60497100

CONTROL DATA CORPORATION

COBOL VERSION 5 REFERENCE MANUAL

CDC® OPERATING SYSTEMS

NOS 1

NOS/BE 1

REVISION RECORD

REVISION	DESCRIPTION
A	Original release.
(03-08-76)	
B	Miscellaneous errors have been corrected. No technical changes have been made.
(07-20-76)	
C	This revision reflects COBOL 5.1 (feature CP176) at PSR level 439.
(12-06-76)	
D	This revision includes the full CALL/CANCEL facility (feature CP176) at PSR level 446.
(03-25-77)	
E	This revision reflects COBOL 5.2 (feature CP186) at PSR level 472. Changes include an interface
(04-15-78)	to Basic Access Methods 1.5, Advanced Access Methods 2, and CYBER Database Control System 2.
F	This revision reflects COBOL 5.3 (feature 1250). Changes include an interface to Advanced Access
(7-20-79)	Methods 2.1, and CYBER Database Control System 2.1.
G	This revision includes the COBOL Communication Facility (CCF) for the Message Control System 1.0
(12-07-79)	plus miscellaneous technical corrections.
H	Released at PSR level 528. This revision includes the ANSI=AUDIT parameter option, an interface to
(10-31-80)	the Common Memory Manager (CMM), and miscellaneous technical corrections.
Publication No. 60497100	

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

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

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

Page	Revision
Front Cover	-
Title Page	-
ii	H
iii/iv	H
v/vi	A
vii	H
viii	H
ix thru xiv	H
xv	F
1-1 thru 1-5	G
1-6 thru 1-12	F
1-13	H
1-14	F
1-15	F
1-16 thru 1-18	G
1-19	H
1-20	G
2-1	G
3-1 thru 3-9	F
3-10	H
3-11	F
3-12	F
3-13	G
3-14 thru 3-19	H
4-1	G
4-2	G
4-3 thru 4-5	H
4-6	G
4-7	H
4-8	H
4-8.1/4-8.2	H
4-9	H
4-10 thru 4-13	F
4-14	H
4-15	G
4-16 thru 4-20	F
4-21	H
4-22	G
4-23	F
4-24	G
4-25	H
4-26 thru 4-31	G
5-1	F
5-2	G
5-3	G
5-4 thru 5-14	H
5-15	G
5-16 thru 5-18	H
5-18.1	H
5-18.2	H
5-19	F
5-20	H
5-21	F

Page	Revision
5-22	F
5-23 thru 5-29	G
5-30 thru 5-34	H
5-35 thru 5-37	G
5-38 thru 5-40	H
5-41	G
6-1 thru 6-11	F
6-12	H
7-1	H
7-2	F
7-3	H
7-4	F
7-5	H
7-6	H
8-1 thru 8-3	F
9-1 thru 9-4	F
10-1	F
10-2	H
10-3	F
10-4	F
10-5	H
10-6	G
10-7	F
11-1 thru 11-5	H
12-1 thru 12-11	H
13-1 thru 13-4	F
14-1	G
14-2	G
14-3 thru 14-5	F
14-6 thru 14-8	H
15-1	F
15-2	G
15-3	G
15-4 thru 15-7	H
A-1	H
A-2	H
A-3 thru A-8	F
B-1 thru B-8	H
C-1 thru C-10	H
D-1	F
D-2	F
E-1	H
E-2	H
E-3	G
E-4	H
E-5	H
E-6	G
E-7 thru E-31	H
F-1	F
F-2	F
Index-1 thru -15	H
Comment Sheet	H
Mailer	-
Back Cover	-

ACKNOWLEDGEMENT

The following acknowledgement is reproduced in its entirety at the request of the American National Standards Institute.

"Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document 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 the following acknowledgment paragraphs in their entirety as part of the preface to any such publication (any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment):

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 CODASYL Programming Language 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.

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, 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."

PREFACE

This manual describes the COBOL Version 5.3 language which operates under control of the following operating systems:

NOS 1 for the CONTROL DATA® CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

NOS/BE 1 for the CDC® CYBER 170 Series, CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

COBOL 5 is designed to be a superset of the language specified in American National Standard X 3.23- 1974, COBOL. Extensions to the standard language are indicated in this manual by shading.

The manual is written for a programmer familiar with a COBOL language and the operating system under which the COBOL 5 compiler is operating.

The users of COBOL can find additional pertinent information in the Control Data Corporation publications. The NOS manual abstracts and the NOS/BE manual abstracts are instant-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals can be useful in determining which manuals are of greatest interest to a particular user. The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

The publications are listed below in alphabetic order in groupings that indicate relative importance to readers of this manual.

The following publications are of primary interest:

Publication	Publication Number
COBOL Version 5 Diagnostic Handbook	60482500
COBOL Version 5 Instant Manual	60497300
COBOL Version 5 User's Guide	60497200
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS/BE Version 1 Reference Manual	60493800

The following publications are of secondary interest:

Publication	Publication Number
COBOL Version 4 to COBOL Version 5 Conversion Aid Reference Manual	19265021
COBOL Version 5 Report Writer User's Guide	60496900
Common Memory Manager Version 1 Reference Manual	60499200
CYBER Database Control System Version 1 Reference Manual	60498700
CYBER Database Control System Version 2 Reference Manual	60481800
CYBER Loader Version 1 Reference Manual	60429800
CYBER Record Manager Advanced Access Methods Version 2 Multiple-Index Processor User's Guide	60480900
CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300

CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700
DDL Version 2 Reference Manual, Volume 1: Schema Definition	60498400
DDL Version 2 Reference Manual, Volume 2: COBOL Sub-Schema Definition	60498500
DMS-170 DDL Version 3 Reference Manual, Volume 1: Schema Definition for Use With: COBOL FORTRAN Query Update	60481900
DMS-170 DDL Version 3 Reference Manual, Volume 2: Sub-Schema Definition for CYBER Database Control System Use With: COBOL Query Update	60482000
Message Control System Version 1 Reference Manual	60480300
Network Products Transaction Facility Version 1 Reference Manual	60455340
NOS Version 1 Manual Abstracts	84000420
NOS/BE Version 1 Manual Abstracts	84000470
Software Publications Release History	60481000
Update Version 1 Reference Manual	60449900

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

NOTATIONS USED IN THIS MANUAL

NOTATION USED IN FORMATS

UPPERCASE	Words are COBOL reserved words. They must be spelled correctly including any hyphens; they cannot be used in a source program except as specified.
UNDERLINED	Words are required when the format in which they appear is used.
Lowercase	Generic terms which represent the words or symbols supplied by the programmer. When generic terms are repeated in a format, a number or letter is appended to the term for identification in the subsequent discussion.
Brackets [ ]	Optional portion of a format. All of the format within the brackets can be omitted or included at programmer option. If items are stacked vertically within brackets, only one of the stacked items can be used.
Braces { }	Portion of a format in which only one of the vertically stacked items can be used. If one of the options contains only reserved words that are not keywords, that option is the default option. Braces are also used to enclose the portion of a required entry that can be repeated.
Ellipses . . .	Repetition indicator. Portion of format enclosed in the immediately preceding braces or brackets can be repeated at programmer option.

Punctuation symbols shown within the formats are required unless enclosed in brackets or specifically noted as optional. In general, commas and semicolons are optional; periods are required to terminate paragraphs, sentences, and Data Division entries. At least one space must follow all punctuation symbols.

NOTATION USED IN EXAMPLES

↑ indicates the position of an assumed decimal point in an item.

A plus or minus sign above a numeric character indicates an operational sign is stored in combination with the numeric character.

Character positions in storage are shown by boxes. $\boxed{A}\boxed{B}\boxed{C}\boxed{D}$ An empty box means an unpredictable result.

Δ indicates a space (blank).

SHADING USED IN MANUAL

Control Data extensions to the language described in American National Standard X3.23-1974, COBOL, are indicated by shading. Shading is also used to indicate processing that is different from that specified in the standard. Language and processing that are implementor-defined but within the standard are not shaded. Further, references to Control Data features, such as the direct file organization, are not shaded unless that feature is the topic under discussion.

COBOL SOURCE PROGRAM

A COBOL program is composed of a series of lines that conform to the structure and syntax of the COBOL language. The source program is processed by the COBOL 5 compiler and changed into a set of tables that can be loaded by the operating system loader and that can be executed by hardware instructions.

When the compiler executes in response to a control statement entered through a batch job or interactive terminal, it performs the following functions:

Read the file containing the source program. INPUT is assumed to be the file name in the absence of a parameter specifying another name.

Check the source program for errors in program structure and language syntax. Write messages that note any errors to a file to be printed on the line printer or displayed at a terminal.

Depending on parameters specified on the compiler call, write a copy of the source program to a print or display file. Write a data map and a data cross reference map to this file to summarize data item descriptions and references within the program. OUTPUT is assumed to be the output file name in the absence of a parameter specifying another name.

Compile the source lines into executable code, unless binary code generation is suppressed by a compiler call parameter. Write the executable instructions to a file in a format suitable for loading and executing. LGO is assumed to be the name of the file to hold the compiled program in the absence of a parameter specifying another name.

Once the source program is compiled, it can be loaded and executed by a control statement that names the file of executable instructions. Section 12, Compiler Call and Execution, describes other functions that can be performed during compilation and execution.

During execution, the compiled program makes use of execution-time routines that are part of the system library. Files referenced in the program are opened, read, written, rewritten, and closed through the CDC CYBER Record Manager facility common to several products running under the NOS and NOS/BE operating systems. Depending on the source program statements, Sort/Merge routines, Report Writer routines, and CDC CYBER Database Control System routines might also be used during execution.

The job in which the program executes is responsible for making data files available before the program begins execution and for properly disposing of output files after execution ends. If the compilation involves information on a library, the job is also responsible for making the library file available before compilation.

CDC offers guidelines for the use of the software described in this manual. These guidelines appear in appendix F. Before using the software described in this manual, the reader is strongly urged to review the content of this appendix. The guidelines recommend use of this software in a manner that reduces the effort required to migrate application programs to future hardware or software systems.

SOURCE PROGRAM FORMAT

A source program consists of a series of lines with a particular syntax and structure.

SOURCE LINES

Each line in the source program can be represented as an 80-column punch card or as a card image within the range of one through 100 characters. The maximum number of lines that can be specified in a program is 32 767. Columns define areas of varying significance within a line:

Columns 73 through 100

Optional program identification. Any character in the computer character set listed in appendix A is allowed in these columns. Characters in these columns appear on the source listing for the program, but are not otherwise processed by the compiler.

Columns 1 through 6

Sequence number for the source line. The use of the sequence number depends on the PSQ parameter in the compiler call statement. When the PSQ parameter is specified, the following rules apply:

The sequence number can contain only digits and spaces. Spaces are ignored in a sequence field; however, the field cannot be all spaces. Characters other than a digit or a space cause a diagnostic to be issued. If a line with an illegal sequence number follows a line with a legal sequence number, the last legal sequence number is used. If no legal sequence numbers exist, the normal compiler generated lines are assigned (as if PSQ were omitted).

Every line must be numbered.

All diagnostics at execution time and compile time refer to sequence numbers.

The numbers are not required to be in sequence; however, for ease in locating lines referenced in diagnostics, the numbers should be sequenced.

A sequence number cannot exceed 65535 and cannot equal zero or all spaces. If one of these errors occurs, a diagnostic is issued and the sequence number is determined as indicated in the preceding paragraphs.

If the PSQ parameter is not specified, the sequence number is optional and can contain any character in the computer character set. The compiler does not process these characters.

