line 5), so the next available logical unit number is used. This was obtained from the LIN parameter on line 4 in example 2a.
8	Indicates that the copy library is labeled COPYLIB1, with an LFD name of COPY$. This was obtained from the LIN parameter on line 4 in example 2a.

Line	Explanation
9–11	Indicates that the first work file needed for compiling is, by default, on the SYSRES device, has both a file identifier and LFD name of $SCR1, uses the sequential access technique; that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation.
12–14	Indicates that the second work file needed for compiling has a volume serial number of DSC4. This volume serial number has not been previously used in this job, so the next available logical unit number (52) is assigned to this device. This work file has both a file identifier and LFD name of $SCR2, and has the same file extent specification as the first work file. This was obtained from the SCR2 parameter on line 5 in example 2a.
15–17	Indicates that the third work file needed for compiling has a volume serial number of DSC1. Since this volume serial number was already used, this work file uses the same device logical unit number of 51. This work file has both a file identifier and LFD name of $SCR3, and has the same file extent specification as the first and second work files. This was obtained from the SCR3 parameter on line 5 in example 2a.
18	Loads the COBOL compiler from YLOD
19–21	PARAM control statements which identify the processing options for the COBOL compiler. These are generated in the following manner: Line 19 – The module name PROGNM is generated from the label field in line 2 of example 2a. The filename INCPUT is generated automatically when the IN parameter is specified. Line 20 – The filename OUTCPUT is generated automatically when the OBJ parameter is used. Line 21 – The S and O COBOL list options are generated by the LST parameter on line 5 in example 2a.
22	End of job
23	Terminates card reader operations

Example 3a:

The following example illustrates the use of the COBOLL procedure call statement:

```
      LABEL     OPERATION     OPERAND
      1         10            16
1     // JOB MASTER
2     // DVC SO
3     // VOL DSC1
4     // LBL U$LOD
5     // LFD LNKLIB
6     // COBOLL LST=(S,L,C)
7     /$
      .
8     .  COBOL source program
      .
9     /*
10    /$
11       LINKOP OUT=LNKLIB
12       LOADM ABC123
13    /*
14    /&
15    // FIN
```

Line	Explanation
1	Indicates that the name of the job is MASTER
2–5	Defines a file U$LOD on volume DSC1 to be used to hold the linked object module
6	Indicates the name of the procedure being called (COBOLL) and indicates the compiler options for this compilation
7	Indicates start of data
8	Indicates the COBOL source program
9	Indicates end of data
10	Indicates start of data
11	Indicates that the linkage editor is to write the load module to the file with the lfdname LNKLIB
12	Indicates that the name of the load module is ABC123

Line	Explanation
13	Indicates end of data
14	Indicates end of job
15	Terminates card reader operations

Example 3b:

Based on the keyword parameters specified explicitly and implicitly in example 3a, the following control stream is generated:

```
   LABEL    ΔOPERATIONΔ      OPERAND                                  Δ          COMMENTS
   1            10       16
1  // JOB MASTER
2  // DVC 50 // VOL DSC1 // LBL U$LOD // LFD LNKLIB
3  // DVC 20 // LFD PRNTR
4  // DVC RES // EXT ST,C,3,CYL,1 // LBL $SCR1 // LFD $SCR1
5  // DVC RES // EXT ST,C,3,CYL,1 // LBL $SCR2 // LFD $SCR2
6  // DVC RES // EXT ST,C,3,CYL,1 // LBL $SCR3 // LFD $SCR3
7  // EXEC COBOL
8  // PARAM LIST=(S,L,C)
9  /$
   .
10 . COBOL source program
   .
11 /*
12 // DVC RES // EXT ST,C,3,CYL,1 // LBL $SCR1 // LFD $SCR1
13 // EXEC LNKEDT
14 /$
15     LINKOP OUT=LNKLIB
16     LOADM ABC123
17 /*
18 /&
19 // FIN
```