Columns 7 through 72

Language statements that are processed by the compiler to produce executable code. Only COBOL language characters listed in table 1-1 are allowed in these columns, except when the explanation of a language element specifically allows any character from the computer character set. These columns are divided into three areas:

7	Indicator area
8-11	Area A
12-72	Area B

The indicator area can contain only five characters that identify the type of line in which it occurs:

space	Normal line to be compiled
-	Continuation line explained below
*	Comment line explained below
/	Comment line explained below
D	Debugging line explained in section 10, Debugging Aids

TABLE 1-1. COBOL LANGUAGE CHARACTERS

Character	Meaning
0-9	Digit
A-Z	Letter
	Space (blank)
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma
;	Semicolon
.	Period (decimal point, full stop)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
:	Colon

Area A is used to begin the following language elements:

Division header

Section header

Paragraph header

Paragraph-name

Level 01 and level 77 Data Description entries

Level indicator

Keywords DECLARATIVES and END DECLARATIVES

Area B is the starting area for:

Record-name or item name and record or item description

Statements and clauses

Continuation of statements and clauses

Statements that follow a paragraph-name can start in area A of the same line as the paragraph-name when the paragraph-name is limited to a single character. Language elements that should begin in area B are accepted when they begin in area A.

Figure 1-1 summarizes source program line format.

Continuation Lines

Any sentence, entry, phrase, or clause can be continued in area B of the immediately following line. Under two circumstances the continuation line must contain a hyphen in column 7 to indicate that the line is a continuation of the previous line.

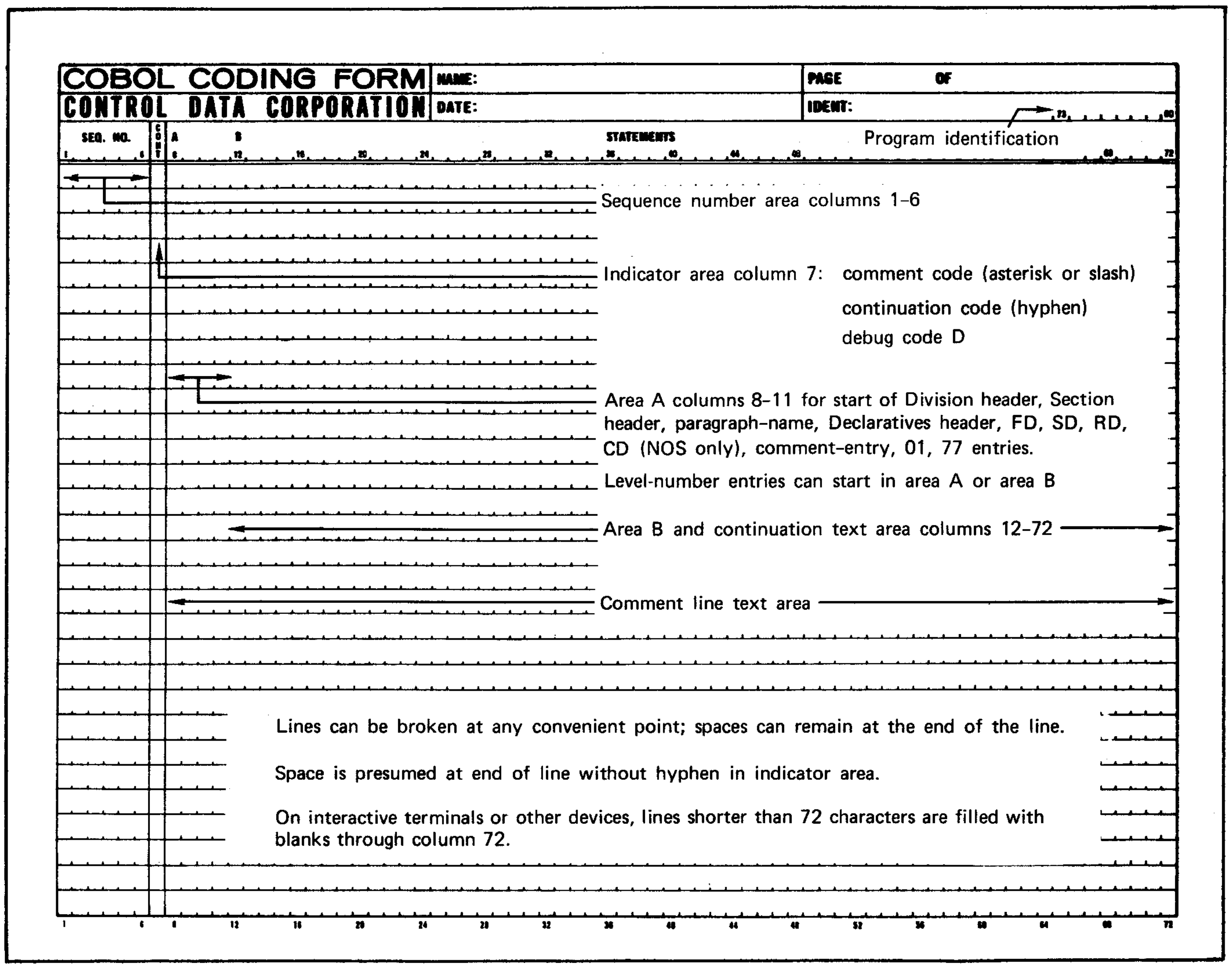
A word or a numeric literal is split between lines. The continuation line must have a hyphen in the indicator area, a blank area A, and text beginning in area B. The first nonblank character in area B is presumed to immediately follow the last nonblank character of the previous line.

A nonnumeric or boolean literal is split between lines. The continuation line must have a hyphen in the indicator area, a blank area A, and the first nonblank character in area B must be a quotation mark. All spaces at the end of the continued line through column 72 are considered part of the literal and are presumed to immediately precede the first character after the quotation mark in the continuation line.

The hyphen should not be used in the indicator area except under these two circumstances. If a hyphen does not appear in the indicator area, the last nonblank character in the preceding line is presumed to be followed by a space.

Comment Lines

A comment line has an asterisk or slash in the indicator area. A comment line can appear anywhere in a source program after the Identification Division header. Any character in the computer character set can appear in area A and area B. Comment lines are listed in the source listing of the program, but are not otherwise processed by the compiler.



COBOL CODING FORM
CONTROL DATA CORPORATION
NAME:
DATE:
PAGE OF
IDENT:
SEQ. NO.
CONT.
A
B
STATEMENTS
Program identification
Sequence number area columns 1-6
Indicator area column 7: comment code (asterisk or slash)
continuation code (hyphen)
debug code D
Area A columns 8-11 for start of Division header, Section header, paragraph-name, Declaratives header, FD, SD, RD, CD (NOS only), comment-entry, 01, 77 entries.
Level-number entries can start in area A or area B
Area B and continuation text area columns 12-72
Comment line text area
Lines can be broken at any convenient point; spaces can remain at the end of the line.
Space is presumed at end of line without hyphen in indicator area.
On interactive terminals or other devices, lines shorter than 72 characters are filled with blanks through column 72.

Figure 1-1. Source Program Line Format

The two characters that indicate a comment line differ as follows:

*	Comment is listed as it is encountered. Must be used when a comment line appears in the source program.
/	Causes a page eject to occur for the source listing file before the comment line is listed. Can be used for the first line of a group of successive comment lines.

The LIST/NOLIST commands, which control the printing of source program and object listings, are specified on comment lines. This special usage of comment lines is described in section 12.

Blank Lines

A blank line is one that is blank in columns 7 through 72. A blank line can appear anywhere in the source program, except immediately preceding a continuation line.

PROGRAM FORMAT

A COBOL program has four divisions. All divisions are required, and they must appear in a specific order. The division names in the required order are: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. The general function of each division is:

Identification Division

Names the program and documents the author and date.

Environment Division

Names each file referenced in the program and equates it to a file known to the operating system or to a console or terminal. Establishes checkpoint conditions, collating sequence, debugging mode, and sub-schema name. Documents the computer used to compile and execute the program. Files contained in a data base described by a sub-schema might be described here. Refer to section 14, Sub-Schema Facility.

Data Division

Describes in detail each file and each data item to be used in the program, including the data items used for the Message Control System (MCS) under NOS. Items contained in a data base described by a sub-schema are not described here.

Procedure Division

Specifies processing to be performed including file input and output, arithmetic operations, sorting, report generation, and procedures to execute only under specific conditions.

Each division contains source statements made up of language elements described below. Within each division, most source statements are optional, depending on the needs of the program. Each division is described in detail in a separate section of this manual.

Figure 1-2 summarizes source program structure, showing the sections or paragraphs within each division and the order in which they must appear.

```
IDENTIFICATION DIVISION.
     program-id paragraph followed by predefined
     paragraphs and comment entries

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
     predefined paragraphs and entries
INPUT-OUTPUT SECTION.
     predefined paragraphs and entries

DATA DIVISION.
FILE SECTION.
     predefined level indicator entries and
     record description entries
COMMON-STORAGE SECTION.
     record description entries
WORKING-STORAGE SECTION.
     record description entries
SECONDARY-STORAGE SECTION.
     record description entries
LINKAGE SECTION.
     record description entries
COMMUNICATION SECTION.
     communication description entries
REPORT SECTION.
     report description entries

PROCEDURE DIVISION.
DECLARATIVES.
     declarative statements, procedures
END DECLARATIVES.
     statements in paragraphs or sections
```

Figure 1-2. Source Program Overview

LANGUAGE STRUCTURE

The COBOL language is composed of the 51 characters listed in table 1-1. Additional characters in the computer character set listed in appendix A can be used in a program when they are a part of a nonnumeric literal, comment-entry, or comment line.

Individual characters of the language are concatenated to form character-strings and separators. Concatenation of character-strings and separators forms the text of a source program.

Figure 1-3 shows the various types of separators and character-strings in a source program. Separators can be concatenated with other separators or a character-string; a character-string must be concatenated with separators. The words listed under the box titled Character-Strings in figure 1-3 present the terminology used throughout this manual. The rules for forming COBOL words are also summarized in this figure.

A discussion of the separators and character-strings of figure 1-3 follows.

SEPARATORS

A separator is a string of one or more of the punctuation characters , ; : . " ( ) == and space. Separators must appear in the source program in specific places. Not all separators are interchangeable.

Punctuation characters can have uses other than as separators. These characters do not function as separators when they appear as part of:

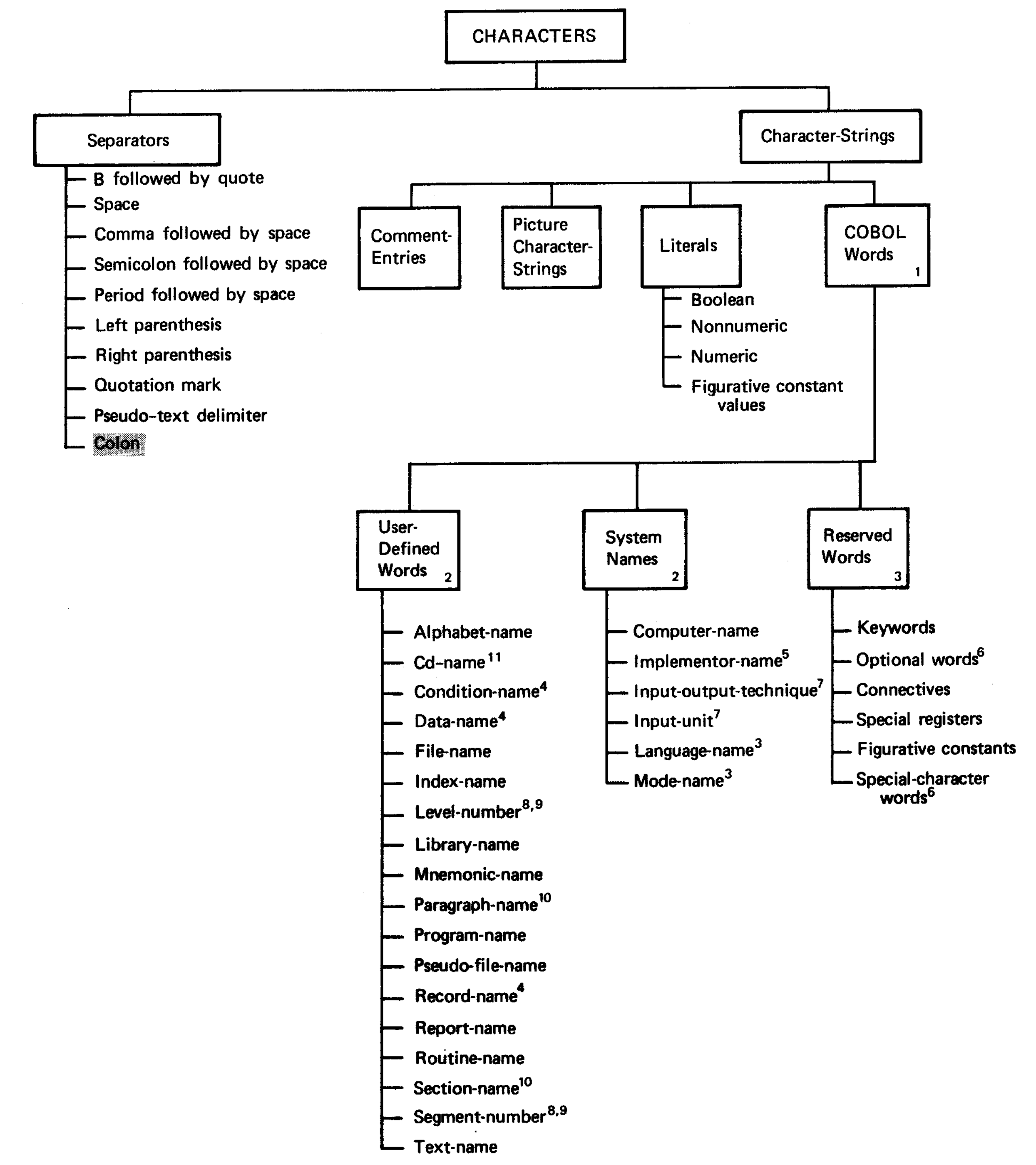
PICTURE clause character-string.

Nonnumeric literal.

Numeric literal.

The punctuation characters have the following functions as separators:

space	Separates language elements. Can be concatenated with other separators except as otherwise noted.
. space	Terminates headers, entries, and sentences as defined by formats.
, space	Separates clauses and statements for readability. Optional, and restricted to the usage indicated in the formats. Cannot immediately precede the first clause of an entry or paragraph. Interchangeable with ; space.
; space	Separates clauses and statements for readability. Optional, and restricted to the usage indicated in the formats. Cannot immediately precede the first clause of an entry or paragraph. Interchangeable with , space.
: colon	Delimits leftmost character position and length of reference modified item.
(and)	Delimits subscripts, indexes, arithmetic expressions, or conditions. Must appear in balanced pairs.
"	Delimits nonnumeric literal. Can appear only in balanced pairs except when the literal is continued on subsequent lines. Opening quotation mark must be immediately preceded by a space or left parenthesis; closing quotation mark must be immediately followed by a separator space, comma, semicolon, period, or right parenthesis. Also used to delimit boolean literals on the right.



[1] 1 through 30 characters composed of A through Z, 0 through 9, or -; however the character - cannot be first or last character, except as noted in 6.

[2] Must contain at least one alphabetic character except as noted in 9 and 10. Must be unique except as noted in 4.

[3] Particular names required.

[4] Words can be duplicated among these groups if qualified for unique reference.

[5] Must begin with a letter.

[6] See text for syntax.

[7] Documentary only.

[8] Need not be unique.

[9] Digits only.

[10] Need not contain alphabetic characters.

[11] Can only be used under NOS.

Figure 1-3. Language Structure

==	Delimits pseudo-text in COPY and REPLACE statements. Can appear only in balanced pairs. Opening delimiter must be immediately preceded by a space; closing delimiter must be immediately followed by a separator space, comma, semicolon, or period.
B"	Delimits boolean literals on the left. This separator must be immediately preceded by a space or left parenthesis. The boolean literal is delimited on the right by a closing quotation mark. The opening and closing quotation marks must appear in balanced pairs except when the literal is continued on subsequent lines.

A character used in a source program as a quotation mark might appear on an output device as a different character, depending on the device and the character set. In particular, it might appear as ≠ on a print file.

When the QUOTE IS APOSTROPHE clause is specified in the SPECIAL-NAMES paragraph of the Environment Division, the character apostrophe assumes the role of the character quotation mark. This interchange of character functions can also be selected by the APO parameter on the compiler call.

COBOL WORDS

A COBOL word is a character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

User-Defined Words

A user-defined word is a COBOL word that must be supplied by the programmer to satisfy the format of a clause or statement. These words are grouped into the sets listed in figure 1-3. Condition-name, data-name, and record-name belong to the same set. All user-defined words, except segment-number and level-number, can belong to one and only one set.

Except for level-number and segment-number, all user-defined words within a given set must be unique or be capable of being referenced uniquely. Uniqueness can be achieved by specifying a character-string that is not identical to any other character-string, by specifying a REDEFINES reference, or by qualifying the character-string each time it is referenced. Uniqueness of reference is discussed in section 5, Procedure Division.

User-defined words are formed by the letters A through Z, the digits 0 through 9, and the hyphen. Minimum size is one character; maximum size is 30 characters. The hyphen cannot be the first or last character in the string.

At least one letter A through Z must appear in user-defined words, except for those in the sets: paragraph-name, section-name, level-number, and segment-number. Level-number and segment-number must be one or two digits.

System-Names

A system-name is a COBOL word that is used to communicate with the operating system. These words are grouped into the six sets listed in figure 1-3; a given system-name can belong to one and only one set.

System-names are formed the same as user-defined words. Implementor-names, language-names, and mode-names, however, are subject to further restrictions, as discussed with clauses in which these words appear.

Reserved Words

A reserved word is a COBOL word listed in appendix D that can be used in COBOL source programs in a given context. Reserved words must not appear in the program as user-defined words or system-names. Reserved words can be used only as specified in the formats.

Six sets of reserved words exist, as shown in figure 1-3.

Keywords

A keyword is a word whose presence is required when the format in which the word appears is used in a source program. Within each format, such words are indicated by uppercase letters and underlining.

Keywords can be required words in statements and formats, and words with specific functional meaning such as SECTION or NEGATIVE.

Optional Words

An optional word is a word whose presence is not required when the format in which the word appears is used in a source program. The presence of an optional word often adds to the understanding of the purpose of the format, but does not affect the generation of the executable code. Within each format, such words are indicated by uppercase letters without underlining.

Connectives

A connective is a word, or a separator comma, or a separator semicolon. Three types of connectives are:

Qualifier connectives that associate a data-name, a condition-name, a text-name, or a paragraph-name with its qualifier. OF and IN are the only two qualifier connectives.

Series connectives that link two or more consecutive operands. The separator comma and separator semicolon are the only two series connectives.

Logical connectives that form conditions. AND and OR are the only two logical connectives.

Special Registers

A special register is a word whose name references a compiler-generated storage area. The primary use of a special register is storage of information produced in conjunction with a particular COBOL feature. These registers can be referenced in a source program, but must not be defined in the program. Each has an implicit description.

LINE-COUNTER

Exists only when the Report Writer facility is used. One special register exists for each report. See section 6, Report Writer Facility.

PAGE-COUNTER

Exists only when the Report Writer facility is used. One special register exists for each report. See section 6, Report Writer Facility.

DEBUG-ITEM

Exists only when the Debugging facility is used. Only one special register exists for the program. See section 10, Debugging Aids.

LINAGE-COUNTER

Exists for each file in which a LINAGE clause appears in the file FD entry. See the LINAGE clause in the File Description entry discussion of section 4.

HASHED-VALUE

Applicable only when files with direct organization are referenced in a USE FOR HASHING declarative. Must be set to a randomized (that is, hashed) value of a key for a particular record to be accessed. See the direct file organization discussion in section 3 and the USE statement in section 5. Only one special register exists for all direct files.

Figurative Constants

Figurative constant values are generated by the compiler and referenced in a program by a reserved word. These words must not be bounded by quotation marks.

The singular and plural forms are equivalent and can be used interchangeably. The figurative constants and their meanings are shown in table 1-2.

A figurative constant can represent more than one character. The length of the string depends on the context:

In a DISPLAY, STRING, STOP, or UNSTRING statement, a reference to a figurative constant represents a single character.

In either a VALUE clause or in a statement in which the figurative constant is associated with a data item, a figurative constant character-string is repeated until the size of the string equals the size in characters of the data item. The repetition is independent of any JUSTIFIED clause associated with the data item.

Special-Character Words

A special-character word is an arithmetic operator or a relation character.

Arithmetic operators indicate an operation to be performed.

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Relation characters indicate a relation to be tested.

>	Greater than
<	Less than
=	Equal to

TABLE 1-2. FIGURATIVE CONSTANTS

Figurative Constant	Meaning
ZERO ZEROS ZEROES	Depending on the context, represents the numeric value 0 or one or more characters 0.
SPACE SPACES	Represents the character space, which is also known as a blank.
HIGH-VALUE HIGH-VALUES	Represents the character that has the highest ordinal position in the program collating sequence.
LOW-VALUE LOW-VALUES	Represents the character that has the lowest ordinal position in the program collating sequence.
QUOTE QUOTES	Represents the character ". This figurative constant cannot be used in place of a quotation mark to specify nonnumeric or boolean literals. The default character " can be changed to the character ' by specifying the QUOTE IS APOSTROPHE clause in the SPECIAL-NAMES paragraph.
ALL literal	Represents the string of characters comprising the literal. Literal must be either a nonnumeric literal or a figurative constant other than ALL literal. The word ALL is redundant in the form ALL figurative constant.

Within each format, special-character words are required when such portions of the formats are used.

LITERALS

A literal is a character-string whose value is implied either by an ordered set of characters of which the literal is composed or by a reserved word referencing a figurative constant. The three types of literals are numeric, nonnumeric, and boolean. Nonnumeric literals are enclosed in quotation marks. Boolean literals are enclosed in quotation marks and preceded by the character B before the opening quotation mark. Any character-string enclosed in quotation marks not preceded by the character B (including strings that conform to numeric literal or reserved word formats) is a nonnumeric literal.

Boolean Literals

A boolean literal is a character-string delimited on the left by the separator B followed by a quotation mark, and delimited on the right by a quotation mark. The character-string consists only of the boolean characters 0 or 1. The value of a boolean literal is the string of boolean characters itself. All boolean literals are of the category boolean.

Examples of boolean literals are:

B"0011010"
Produces the 7-character boolean value 0011010.

B"110010101"
Produces the 9-character boolean value 110010101.

B"11111001"
Produces the 8-character boolean value 11111001.

Numeric Literals

A numeric literal is a character-string that represents the value of an algebraic quantity. Every numeric literal is category numeric, as discussed with the PICTURE clause. Numeric literals are specified in standard or scientific notation. (Scientific notation is also known as external floating point notation.)

In standard notation, a numeric literal is formed of at least one and no more than 18 digits 0 through 9, and optionally, one sign character and/or one decimal point.

The sign character is the character + or the character -. It must be the leftmost character of the literal. When the sign character is omitted, the literal is positive.

The decimal point is the character period, except when the DECIMAL-POINT IS COMMA clause is specified. It can appear anywhere in the literal except as the rightmost character. When the decimal point is omitted, the literal is an integer.

Some examples of numeric literals in standard notation are:

567 +567 -5.67 +.567

In scientific notation, a numeric literal is formed of the following elements:

At least two and no more than 18 digits 0 through 9

The letter E

One decimal point

Zero, one, or two sign characters

Notation is:

[sign] mantissa E [sign] exponent

The value of the literal is the product of the mantissa times ten raised to the power given by the exponent. For example:

3.E2 represents 3. * 10^2

The sign character can be the character + or the character - . It must be the leftmost character of the mantissa and/or exponent. When the sign character is omitted, the corresponding mantissa or exponent is positive.

The mantissa can be one through 14 digits plus a required decimal point. The decimal point can appear anywhere in the mantissa, including the rightmost or leftmost position.

The exponent can be one through four digits, although the largest values that can be used are expressed through three digits. A zero exponent must be written as a string of one through four zeros.

The range of values that can be specified in scientific notation is .31315130625140E-293 through .12650140831706E+323; thus the number is legal if the exponent is between -279 and +308, regardless of the position of the decimal point.

Numeric literals expressed in scientific notation can be used as follows:

In the Data Division, only for elementary items whose usage is COMP-2.

In the Procedure Division, any place a noninteger numeric literal is allowed.

Some examples of numeric literals in scientific notation are:

+3.14159E-4 .314E+04 10.E-20

Nonnumeric Literals

A nonnumeric literal is a string of 1 through 255 characters delimited on both ends by quotation marks. Any character in the computer character set can be part of a nonnumeric literal.

When a quotation mark is to be part of the literal, it must be represented as two contiguous quotation marks within the character-string. Two quotation marks embedded between other characters produce a single quotation mark within the literal.

The value of a nonnumeric literal in the executable program is the string of characters itself, excluding the delimiters. Punctuation characters are part of the value of the literal and not separators. All nonnumeric literals are category alphanumeric, as discussed with the PICTURE clause.

When the QUOTE IS APOSTROPHE clause is specified in the SPECIAL-NAMES paragraph of the Environment Division, the character apostrophe assumes the role of the character quotation mark.

Examples of nonnumeric literals are:

"PAGEΔ14ΔISΔMISSING. "
    Produces the 20-character value PAGEΔ14ΔIS MISSING. Δ

"+24.50"
    Produces the 6-character value +24.50

" " " EMBEDDEDΔ" " " "ΔQUOTEΔ" " "
    Produces the 20-character value "EMBEDDEDΔ" " ΔQUOTEΔ"

"DEBUG-ITEM"
    Produces the 10-character value DEBUG-ITEM

Figurative-Constant Values

Any place a literal is indicated in a clause or statement format, a figurative constant can be used. Whenever a literal is restricted to having only numeric characters, only ZERO or its equivalent spellings can be used. See table 1-2.

Examples of figurative constant use are:

MOVE QUOTES TO AREA-A
    Fills AREA-A with the repeated character ".

DISPLAY QUOTE "NAME" QUOTE
    Displays "NAME".

MOVE SPACES TO TITLE
    Sets TITLE to all blanks.

MOVE ALL "4" TO COUNT-FIELD
    Fills COUNT-FIELD with the repeated character 4.

IF ALL "5" IS EQUAL TO DFG
    Expands the repeated character 5 to the size of DFG and compares the expanded size with the current value of DFG.

PICTURE CLAUSE CHARACTER-STRINGS

A PICTURE character-string is a combination of characters used as symbols rather than characters. These symbols appear only after the reserved words PICTURE or PIC in a Data Description entry in the Data Division. Punctuation characters within a PICTURE character-string are symbols rather than separators. See section 4, Data Division, for a discussion of the PICTURE clause.

COMMENT-ENTRIES

A comment-entry is a combination of characters following a predefined paragraph-name in the Identification Division. The entry must follow a paragraph-name and a separator period and must be terminated by a separator period. See section 2, Identification Division.

A comment-entry differs from a comment line only in the format in which it appears in the source program.

ARITHMETIC OPERATIONS

Arithmetic operations are specified in the Procedure Division statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. Arithmetic expressions are specified in the COMPUTE statement and in conditional expressions. Refer to the COBOL 5 user's guide for examples of arithmetic operations.

ARITHMETIC EXPRESSIONS

An arithmetic expression is one of the following:

Identifier of an elementary numeric item

Numeric literal

Identifier of elementary numeric item and/or numeric literal separated by an arithmetic operator

Two arithmetic expressions separated by an arithmetic operator

Arithmetic expression enclosed in parentheses

An arithmetic expression must begin with a unary operator, a left parenthesis, or a variable; it must end with a right parenthesis or a variable. Consecutive variables and consecutive binary operators must not appear.

Both binary and unary operators must be preceded and followed by a space. A parenthesis can, but need not, be separated by a space from the element the parenthesis is enclosing.

Identifiers and literals in an arithmetic expression must represent either elementary numeric items or numeric literals on which arithmetic can be performed. Elementary numeric items can be any of the following:

DISPLAY items containing only digits and possibly a sign described by a SIGN IS SEPARATE clause

COMPUTATIONAL item

COMPUTATIONAL-1 item

COMPUTATIONAL-2 item

COMPUTATIONAL-4 item

Special registers LINAGE-COUNTER, HASHED-VALUE, LINE-COUNTER, and PAGE-COUNTER.

Binary arithmetic operators and their meaning are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary arithmetic operators and their meaning are:

+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

Parentheses can be used in arithmetic expressions to specify the order in which elements are to be evaluated. They can eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear and can modify the normal sequence of evaluation. Expressions within parentheses are evaluated first, and, within nested parentheses, from the least inclusive set to the most inclusive set of parentheses. A one-to-one correspondence must exist between a left parenthesis and a right parenthesis.

When parentheses are omitted, or when parenthetical expressions are at the same level of inclusiveness, operations occur in this order:

Unary plus and minus

Exponentiation

Multiplication and division

Addition and subtraction

The order of execution of consecutive operations of the same hierarchical level is from left to right, unless parentheses specify a different order.

EVALUATION OF EXPRESSIONS

Arithmetic operations are carried out in intermediate fields known only to the execution-time routines of the system. Using operands supplied by the program, the routines evaluate the expression or produce an intermediate result. The intermediate result is then moved to any receiving item specified, with any editing specified by the description of the receiving item taking place during the move. When a sending item and a receiving item share a part of their storage areas, the results are unpredictable.

Three types of arithmetic can occur:

Display code arithmetic, in which the character representation (not the algebraic value) of an item is manipulated; a maximum of 18 digits can be accommodated.

Integer arithmetic, in which the algebraic value is manipulated in 48 bits, with bit 60 being the algebraic sign; a maximum of 14 digits can be accommodated.

Floating point arithmetic, in which the algebraic value is manipulated according to mantissa and characteristic conventions; a maximum of 28 digits can be accommodated, with one computer word used for single precision operations and two words for double precision.

Decimal point alignment occurs as required during these operations. All intermediate result fields are established such that no significant digits are lost during operations, except in cases of COMP-2 items in which the data format might produce a slight fraction inaccuracy. (The ROUNDED option should be specified for results involving COMP-2 items.)

Table 1-3 shows the type of arithmetic used for various operands. The table also indicates when conversions from storage format to arithmetic format occur. COMP items, for instance, must be converted to integer format before they are multiplied but COMP-1 items need not be converted.

Different types of operations and arithmetic have restrictions as follows:

Intermediate results in a COMPUTE statement are restricted to 28 significant digits.

The number of decimal places provided for division is not infinite, so the sum of two expressions involving division might be inexact. For example, (1/3) + (2/3) gives the result of 0.999.... To increase the probability of giving an exact result when a computation involves the sum of two expressions involving division of fixed point operands, the division operations are performed so that the operands of the sum operation have one more decimal position than required by the result field. For instance, when the result field in a COMPUTE statement is defined as 9V9 and the operands are 2/3 and 1/3, the computation is performed to two decimal places, not the one place required by the result field; therefore, the result is 1.0.

The situation with exponentiation and subtract is the same as noted above for division and addition.

For evaluations performed in floating point mode, the precision and accuracy of the results are determined by the floating point hardware of the computer itself.

Some special cases of exponentiation are defined as follows, assuming x is to be raised to the nth power.

x	n	Result
0	>0	0
0	≤ 0	Size error condition
$\neq 0$	0	1
0	not integer	Size error condition

TABLE 1-3. ARITHMETIC USED FOR OPERAND PAIRS

Operand	Operator	Operand: DISPLAY or COMP	Operand: COMP-1 or COMP-4	Operand: COMP-2
DISPLAY or COMP	+	D	I1	F
	-	D	I1	F
	*	I2	I2	F
	/	F	F	F
	**	I3	I3	F
COMP-1 or COMP-4	+	I1	I1	F
	-	I1	I1	F
	*	I2	I2	F
	/	F	F	F
	**	I3	I3	F
COMP-2	+	F	F	F
	-	F	F	F
	*	F	F	F
	/	F	F	F
	**	F	F	F

Legend:

D	Display code arithmetic; maximum of 18 digits
F	Floating point arithmetic
I1	Integer arithmetic; F if size of intermediate result determined by aligning operands on the decimal points is 15 or more digits
I2	Integer arithmetic; F if sum of operand sizes is 15 or more digits
I3	Integer arithmetic if integer value; otherwise, F

BOOLEAN OPERATIONS

Boolean operations are specified in Procedure Division statements which include the boolean operators BOOLEAN-AND, BOOLEAN-OR, BOOLEAN-EXOR, or BOOLEAN-NOT. Boolean expressions are used in the COMPUTE statement and in relation conditions.

BOOLEAN EXPRESSIONS

A boolean expression can appear in one of the following forms:

A boolean variable or identifier of a boolean data item

A boolean literal consisting of the characters 0 or 1

A combination of boolean variables and/or literals separated by a boolean operator

Two boolean expressions separated by a boolean operator

A boolean expression enclosed in parentheses

Boolean expressions are made up of boolean operands (boolean variables or literals), boolean operators, and parentheses. The permissible combinations of boolean variables, boolean literals, boolean operators, and parentheses are given in table 1-4.

TABLE 1-4. COMBINATION OF SYMBOLS IN BOOLEAN EXPRESSIONS

First Symbol \ Second Symbol	Boolean Variable or Literal	BOOLEAN-OR AND EXOR	BOOLEAN-NOT	(	)
Boolean variable or literal	-	P	-	-	P
BOOLEAN-OR AND EXOR	P	-	P	P	-
BOOLEAN-NOT	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Note:

P indicates a permissible pair
- indicates an invalid pair

Three binary boolean operators can be used in boolean expressions: BOOLEAN-AND, BOOLEAN-OR, and BOOLEAN-EXOR. One unary boolean operator, BOOLEAN-NOT, can also be used in boolean expressions. The boolean operators and their corresponding meanings are as follows:

Boolean Operator	Meaning
BOOLEAN-AND	Boolean conjunction (logical AND)
BOOLEAN-OR	Boolean inclusive disjunction (logical OR)
BOOLEAN-EXOR	Boolean exclusive disjunction (logical exclusive OR)
BOOLEAN-NOT	Boolean negation (logical complement)

All boolean operators are reserved words and must be both preceded and followed by the separator space.

Whenever two boolean expressions are separated only by BOOLEAN-AND, BOOLEAN-OR, or BOOLEAN-EXOR, a new boolean expression is formed with BOOLEAN-AND, BOOLEAN-OR, or BOOLEAN-EXOR as the boolean operator. Whenever a boolean expression is immediately preceded by BOOLEAN-NOT, a new boolean expression is formed with BOOLEAN-NOT as the boolean operator.

A boolean expression must begin with a left parenthesis, a boolean variable, a boolean literal, or the operator BOOLEAN-NOT. A boolean expression must end with a right parenthesis, a boolean variable, or a boolean literal.

Parentheses are used in boolean expressions to specify the order in which boolean expression elements are evaluated. Expressions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. Parentheses must appear in balanced pairs.

When parentheses are omitted, or when parenthetical expressions are at the same level of inclusiveness, operations occur in the following order:

Negation (BOOLEAN-NOT)

Conjunction (BOOLEAN-AND)

Disjunction (BOOLEAN-OR and BOOLEAN-EXOR)

Parentheses are used to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear. Parentheses are also used to modify the normal hierarchical sequence of evaluation within an expression. The order of execution of consecutive operations of the same hierarchical level is from left to right, unless parentheses specify a different order.

EVALUATION OF EXPRESSIONS

Boolean operations are performed in intermediate fields. The intermediate result is moved to any receiving item specified. Assignment of boolean variable or identifier values can be achieved with the COMPUTE statement.

A boolean operation involves the conjunction, inclusive disjunction, exclusive disjunction, or negation performed on operands of type boolean. The results of these operations for all possible combinations of boolean character values are listed in table 1-5.

Boolean operations are performed as shown in the following examples:

P = 0101011101 and Q = 1100110101

P BOOLEAN-AND Q

The conjunction of P and Q is formed as follows:

```
P   0101011101
Q   1100110101
    0100010101      Result of conjunction
```

P BOOLEAN-OR Q

The inclusive disjunction of P and Q is formed as follows:

```
P   0101011101
Q   1100110101
    1101111101      Result of disjunction
```

P BOOLEAN-EXOR Q

The exclusive disjunction of P and Q is formed as follows:

```
P   0101011101
Q   1100110101
    1001101000      Result of disjunction
```

TABLE 1-5. RESULTS OF BOOLEAN OPERATIONS

Boolean Character Value P	Q	Conjunction P BOOLEAN-AND Q	Inclusive Disjunction P BOOLEAN-OR Q	Exclusive Disjunction P BOOLEAN-EXOR Q	Negation BOOLEAN-NOT Q
1	1	1	1	0	0
1	0	0	1	1	1
0	1	0	1	1	0
0	0	0	0	0	1

BOOLEAN-NOT Q

The negation of Q is formed as follows:

```
Q   1100110101
    0011001010        Result of negation
```

If the operands are of equal length, the operation proceeds by conjoining, disjoining, or exclusively disjoining boolean characters in corresponding character positions starting from the high order position and continuing to the lower order position. If the operands are of unequal length, the operation proceeds as though the shorter operand were extended on the low order end by boolean zeros to make the operands of equal size.

CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that can be tested in a program to determine alternative paths of action. They are specified in the Procedure Division statements IF, PERFORM, and SEARCH.

Two categories of conditions are associated with conditional expressions: simple conditions and complex conditions. Paired parentheses can be used in both simple and complex conditions without changing the category of the condition.

SIMPLE CONDITIONS

Simple conditions include the following:

Class, which determines whether an operand is purely numeric or purely alphabetic or neither

Condition-name, which compares a variable and a predetermined value associated with a condition-name

Relational, which compares values of two operands

Sign, which determines the sign of an arithmetic expression

Switch-status, which tests the ON or OFF status of an external or internal software switch

Class Condition

The class condition determines whether an operand is numeric or alphabetic. Class condition format is:

$$\text{identifier IS } \left[\underline{\text{NOT}}\right] \left\{\begin{matrix}\underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}}\end{matrix}\right\}$$

The operand belongs to the numeric class if one of the following criteria is met:

USAGE IS COMP-2 is specified.

USAGE IS COMP-4 is specified.

USAGE IS COMP-1 is specified and the leftmost 12 bits of the word containing the item consist of all ones or all zeros; (that is, no item has been moved to the item that destroyed its COMP-1 format).

The operand consists entirely of the digits 0 through 9 and the presence of any operational sign agrees with the description of the sign. For example, any item described by SIGN IS SEPARATE must have a separate, not an overpunch, sign.

The NUMERIC test cannot be used with an item described as alphabetic or as a group item containing elementary items with operational signs.

The operand belongs to the alphabetic class only when the contents consist entirely of the characters A through Z and the space. The ALPHABETIC test cannot be used with an item described as numeric.

Condition-Name Condition

The condition-name condition determines whether or not the specified item is equal in value to one of the predefined values for the condition. Format of this condition is simply:

condition-name

The condition-name must be a level 88 item in the Data Division that is associated with a constant or a range of values.

The condition is true when the value is either within the range specified, including both ends of the range, or is equal to the constant specified. Any sign character must agree with the description of the condition-name for the condition to be true.

Relation Condition

A relation condition (figure 1-4) determines the relative magnitude of two operands. Table 1-6 shows the type of operands that can be compared.

Figure 1-4 shows the acceptable formats for the comparison of two operands. Format 1 is used for comparison of operands other than those of the boolean class. At least one variable must be used in the condition expression. Format 2 is used for comparison of operands of the boolean class.

Any two numeric class operands can be compared, regardless of the USAGE clause descriptions. Comparison is made with respect to the algebraic value of the operands. The number of digits represented is not significant. Zero is a unique value, regardless of its sign. Unsigned operands are considered positive.

A numeric class operand can be compared with a nonnumeric operand only when both operands have the same USAGE clause descriptions and the numeric operand is an integer data item, integer literal, or an arithmetic expression containing only integer operands and/or literals.

Two nonnumeric operands or one numeric and one nonnumeric operand are compared with respect to a collating sequence in which each character has a particular value. The collating sequence can be established in four ways: by default, by the ALPHABET clause in the SPECIAL-NAMES paragraph, by the COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, or by the SET statement. Comparison is made from left to right of each operand. When operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by spaces to make the operands of equal size.

```
Format 1

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

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


Format 2

                        { IS [NOT] EQUAL TO }
  boolean expression-1  { IS [NOT] =        }  boolean expression-2
                        { IS UNEQUAL TO     }
                        { EQUALS            }
```

Figure 1-4. Relation Condition Format

TABLE 1-6. RELATION CONDITION COMPARISON

Item Type	Item Type															
	Literal				DISPLAY			COMP-1 or COMP-4								
	Nonnum.	Integer	Nonint.	Boolean	Integer	Nonint.	Boolean	Integer	Nonint.	COMP-2	INDEX	Index data name	Group	A, AN, edited	Arithmetic Expression	Boolean Expression
Literal:																
Nonnumeric	-	-	-	-	S	-	-	-	-	-	-	-	P	P	-	-
Integer value	-	-	-	-	A	A	-	A	A	A	O	O	S	S	A	-
Noninteger value	-	-	-	-	A	A	-	A	A	A	O	O	-	-	A	-
Boolean	-	-	-	-	-	-	B	-	-	-	-	-	-	-	-	B
DISPLAY item:																
Numeric integer	S	A	A	-	A	A	-	A	A	A	O	O	S	S	A	-
Numeric noninteger	-	A	A	-	A	A	-	A	A	A	O	O	-	-	A	-
Boolean	-	-	-	B	-	-	B	-	-	-	-	-	-	-	-	B
COMP-1 or COMP-4 item:																
Integer	-	A	A	-	A	A	-	A	A	A	O	O	-	-	A	-
Noninteger	-	A	A	-	A	A	-	A	A	A	O	O	-	-	A	-
COMP-2 item	-	A	A	-	A	A	-	A	A	A	O	O	-	-	A	-
INDEX item	-	O	O	-	O	O	-	O	O	O	O	O	-	-	-	-
Index data name	-	O	O	-	O	O	-	O	O	O	O	O	-	-	-	-
Group item	P	S	-	-	S	-	-	-	-	-	-	-	P	P	-	-
Alphabetic, alphanumeric, or edited	P	S	-	-	S	-	-	-	-	-	-	-	P	P	-	-
Arithmetic expression	-	A	A	-	A	A	-	A	A	A	-	-	-	-	A	-
Boolean expression	-	-	-	B	-	-	B	-	-	-	-	-	-	-	-	B

-	Illegal	S	Sign removed, then character compare
A	Algebraic value compare	O	Occurrence number algebraic compare
P	Character compare	B	Boolean compare

The first nonmatching character pair establishes the greater operand, with the higher collating sequence value determining the greater operand.

Two boolean operands can be compared only for equality or inequality. Boolean operands can only be compared with other boolean operands. A boolean operand is a single boolean item, a boolean literal, or a boolean expression. If the operands are of equal size, comparison proceeds by comparing boolean characters in corresponding boolean character positions. Comparison starts from the high order position and continues until either a pair of unequal boolean characters is encountered or the low order position of the operands is reached. The operands are equal if all pairs of boolean characters compare equally; otherwise, the operands are unequal. If the operands are of unequal length, comparison proceeds as though the shorter operand were extended on the right by boolean character zeros to make the operands of equal size.

Switch-Status Condition

The switch-status condition determines whether a specified switch is ON or OFF. Six external switches are available, identified as SWITCH-1 through SWITCH-6. Internal switches SWITCH-7 through SWITCH-126 are also available for testing. Format of this condition is:

condition-name

The condition-name must be specified in the SPECIAL-NAMES paragraph of the Environment Division and equated to an ON or OFF status for a particular switch. The result of the test is true only when the particular switch associated with the condition-name is set to the value indicated by the status. External switches can be manipulated by the SWITCH control statement in the job deck, by operator action at any time, or by the terminal user during a program pause (a STOP literal statement). Both external and internal switches can be manipulated by the SET statement within a program.

Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The arithmetic expression must contain at least one reference to a variable. Format of this condition is:

$$\text{arithmetic-expression IS } \left[\underline{\text{NOT}}\right] \left\{\begin{array}{l}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{array}\right\}$$

The result is true when:

Keyword	Value
POSITIVE	Greater than zero
NEGATIVE	Less than zero
ZERO	Zero
NOT POSITIVE	Zero or negative
NOT NEGATIVE	Zero or positive
NOT ZERO	Greater than or less than zero

COMPLEX CONDITIONS

Complex conditions are:

Simple conditions and/or complex conditions combined with a logical operator AND or OR.

Simple conditions and/or complex conditions negated with the logical operator NOT.

The logical operators and their meanings are:

AND	Logical conjunction; true only when both conditions are true
OR	Logical inclusive OR; true when at least one condition is true
NOT	Logical negation; true when the condition is false

Complex conditions can be negated simple conditions or combined conditions.

Negated Simple Conditions

A simple condition is negated by the logical operator NOT. The general format for a negated simple condition is:

<u>NOT</u> simple-condition

Combined and Negated Combined Conditions

A combined condition results from connecting conditions with one of the logical operators AND or OR. The general format for a combined condition is:

$$\text{condition} \left\{\left\{\begin{array}{l}\underline{\text{AND}}\\ \underline{\text{OR}}\end{array}\right\} \text{ condition}\right\} \ldots$$

Condition can be any of the following:

Simple condition

Negated simple condition

Combined condition

Negated combined condition in which the logical operator NOT is followed by a combined condition enclosed within parentheses

Combinations of the above

Parentheses can be used to specify the order in which individual conditions are to be evaluated. Conditions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. Left and right parentheses must be paired.

When parentheses are omitted, or when parenthesized conditions are at the same level of inclusiveness, evaluation proceeds in the following order:

Values are established for arithmetic expressions.

Truth values for simple conditions are established in this order:

relation
class
condition-name
switch-status
sign

Truth values for negated simple conditions are established.

Truth values for combined conditions are established for AND logical operators, then for OR logical operators.

Truth values for negated combined conditions are established.

Consecutive operations of the same level are evaluated from left to right.

Combined conditions can be abbreviated such that the relation-condition and relational-operator need not be repeated for a consecutive sequence of identical comparisons. The general format for an abbreviated combined relation condition is:

$$\text{relation-condition} \left\{ \left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} \left[\underline{\text{NOT}} \right] \right. \quad \left[\text{relational-operator}\right] \text{ object} \left. \right\} \ldots$$

The word NOT is interpreted according to the word following NOT:

NOT is part of the relational operator when followed by: GREATER, >, LESS, <, EQUAL, or =.

NOT is a logical operator otherwise and results in a negated relation condition.

Examples of abbreviated combined and negated combined relation conditions and their expanded equivalents are:

a>b AND NOT<c OR d
is equivalent to
((a>b) AND (a NOT<c)) OR (a NOT<d)

a NOT EQUAL b OR c
is equivalent to
(a NOT EQUAL b) OR (a NOT EQUAL c)

NOT a = b OR c
is equivalent to
(NOT (a = b)) OR (a = c)

NOT (a GREATER b OR<c)
is equivalent to
NOT ((a GREATER b) OR (a<c))

a / b NOT EQUAL c AND NOT d
is equivalent to
((a / b) NOT EQUAL c) AND
(NOT ((a / b) NOT EQUAL d))

NOT (a NOT>b AND c AND NOT d)
is equivalent to
NOT ((((a NOT>b) AND (a NOT>c)) AND
NOT (a NOT>d)))

ITEM AND TABLE REFERENCES

All items referenced within the Procedure Division must be uniquely identified. User-defined words can be qualified to achieve uniqueness. Items in a table defined by an OCCURS clause in the Data Division can be uniquely identified by subscripts or indexes.

Data-names and condition-names can be referenced uniquely through a combination of qualification, indexing, or subscripting shown below under the heading Identifier Definition. In the formats of this manual, the term identifier refers to a data-name uniquely referenced; condition-name refers to a condition-name uniquely referenced.

Reference modification can be used to reference a portion of a data item of the alphanumeric or boolean class.

Qualification of user-defined words is allowed in all formats of the Procedure Division unless qualification is specifically prohibited by a specific format.

QUALIFICATION OF USER-DEFINED WORDS

Qualification can be used to uniquely identify user-defined words or particular special registers. Qualification is performed by following the user-defined name and separator with the reserved word OF or the reserved word IN and a qualifying word. OF and IN are logically equivalent and can be used interchangeably.

The element that can be used as a qualifier depends on the item being qualified, as shown in figure 1-5. A qualifying element can, itself, be qualified. A data-name cannot be subscripted when it is being used as a qualifier. A user-defined name can be qualified even though it does not require qualification. If more than one combination of qualifiers ensures uniqueness, any such set can be used.

In format 1, each qualifier must be the name associated with a level indicator or the name of a group item to which the user-defined word item is subordinate. Qualifiers are specified in the order of successively more inclusive levels in the hierarchy. The cd-name can only be used under NOS.

Formats 2 through 6 show the format for qualifying each of the following:

Paragraph-name

Text-name

LINAGE-COUNTER special register

PAGE-COUNTER and LINE-COUNTER special registers

Data-name in a report

Format 1

$$\left\{\begin{matrix}\text{data-name-1}\\ \text{condition-name}\end{matrix}\right\} \left(\begin{matrix} \left\{\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{data-name-2}\right\} \ldots \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\}\left\{\begin{matrix}\text{file-name}\\ \text{cd-name}^{\dagger}\end{matrix}\right\}\right] \\ \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\}\left\{\begin{matrix}\text{file-name}\\ \text{cd-name}^{\dagger}\end{matrix}\right\}\right]\end{matrix}\right)$$

Format 2

$$\text{paragraph-name} \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{section-name}\right]$$

Format 3

$$\text{text-name} \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{library-name}\right]$$

Format 4

$$\underline{\text{LINAGE-COUNTER}} \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{file-name}\right]$$

Format 5

$$\left\{\begin{matrix}\underline{\text{PAGE-COUNTER}}\\ \underline{\text{LINE-COUNTER}}\end{matrix}\right\} \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{report-name}\right]$$

Format 6

$$\text{data-name-3} \left\{\begin{matrix} \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{data-name-4}\right] \ldots \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{report-name}\right] \\ \left[\left\{\begin{matrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{matrix}\right\} \text{report-name}\right]\end{matrix}\right\}$$

†Can only be used under NOS.

Figure 1-5. Qualification Format

If qualification is used to make a condition-name unique, the associated conditional variable can be used as the first qualifier. The hierarchy of names associated with the conditional variable or the conditional variable itself must be used to make the condition-name unique.

TABLE ITEM IDENTIFICATION

References can be made to individual elements within a table by specifying indexing or subscripting. Indexing is usually more efficient than subscripting. Indexing is not allowed where subscripting is not allowed. Indexing and subscripting cannot be mixed within a given reference, for strict ANSI usage. In order to provide compatibility with future ANSI standards, mixing of indexing and subscripting is allowed. Forty-eight levels of subscripting are also allowed.

During execution, the program is responsible for establishing a table reference that is within the bounds of the table. Use of the DB=SB parameter on the compiler call, however, causes the system to diagnose out-of-bounds table references during execution.

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names. Figure 1-6 shows the general format of subscripting.

$$\left\{\begin{matrix}\text{data-name}\\ \text{condition-name}\end{matrix}\right\} \text{(subscript-1} \left[\text{, subscript-2} \; [\text{, subscript-3}] \ldots [\text{, subscript-n}]\right] \text{)}$$

Figure 1-6. Subscripting Format

The subscript can be represented either by an integer or a data-name that defines an elementary numeric integer item. The subscript can, but need not, have a plus sign. Data-name cannot be an index data item nor a COMP-2 item. Data-name can be qualified, but no subscript itself can be subscripted. In Report Writer references, neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

The lowest possible subscript value is 1, which points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, 4, and so forth. The highest permissible subscript value is the maximum number of occurrences of the item as specified in the OCCURS clause.

Any subscripts must be enclosed in parentheses. They can, but need not, be separated by commas. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. Refer to the COBOL 5 user's guide for examples of subscripting.

Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY clause in the definition of a table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index.

The value of an index corresponds to the occurrence number of an element in the associated table. An index must be initialized by execution of a SET statement, a SEARCH ALL statement, or a format 4 PERFORM statement before it is used as a table reference.

Figure 1-7 shows the general format for indexing. Two types of indexing are possible:

> Direct indexing occurs when an index-name is used as a subscript.
>
> Relative indexing occurs when an index-name used as a subscript is followed by a literal that increments or decrements the current value of the index-name item.

When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

The value of index-name during execution must be within the range of the possible occurrence number of an element in the associated table.

IDENTIFIER DEFINITION

In the formats in this manual, the term identifier is used to reflect a unique reference to a data-name. If the data-name itself is not unique in the program, the term identifier implies that the data-name is referenced uniquely through a syntactically correct combination of qualifiers, subscripts, or indexes.

Figure 1-8 shows the general format for identifiers. This format also applies to condition-names. The cd-name can only be used under NOS.

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

REFERENCE MODIFICATION

Reference modification is a method of accessing a part of a data item. The format of reference modification is shown in figure 1-9. Reference modification creates a unique data item which is a subset of the data item referenced by data-name. Data-name must reference a data item that has DISPLAY usage. Leftmost-character-position and length must be arithmetic expressions. Each character of the data item referenced by data-name is assigned an ordinal number, which is incremented by one (starting at one) from the leftmost position to the rightmost position. If the Data Description entry for data-name contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item. Figure 1-10 provides examples of reference modification.

Reference modification is allowed anywhere an identifier referencing a data item of the class alphanumeric or boolean is permitted, unless otherwise specified. If the data item referenced by data-name is described as numeric, numeric-edited, alphanumeric, or alphanumeric-edited, it is operated upon as if it were an alphanumeric data item. Otherwise, the unique data item has the same class and category as that of the data-item referenced by data-name. The unique data item is considered an elementary data item without the JUSTIFIED clause. Reference modification for an operand is evaluated immediately after evaluation of any subscripts or indexes that are specified for that operand.

$$\left\{\begin{array}{l}\textbf{data-name}\\ \textbf{condition-name}\end{array}\right\}\ (\ \left\{\begin{array}{l}\textbf{index-name-1}\ \left[\{\pm\}\ \textbf{literal-2}\right]\\ \textbf{literal-1}\end{array}\right\}$$

$$\left[\ ,\left\{\begin{array}{l}\textbf{index-name-2}\ \left[\{\pm\}\ \textbf{literal-4}\right]\\ \textbf{literal-3}\end{array}\right\}\ \left[\ ,\left\{\begin{array}{l}\textbf{index-name-3}\ \left[\{\pm\}\ \textbf{literal-6}\right]\\ \textbf{literal-5}\end{array}\right\}\right]\right]\ )$$

Figure 1-7. Indexing Format

Format 1

```
                ┌ ┌ OF ┐              ┐       ┌ ┌ OF ┐ ┌ file-name    ┐ ┐
data-name-1     │ ┤    ├ data-name-2 │ . . . │ ┤    ├ ┤ cd-name†     ├ │   [(subscript-1 [, subscript-2 [, subscript-3]]
                └ └ IN ┘              ┘       └ └ IN ┘ └ report-name  ┘ ┘
                                                                              . . . [, subscript-n] )]
```

Format 2

```
                ┌ ┌ OF ┐              ┐       ┌ ┌ OF ┐ ┌ file-name    ┐ ┐   ┌   ┌ index-name-1 [{±}  literal-2] ┐
data-name-1     │ ┤    ├ data-name-2 │ . . . │ ┤    ├ ┤ cd-name†     ├ │   │ ( ┤                                ├
                └ └ IN ┘              ┘       └ └ IN ┘ └ report-name  ┘ ┘   └   └ literal-1                     ┘

   ┌   ┌ index-name-2  [{±}  literal-4] ┐   ┌   ┌ index-name-3  [{±}  literal-6] ┐ ┐ ┐   ┐
   │ , ┤                                ├   │ , ┤                                ├ │ │ ) │
   └   └ literal-3                      ┘   └   └ literal-5                      ┘ ┘ ┘   ┘
```

†Can only be used under NOS.

Figure 1-8. Identifier and Condition-name Unique Reference Format

```
                                               ┌ length ┐
data-name (leftmost-character-position :       ┤        ├ )
                                               └ END    ┘
```

Figure 1-9. Reference Modification Format

The leftmost-character-position specifies the ordinal position of the leftmost character of the unique data item created in relation to the leftmost character of the data item referenced by data-name. The leftmost-character-position must be a positive nonzero integer less than or equal to the number of characters in the data item referenced by data-name. If an illegal leftmost-character-position is used, the results are undefined; however, if DB=RF is specified on the compiler call statement, an execution diagnostic results and the run aborts.

The length specifies the size of the data item to be used in the operation. The length must be a positive nonzero integer. The sum of leftmost-character-position and the length, minus one, must be less than or equal to the number of characters in the data item referenced by data-name. If this sum is greater than the allowed value, or is negative or zero, the results are undefined; however, if DB=RF is specified on the compiler call statement, an execution diagnostic results and the run aborts. If the END phrase is specified, the unique data item extends from and includes the character identified by leftmost-character-position up to and including the rightmost character of the data item referenced by data-name.

Reference modification is not allowed in the following:

The MOVE CORRESPONDING, ADD CORRESPONDING, SUBTRACT CORRESPONDING, and USE FOR DEBUGGING statements

The table name in a SEARCH statement

The receiving data item following the word INTO in a STRING statement

The sending data item in an UNSTRING statement

```
IDENTIFICATION DIVISION.
      .
      .
      .
WORKING-STORAGE SECTION.
01  ITEM PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
01  A  PICTURE 99  VALUE IS 2.
01  B  PICTURE 99  VALUE IS 5.
01  D.
02  C  PICTURE X(10) OCCURS 3 TIMES.
01  NUM  PICTURE 99 VALUE IS 24.
      .
      .
      .
PROCEDURE DIVISION.
PAR-1.
    MOVE ITEM( 2 : 5 ) TO C( 1 ).†
    MOVE ITEM( 3 : A + B ) TO C( 2 ).††
    MOVE C( A ) ( 4 : 3 ) TO ITEM( NUM : 3 )†††
        .
        .
        .
    STOP RUN.
```

†Moves five characters, beginning with the second position character of ITEM. The characters BCDEF are moved to C(1).

††Moves the number of characters represented by A plus B, beginning with the third position character of ITEM. The characters CDEFGHI are moved to C(2).

†††Moves three characters of C(2), beginning with its fourth position. The characters FGH are moved to positions 24 through 26 of ITEM, and ITEM becomes ABCDEFGHIJKLMNOPQRSTUVWFGH.

Figure 1-10. Reference Modification Example

IDENTIFICATION DIVISION

The Identification Division identifies the source program. The division has one required paragraph and five optional paragraphs. Format of the Identification Division is shown in figure 2-1.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  program-name.
[AUTHOR.  [comment-entry]]
[INSTALLATION.  [comment-entry]]
[DATE-WRITTEN.  [comment-entry]]
[DATE-COMPILED.  [comment-entry]]
[SECURITY.  [comment-entry]]
```

Figure 2-1. Identification Division Format

The division header is:

IDENTIFICATION DIVISION.

The header must appear on a separate line, beginning in area A. The separator period must terminate the header.

The first paragraph must be PROGRAM-ID. All other paragraphs are optional, but, if present, must appear in the order shown.

The AUTHOR, INSTALLATION, DATE-WRITTEN, and SECURITY paragraphs are documentary only.

NOTE

Because of anticipated changes in this product, use of comment entries is not recommended. For guidelines, see appendix F.

The comment-entry of the last five paragraphs in the division can be any combination of characters from the computer character set shown in appendix A. When the comment extends to more than one line, the hyphen must not appear in the indicator area.

PROGRAM-ID PARAGRAPH

The PROGRAM-ID paragraph (figure 2-2) gives the program an entry point for use by the loader of the operating system, in addition to giving the program a name to identify output listings.

```
PROGRAM-ID.  program-name.
```

Figure 2-2. PROGRAM-ID Paragraph Format

Program-name is a user-defined word. The first seven characters of the word become the program entry point name.

When the program is a subprogram that will be executed as a result of a CALL statement, the program-name must begin with a letter. If multiple programs are combined to form a run unit, the first seven characters in the program-name of each of the programs must be unique.

DATE-COMPILED PARAGRAPH

The DATE-COMPILED paragraph (figure 2-3) causes the current date to be inserted in the source program listing produced during compilation. Any comment-entry specified is replaced during compilation with the current date.

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

Figure 2-3. DATE-COMPILED Paragraph Format

Since the date of compilation appears in the heading of each page of the compilation output listings, the paragraph is redundant.

ENVIRONMENT DIVISION

Environment Division documents the configuration of the computers used to process the program and supplies operating system interface information. The division has two sections, both of which can be omitted. The division header is always required, however. A skeleton of the Environment Division is shown in figure 3-1.

The division header is:

ENVIRONMENT DIVISION.

The header must appear on a separate line, beginning in area A. The separator period must terminate the header.

```
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER.  [source-computer-entry]]
[OBJECT-COMPUTER.  [object-computer-entry]]
[SPECIAL-NAMES.  [special-names-entry]]]
[INPUT-OUTPUT SECTION.
[FILE-CONTROL.  {file-control-entry} . . .]
[I-O-CONTROL.  input-output-control-entry]]
```

Figure 3-1. Environment Division Skeleton

CONFIGURATION SECTION

The Configuration Section contains three optional paragraphs: SOURCE-COMPUTER, OBJECT-COMPUTER, and SPECIAL-NAMES. The entire section can be omitted.

The section header is:

CONFIGURATION SECTION.

The header must appear on a separate line, beginning in area A. The separator period must terminate the section header. A header without following paragraphs can appear.

SOURCE-COMPUTER PARAGRAPH

The SOURCE-COMPUTER paragraph documents the computer upon which the program is to be compiled and selects the debugging facility. The paragraph is optional. The skeleton for the SOURCE-COMPUTER paragraph is shown in figure 3-2. The paragraph header is:

SOURCE-COMPUTER.

The computer-name clause must appear first; other clauses can be in any order.

```
SOURCE-COMPUTER.  [computer-name clause
    DEBUGGING MODE clause.]
```

Figure 3-2. SOURCE-COMPUTER Paragraph Skeleton

Computer-name Clause

The computer-name clause is documentary only. Computer-name must conform to the syntax of a system-name. The system-name CYBER or CYBER-170 is recommended for its documentary value. Clause format is:

computer-name

DEBUGGING MODE Clause

The DEBUGGING MODE clause specifies that all USE FOR DEBUGGING declarative statements and all debugging lines in the program are to be compiled as executable code. See section 10, Debugging Aids.

OBJECT-COMPUTER PARAGRAPH

The OBJECT-COMPUTER paragraph documents the computer on which the program is to be executed, selects the collating sequence for nonnumeric comparisons, and specifies a segment section limit. The paragraph is optional. The skeleton for the OBJECT-COMPUTER paragraph is shown in figure 3-3. The paragraph header is:

OBJECT-COMPUTER.

The computer-name clause must be first; other clauses can appear in any order.

```
OBJECT-COMPUTER.  [computer-name clause
[, COLLATING SEQUENCE clause]
[, SEGMENT-LIMIT clause].]
```

Figure 3-3. OBJECT-COMPUTER Paragraph Skeleton

Computer-name Clause

The computer-name clause is optional and documentary only. Computer-name must conform to the syntax of a system-name. The system-name CYBER or CYBER-170 is recommended for its documentary value. Clause format is:

computer-name

COLLATING SEQUENCE Clause

The COLLATING SEQUENCE clause (figure 3-4) specifies the collating sequence used to determine the truth value of all nonnumeric comparisons in a program, including comparisons for:

Sort or merge operations

Keys of records in files with random organization

Control break processing of report

```
PROGRAM COLLATING SEQUENCE IS alphabet-name
```

Figure 3-4. COLLATING SEQUENCE Clause Format

The values of the figurative constants HIGH-VALUE and LOW-VALUE are also affected by this clause.

If the clause is omitted, the collating sequence associated with the native character set is used for all nonnumeric comparisons, unless another sequence is specified for a sort. The native character set is defined by the installation.

The collating sequence specified in the OBJECT-COMPUTER paragraph can be overridden during execution.

> For nonnumeric comparisons, the collating sequence can be changed by execution of a format 3 SET statement.

> For sort or merge operations, the collating sequence can be changed by including the COLLATING SEQUENCE clause in the SORT statement or the MERGE statement. It also can be changed by execution of a format 3 SET statement.

The alphabet-name of the clause identifies the selected collating sequence. It must be the same as an alphabet-name specified in the SPECIAL-NAMES paragraph. Refer to the ALPHABET clause discussion.

SEGMENT-LIMIT Clause

The SEGMENT-LIMIT clause defines permanent segments. See section 8, Segmentation Facility.

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph (figure 3-5) relates implementor-names to user-specified mnemonic-names and relates alphabet to character sets and/or collating sequences. It can also be used to specify the alternative representation of the quotation mark and currency symbol, the alternative use of the decimal point and comma as punctuation in numeric edited data items, and the default definition of the sign convention for signed numeric display data items. The paragraph header is:

SPECIAL-NAMES.

```
SPECIAL-NAMES.[implementor-name clause] . . .
[; ALPHABET clause] . . . [; CURRENCY SIGN clause]
[; DECIMAL-POINT clause]  [; QUOTE clause]
[; SIGN clause]  [; SUB-SCHEMA clause]
[; SWITCH-n clause] . . . .
```

Figure 3-5. SPECIAL-NAMES Paragraph Skeleton

Implementor-name Clause

The implementor-name clause (figure 3-6) establishes a mnemonic-name for reference in Procedure Division statements. More than one implementor-name clause can appear.

> When the implementor-name is a file name of "TERMINAL" or CONSOLE, the clause associates the file with a mnemonic-name for reference in an ACCEPT statement or DISPLAY statement.

> When the implementor-name is a single carriage control character enclosed in quotation marks, the clause associates the character with a mnemonic-name for reference in the ADVANCING phrase of a WRITE statement.

```
implementor-name IS mnemonic-name
```

Figure 3-6. Implementor-name Clause Format

File Name as implementor-name

Any valid logical file name can be specified as an implementor-name and equated to a mnemonic-name. The mnemonic-name is then used in ACCEPT and DISPLAY statements. When the names INPUT, TERMINAL, and OUTPUT are specified as implementor-names, they should be enclosed in quotation marks (for strict ANSI usage) because they are COBOL reserved words. They are accepted, however, as legal implementor names, if they are specified without the quotes.

See the ACCEPT and DISPLAY statements in the Procedure Division for processing possible with particular implementor-names: "INPUT", "OUTPUT", "OUTPUT-C", "TERMINAL", "TERMINAL-C", and CONSOLE.

Files specified in this clause cannot be named in a SELECT clause in a File-Control entry, with the exception of "INPUT" and "OUTPUT".

Carriage Control Character as implementor-name

An implementor-name that consists of a single carriage control character enclosed in quotation marks is a carriage control character associated with a mnemonic-name. A reference to the mnemonic-name in the ADVANCING phrase of a WRITE statement causes that carriage control character to be added to the beginning of the line being output.

Full carriage control characters are shown in the operating system reference manuals. The more commonly used characters and their effects are:

Character	Effect
space	Single space
1	Eject page before print
0	Double space
-	Triple space
+	Suppress space before print

ALPHABET Clause

The ALPHABET clause (figure 3-7) relates an alphabet-name to a collating sequence or to a character set. The clause is required in the following instances:

COLLATING SEQUENCE clause is used in an OBJECT-COMPUTER paragraph of the Environment Division.

CODE-SET clause is used in an FD entry in the Data Division.

Format 3 SET statement is used in the Procedure Division.

COLLATING SEQUENCE clause or a CODE-SET clause is used in a SORT statement or a MERGE statement in the Procedure Division.

The clause can appear several times to specify different character code sets or collating sequences for different operations within the program.

NOTE

Refer to appendix F for recommendations on the use of collating sequences.

A system-defined character set and associated collating sequence is selected by one of the keywords shown in the clause format. The collating sequence for the character set NATIVE is redefined through the literal phrase of the clause.

Each of the following system-defined code sets and collating sequences is shown in appendix A:

ASCII-64

CDC 64-character subset of the 128-character code set defined in American National Standard X3.4-1968, Code for Information Interchange

CDC-64

CDC 64-character code set for the computers on which COBOL 5 is supported

EBCDIC

CDC 64-character subset of the 128-character code set defined for IBM System 360/370 computers

NATIVE

Can be either CDC-64 or ASCII-64, depending on installation option

STANDARD-1

Equivalent to ASCII-64

UNI

Collating sequence associated with the UNIVAC 1100 series computer

The character that has the highest ordinal position in the program collating sequence named in the ALPHABET clause is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE. Likewise, the character that has the lowest position (or the first character specified for the lowest position) is associated with the figurative constant LOW-VALUE.

```
                                  { STANDARD-1                                                            }
                                  { NATIVE                                                                }
                                  { CDC-64                                                                }
                                  { ASCII-64                                                              }
                                  { EBCDIC                                                                }
                                  { UNI                                                                   }
                                  {                                                                       }
                                  {            [ { THRU    }  literal-2                     ]             }
ALPHABET alphabet-name IS         {  literal-1 [ { THROUGH }                                ]             }
                                  {            [ ALSO literal-3 [, ALSO literal-4] . . .    ]             }
                                  {                                                                       }
                                  {  [            [ { THRU    }  literal-6                   ] ]          }
                                  {  [ , literal-5 [ { THROUGH }                              ] ] . . .   }
                                  {  [            [ ALSO literal-7 [, ALSO literal-8] . . .   ] ]          }
```

Figure 3-7. ALPHABET Clause Format

Literal Phrase

The literal phrase of the ALPHABET clause redefines the collating sequence for the native character set. The first character identified in the phrase becomes the lowest position in the collating sequence; subsequent characters assume ascending contiguous positions. Figurative constants cannot be specified in this clause.

Characters in the native character set can be identified in one of two ways:

When the phrase specifies a numeric literal, the character is identified as the character occupying that position in the native character set. For example, the following clause causes the 11th, 12th, and 13th characters in the set (J, K, and L) to occupy contiguous ascending positions in the collating sequence:

ALPHABET MYALPH 11, 12, 13

Numeric values in the literal phrase must be 1 through 64, (64 is the number of characters in the native character set). Values greater than 64 are ignored. The order of characters in the native set is shown in the table named Standard Character Sets in appendix A.

When the phrase specifies a nonnumeric literal, the character is identified directly. For example, the following clause causes characters J, K, and L to occupy contiguous ascending positions in the collating sequence:

ALPHABET MYALPH "J", "K", "L"

Each character can be specified as its own nonnumeric literal; alternatively, characters in contiguous ascending collating sequence positions can be specified as a single nonnumeric literal. For example, the following clause causes characters J, K, and L to occupy contiguous ascending positions in the collating sequence:

ALPHABET MYALPH "JKL"

No character can be specified more than once in the literal phrase. The order in which literals appear in the literal phrase specifies the ordinal positions of the characters in ascending sequence. The first character identified has the lowest position and is associated with the figurative constant LOW-VALUE.

The THRU phrase, or its equivalent THROUGH, assigns a range of contiguous characters in the native character set to ascending positions in the collating sequence. Contiguous characters, beginning with the character identified by literal-1 and ending with the character identified by literal-2, are assigned successive ascending positions. Literal-1 and literal-2 can specify characters in either ascending or descending order. For example, the following clause causes characters M, L, K, J, and I to occupy the first five positions in the collating sequence, with M in the lowest position:

ALPHABET MYBET "M" THRU "I"

The ALSO phrase allows more than one character to be assigned to the same position in the collating sequence. Therefore, if this phrase is specified, literal-1, literal-3, and literal-4 are assigned the same position in the collating sequence.

If the ALSO phrase has been specified and an OPEN statement is executed for an indexed file or a file with alternate keys (under Advanced Access Methods only), a CYBER Record Manager error results and the run is aborted. Use of the ALSO phrase should be avoided since it causes a noticeable degradation in execution time.

CURRENCY SIGN Clause

The CURRENCY SIGN clause (figure 3-8) specifies a character for the currency symbol. In a source program, the character specified is accepted in a PICTURE clause as a currency symbol in addition to the standard currency symbol. During execution, only the character specified by this clause is produced for edited items. When the clause is omitted, only the standard currency symbol is accepted in a PICTURE clause. (The standard currency symbol is $ or #, depending on the character set in use.)

<u>CURRENCY</u> SIGN <u>IS</u> literal

Figure 3-8. CURRENCY SIGN Clause Format

In the clause, the literal must specify a single nonnumeric character. Any character can be specified except PICTURE clause editing symbols or the characters: C D L R) (" = or space.

DECIMAL-POINT Clause

The DECIMAL-POINT clause (figure 3-9) interchanges the functions of the comma and the period in the character-string of the PICTURE and in numeric literals. The effects of this clause are discussed with the PICTURE clause in the Data Division.

<u>DECIMAL-POINT</u> <u>IS</u> <u>COMMA</u>

Figure 3-9. DECIMAL-POINT Clause Format

QUOTE Clause

The QUOTE IS APOSTROPHE clause (figure 3-10) interchanges the functions of the quote mark (") and the apostrophe ('). When the clause is specified, the apostrophe character is recognized in the source program as the delimiter of nonnumeric literals and in the executing program as the value of the figurative-constant QUOTE. The clause must precede any apostrophe used as a quote mark in a program.

$$\left\{\begin{array}{l}\underline{\textbf{QUOTE}}\ \textbf{IS}\\ \underline{\textbf{QUOTES}}\ \textbf{ARE}\end{array}\right\}\ \underline{\textbf{APOSTROPHE}}$$

Figure 3-10. QUOTE Clause Format

The COBOL compiler call has a similar parameter. Use of the clause is discussed in section 12 with the APO parameter.

SIGN CONTROL Clause

The SIGN CONTROL clause (figure 3-11) specifies the default position and mode of representation of the operational sign. It is discussed with the SIGN clause of the Data Description entry in the Data Division.

```
SIGN CONTROL IS { LEADING  }
                { TRAILING }

        [SEPARATE CHARACTER]
```

Figure 3-11. SIGN CONTROL Clause Format

SUB-SCHEMA Clause

The SUB-SCHEMA clause specifies the name of the sub-schema that has descriptions of data in a data base. See section 14, Sub-Schema Facility.

SWITCH-n Clause

The SWITCH-n clause (figure 3-12) associates the ON and OFF status of an external or internal software switch with:

A condition-name for reference in an IF statement.

A mnemonic-name for reference in a format 4 SET statement.

To comply with strict ANSI usage, the clause, when specified, must immediately follow an implementor-name clause if one is present. If one is not present, the SWITCH-n clause must be the first clause in the paragraph.

The SWITCH-n clause can appear as often as necessary in a source program. The same condition-name, however, should not appear in more than one SWITCH-n clause, because the name cannot legally be qualified in the Procedure Division.

The switches that can be specified in the clause are SWITCH-1 through SWITCH-126. SWITCH-6 has special meaning for debugging, as discussed in section 10, Debugging Aids. SWITCH-1 through SWITCH-6 are equivalent to the external switches 1 through 6 that can be defined for each job in the system. SWITCH-7 through SWITCH-126 reference internal COBOL switches that are set to the OFF status prior to the execution of a COBOL program.

Prior to program execution, the external switches can be set on or off by control statements in the job or by commands from a terminal. The central site operator also has a command that can change external switch setting.

During program execution, the status of an external or internal switch can be altered by a format 4 SET statement that references mnemonic-name. Alteration of an external switch is global to a job. An internal switch is local to a job step.

In the clause, mnemonic-name is the name by which the switch is referenced within the program. If the switch is not to be referenced in a format 4 SET statement, mnemonic-name is not required.

Condition-name-1 and condition-name-2 are user-defined words. They represent, respectively, the condition-name associated with the ON status and OFF status for a switch. Condition-name is used to reference the status of a switch set externally to the program or within the program.

INPUT-OUTPUT SECTION

The Input-Output Section specifies information needed to control transmission and handling of data between a storage device and the executing program. The section consists of two paragraphs, FILE-CONTROL and I-O-CONTROL, both of which can be omitted. If the entire section is omitted, no file input or output is possible, except through ACCEPT and DISPLAY statements.

The section header is:

INPUT-OUTPUT SECTION.

The header must appear on a separate line, beginning in area A. The separator period must terminate the section header. A header can appear without following paragraphs.

FILE-CONTROL PARAGRAPH

The FILE-CONTROL paragraph contains File-Control entries that name each file referenced in the program and specify other file-related information. The information specified in a File-Control entry depends on the file organization.

```
           ( IS mnemonic-name [ ON STATUS IS condition-name-1  ]                       )
           (                  [ OFF STATUS IS condition-name-2 ]                       )
SWITCH-n  <                                                                             >
           ( ON STATUS IS condition-name-1   [, OFF STATUS IS condition-name-2]        )
           ( OFF STATUS IS condition-name-2  [, ON STATUS IS condition-name-1]         )
```

Figure 3-12. SWITCH-n Clause Format

The organization of a file is established at the time the file is created and cannot be subsequently changed. All references to an established file must specify, either implicitly or explicitly, the correct file organization. The file organizations are:

Sequential	Direct
Relative	Actual-key
Indexed	Word-address

All except relative files are implemented according to the CYBER Record Manager file organizations of similar names. Relative files are implemented through word-addressable files with fixed-length records. The CYBER Record Manager manuals, in particular the user guides, present full discussions of the physical and logical structures of the different organizations. Consult these manuals for information about the implications of parameters of the USE clause or the FILE control statement, alternative means of specifying file structure or processing, and utilities available for specific file organizations.

All of the file organizations, except sequential, have a unique key associated with each record in the file: system interpretation of the key item distinguishes the different organizations. Use of mass storage, processing available, and access time differ for each organization. Table 3-1 contrasts some organization characteristics. All except sequential files must reside on rotating mass storage.

NOTE

Refer to appendix F for recommendations on the use of file organizations.

TABLE 3-1. FILE ORGANIZATION SUMMARY

Characteristic	File Organization					
	Sequential	Indexed	Direct	Actual-Key	Relative	Word-Address
File residence	Mass storage or tape	Mass storage	Mass storage	Mass storage	Mass storage	Mass storage
Record lengths	Fixed or variable	Fixed or variable	Fixed or variable	Fixed or variable	Fixed	Fixed or variable
Key contents	None	Integer, decimal, character string	Integer, decimal, character string	Block and record number	Record number ordinal	Record word number
Alternate key	Not applicable	Yes	Yes	Yes	No	No
Physical file structure	Contiguous records; blocked on tape	Index blocks, data blocks; padding in blocks but no record gaps	Data blocks preallocated	Data blocks continuous record slots	Continuous record slots	Continuous word slots
Logical file structure	Continuous records; blocked on tape	Sorted by ascending key value	Random according to hashed key value	Sorted by block and record slot	Sorted by record ordinal	Record at word address
Updating possible	Add to end, replace on mass storage	Replace, delete, insert	Replace, delete, insert	Replace, delete, insert	Replace, delete, insert	Replace, insert
Sequential access	By position	Sorted	Unsorted	Sorted by block	Sorted by ordinal	By position
Random access	None	By key	By key	By key	By key	By key
Utilities available	COPY copy; FORM restructure	COPY copy; FORM restructure; IXGEN/MIPGEN add or delete alternate keys; ESTMATE select block size; SISTAT statistics	COPY copy; FORM restructure; IXGEN/MIPGEN add or delete alternate keys; CREATE creation; KYAN key analysis	COPY copy; FORM restructure IXGEN/MIPGEN add or delete alternate keys	COPY copy	COPYBR copy

The clauses in the File-Control entry are specified in alphabetical order after the discussions of the different file organizations. SELECT must be the first clause; other clauses can appear in any order with the exception of the RELATIVE KEY clause.

Sequential File Organization

Sequential files can exist on tape, cards, or mass storage. Files connected to a terminal must be sequential. The location of any given file record is immediately following the previously written record and immediately before the subsequently written record. The order in which records are referenced in WRITE statements determines the order in which they can be retrieved by READ statements.

Records in the file can be fixed or varying length according to the Record Description entries for the file.

Sequential files offer processing advantages when all file records are always processed. The program is responsible for any ordering of the file.

The skeleton of a File-Control entry for a sequential file is shown in figure 3-13.

```
SELECT clause
[; ACCESS MODE IS SEQUENTIAL]
 ; ASSIGN clause
[; FILE STATUS clause]
[; ORGANIZATION IS SEQUENTIAL]
[; RESERVE clause]
[; USE clause]
```

Figure 3-13. Sequential Organization File-Control Entry Skeleton

Relative File Organization

A record key for a relative file is an integer representing the ordinal of the record in the file. Key value 22 represents the 22nd record in the file; key value 5678 represents the 5678th record in the file, and so forth. The RELATIVE KEY clause defines the key item for the file.

Since all records in the file have the same length, a record with any given key value has a predetermined location in the file. (If the Record Description entries specify variable length records, the system uses the length of the largest record for allocating file space for each record.) If the first record written to the file has a value of 890, the file on mass storage has, at that time, enough storage space to accommodate records with key values 1 through 889 also. The system does not retrieve information from an ordinal position unless a valid record has previously been written to that position.

Relative files have a fixed access time. Only one mass storage access is required by the system to return a record to a program. Unused space exists on mass storage when key values for relative file records are not contiguous and do not begin with record ordinal 1.

The skeleton of a File-Control entry for a relative file is shown in figure 3-14.

```
SELECT clause
[[; ACCESS MODE clause]
    [, RELATIVE KEY clause]]
 ; ASSIGN clause
[; FILE STATUS clause]
 ; ORGANIZATION IS RELATIVE
[; RESERVE clause]
[; USE clause]
```

Figure 3-14. Relative Organization File-Control Entry Skeleton

Indexed File Organization

When Advanced Access Methods (AAM) indexed sequential are installed, one of two types of indexed file [illegible] is selected: initial indexed sequential or extended indexed sequential. The type of indexed file to be used is specified through the ORG parameter of the USE clause or the FILE control statement. ORG=OLD indicates that a file is in initial indexed sequential format; ORG=NEW indicates that a file is in extended indexed sequential format.

If extended indexed sequential files have been installed ORG=NEW, which is the default, can be specified to indicate that the file is in the extended indexed sequential format. If a file in initial indexed sequential format is to be created or accessed and extended indexed sequential files have been installed ORG=OLD must be specified.

If extended indexed sequential files have not been installed, the ORG parameter must not be specified.

Records in an indexed file can be fixed or varying length in any combination. Each record has a unique primary key and might have alternate records keys within it. The primary key item is defined by the RECORD KEY clause; alternate record key items are defined by ALTERNATE RECORD KEY clauses.

A primary key can be a string of characters or a number with or without a decimal point or sign. Usually, a primary key value would be a name or number that has logical meaning to the program or programmer. The system uses the numeric value or collating sequence value of the primary key to determine the location of the record in the file: records always are in physical and logical order by ascending primary key values. When an indexed file is read sequentially from beginning to end, all records are retrieved in sorted order according to primary key value.

The first record written to the file can have any primary key value. The system places the next record written before or after the first record as necessary to maintain the records in sorted order. Storage for the file is not preallocated; rather a data block is created to hold records as the need arises. Overflow records, as such, do not exist. The system splits an existing data block when it is full