Line	Explanation
1	Indicates that the name of the job is MASTER
2	Defines a file U$LOD on volume DSC1 to be used to hold the linked object module
3	Indicates that the printer is to be assigned to logical unit number 20 with an lfdname of PRNTR
4–6	Defines the three work files necessary for compiler execution
7	Loads and executes the COBOL compiler

Line	Explanation
8	Indicates parameter options
9	Indicates start of data
10	Indicates the COBOL source program
11	Indicates end of data
12	Defines the work file necessary for LNKEDT execution
13	Loads and executes the linkage editor
14	Indicates start of data
15	Indicates that the linkage editor is to write the load module to the file with the lfdname LNKLIB
16	Indicates that the name of the load module is ABC123
17	Indicates end of data
18	Indicates end of job
19	Terminates card reader operations

Example 4a:

The following example illustrates the use of the COBOLLG procedure call statement. The input file and the format of the output listings are defined.

```
      LABEL      △OPERATION△      OPERAND                                  72
      1          10               16
1     //JOB       MASTER
2     //MASTER    COBOLLG          IN=(ABC123,PAYMAST),                     X
3     //          LST=(A,C,01,S)
4     /&
5     //FIN
```

Line	Explanation
1	Indicates that the name of the job is MASTER
2	Indicates that the name of the source module is MASTER and that the name of the procedure being called is COBOLLG. Therefore, this example compiles, link-edits, and executes the source program MASTER. The input file is on the device with a volume serial number of ABC123 and has a file identifier of PAYMAST.

Line	Explanation
3	Indicates the format of the compiler listing
4	End of job
5	Terminates card reader operations

Example 4b:

Based on the keyword parameters specified explicitly and implicitly in example 4a, the following control stream is generated:

	LABEL	△OPERATION△	OPERAND	△
	1	10	16	
1	//JOB	MASTER		
2	//OPTION	LINK,GO		
3	//DVC	20	//LFD PRNTR	
4	//DVC	50	//VOL ABC123	
5	//LBL	PAYMAST	//LFD INCPUT	
6	//DVC	RES		
7	//EXIT	SIT,C,3,CYL,11		
8	//LBL	$SCR1	//LFD $SCR1	
9	//DVC	RES		
10	//EXIT	SIT,C,3,CYL,11		
11	//LBL	$SCR2	//LFD $SCR2	
12	//DVC	RUN		
13	//EXIT	SIT,C,3,CYL,11		
14	//LBL	$SCR3	//LFD $SCR3	
15	//EXEC	COBOL		
16	//PARAM	IN=MASTER/INCPUT		
17	//PARAM	LST=(A,C,0,5)		
18	/&			
19	//FIN			

Line	Explanation
1	Indicates that the name of the job is MASTER
2	Indicates that the source program is to be link-edited and then executed after it has been compiled. This was obtained from COBOLLG specified on line 2 in example 4a.

Line	Explanation
3	Indicates, by default, that the printer is to be assigned to the logical unit number 20, with an LFD name of PRNTR
4	Indicates that the input file is on the device with the logical unit number of 50 and has a volume serial number of ABC123. This was obtained from the IN parameter on line 2 in example 4a.
5	Indicates that the input file has a file identifier of PAYMAST, with an LFD name of INCPUT. This was obtained from the IN parameter on line 2 in example 4a.
6–8	Indicates, by default, that the first work file needed for compiling is on the SYSRES device, has both a file identifier and LFD name of $SCR1, uses sequential access technique; that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation
9–11	Indicates, by default, that the second work file needed for compiling is on the SYSRES device. This work file has both a file identifier and LFD namd of $SCR2. It has the same file extent specification as the first work file.
12–14	Indicates, by default, that the third work file needed for compiling is on the SYSRUN device. This work file has both a file identifier and LFD name of $SCR3. It has the same file extent specification as the first and second work file.

Line	Explanation
15	Loads the COBOL compiler from YLOD
16–17	PARAM control statements that identify the processing options for the COBOL compiler. These are generated as follows: Line 16 – The module name MASTER is generated from the label field on line 2 of example 4a. The filename INCPUT is generated automatically when the IN parameter is specified. Line 17 – The A, C, O, and S COBOL list options are generated by the LST parameter on line 3 in example 4a.
18	End of job
19	Terminates card reader operations

Implicit in the // OPTION LINK,GO statement on line 2 of example 4b is the creation of a load module named LNKLOD by the linkage editor and the execution of that load module. This is performed after the source program has been compiled. Any output is temporarily stored on the SYSRUN device.

Example 4c:

If the job requires that additional modules be included from a library, input to the linkage editor must be included in the job control stream. The following example shows where that input would be inserted in job MASTER as described in example 4a.

	LABEL	△OPERATION△	OPERAND	△	72
	1	10	16		
1	// JOB	MASTER			
2	//MASTER	COBOLLG	IN=(ABC123,PAYMAST),		X
3	//		LST=(A,C,O,S)		
4	/$				
5	/*				
6	/$				
7			.		
8	linkage	editor	input		
9			.		
10	/*				
11	/&				
12	// FIN				

Line	Explanation
1–3	Same as example 4a
4–5	Dummy data set to show there are no source corrections for the compiler. This must be included to keep the linkage editor input in proper sequence. (See page G–3.)
6–10	Data set that is input to linkage editor
11	End of job
12	Terminates card reader operations

The generated control stream would be the same as in example 4b, except for the inclusion of the two data sets between lines 17 and 18.

Example 4d:

If job MASTER as described in example 4a had its source program on cards, and both linkage editor input and control stream input to the program were to be included in the run, the control stream would be set up as follows:

	LABEL	OPERATION	OPERAND
	1	10	16
1	/\/ JOB	MASTER	
2	/\/MASTER	COBOLLG	LST=(A,C,0,S)
3	/$		
4	.		
5	source	deck	
6	.		
7	/*		
8	/$		
9	.		
10	linkage	editor input	
11	.		
12	/*		
13	/$		
14	.		
15	control	stream input	
16	.		
17	/*		
18	/&		
19	/\/ FIN		

Line	Explanation
1	Indicates that the name of the job is MASTER
2	Indicates that the name of the source module is MASTER and that the name of the procedure being called is COBOLLG. The LST parameter indicates the format of the compiler listing.
3–7	Source program
8–12	Linkage editor data set
13–17	Data set that contains control stream input to the source program
18	End of job
19	Terminates card reader operations

The generated control stream would be the same as in example 3b, except that lines 4 and 5 describing the source input file would not be present, line 16 would be eliminated, and the three embedded data sets would be inserted between lines 17 and 18. Note that, if control stream input is to be included and there is no linkage editor input, a dummy data set must be inserted after the source deck to keep the control stream input data set in its proper sequence.

G.3. 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 S or U.
- Switch-2 (X'20') is set to 1 if the compiler issues any diagnostic messages with the severity code C.

These bit settings are logically superimposed onto the UPSI byte; therefore, any of the 8 UPSI bits that were set before the compilation still will be set after the compilation.

G.4. SOURCE CORRECTION FACILITY

When the source program resides on a library file, it is possible to change the source as it is read into the compiler. A /$ and /* data set immediately following the PARAM statements may contain correction cards for the source program. The method of correction is the same as the OS/3 system librarian correct module (COR) function. For details, refer to the OS/3 system service programs (SSP) user guide, UP-8062 (current version). The corrections apply only to the compilation. The original source program on the library is not changed.

G.5. DATA DEFINITION (DD) JOB CONTROL STATEMENT KEYWORD PARAMETERS

The DD job control statement is used to change data management keywords at execution time. Instead of changing the COBOL source code, the user can override data management keyword specifications when the COBOL object program is executing. The DD statement keyword parameters that may be specified for a COBOL program are as follows:

```
LACE=n
SIZE=n
UOS=n
ACCESS=(EXC
       {EXCR}
       {SRDO}
       {SRD }
       {SUPD}
       (SADD)

FILABL=(NO  )
       {NSTD}
       (STD )
TPMARK=NO
VMNT=ONE
```

When the user specifies these keyword parameters, extreme care must be used so that the effect of changing one parameter does not cause a conflict. To avoid conflicts, the user should carefully examine the file usage specified in COBOL source programs and the default parameters set by the compiler-generated data management specifications.

The DD statement applies to basic data management users and consolidated data management users. For keyword parameter information, see the basic data management user guide, UP-8068 (current version) or the consolidated data management macroinstructions user guide, UP-8826 (current version). A complete description of the DD job control statement is explained in the job control user guide, UP-8065 (current version).

Appendix H. Shared Code Interface

H.1. GENERAL

COBOL programs to be executed under control of the SPERRY UNIVAC Series 90 Information Management System (IMS/90) should be compiled by using the shared-code parameter when used with the SPERRY UNIVAC OS/3 Operating System (OS/3).

The format of the PARAM statement is:

```
// PARAM OUT=(M)
```

H.2. ACTION PROGRAM

A COBOL program running under control of IMS/90 is called an action program. A COBOL action program compiled under the shared-code parameter is reentrant at CALL interrupts. The following rules and restrictions of COBOL action programs are checked for, and diagnosed at compile time when OUT=(M) is activated.

Rules:

1. The following COBOL verbs, clauses, and sections are illegal in the shared-code mode. They will be diagnosed and deleted from the program.

ALTER	RETURN	SYSLST
CLOSE	REWRITE	WRITE
DECLARATIVE SECTION	SEEK	
ENTRY	SEGMENT-LIMIT	
EXHIBIT	SORT	
EXIT-PROGRAM	STOP	
FILE SECTION	SYSCHAN-t	
INPUT-OUTPUT SECTION	SYSCONSOLE	
INSERT	SYSERR [-m]	
OPEN	SYSIN	
READ	SYSIN-96	
READY TRACE	SYSIN-128	
RELEASE	SYSLOG	
RESET TRACE		

2. The PROCEDURE DIVISION header must contain a USING clause and can be the only entry point in the program.

3. For a list of the valid IMS/90 function names, refer to the Series 90 information management system/90 programmer reference, UP-8083 (current version).

4. A section priority number ⩾50 will be diagnosed and changed to 0.

5. The SPECIAL-NAMES paragraph may define only the following four implementor names.

   SYSCOM
   SYSDATE
   SYSTIME
   SYSSWCH

   In conjunction with this restriction, the ACCEPT and DISPLAY verbs may be used only to reference the allowable system names.

6. The following verbs must not have working-storage items as receiving operands. Upon detection of this condition, the compiler will generate the statement and issue a precautionary diagnostic.

ADD	PERFORM (varying option)
COMPUTE	SEARCH (varying option)
DIVIDE	SET
EXAMINE (replacing option)	SUBTRACT
MOVE	TRANSFORM
MULTIPLY	

7. All USING arguments of the CALL verb must be datanames of any level (except 88) in the working-storage or linkage sections.

For the COBOL object program to be reentrant at CALL interrupts, the volatile work area used by the program must be saved and restored by the IMS/90 system. The size of the area (which varies between programs) is displayed in decimal by the printer immediately prior to the COBOL COMPILATION COMPLETE message. The message reads:

SHARED CODE VOLATILE DATA AREA = nnnn BYTES

This size is used in computing the SHRDSIZE parameter in the IMS/90 configurator. (Refer to the Series 90 information management system/90 programmer reference, UP-8083 (current version)).

Normally, execution-time errors result in a CE error message and program termination. In an action program, execution-time errors result in a program check interrupt and a snapshot dump of the action program with the address of the CE message in register 1. The action program is terminated.

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD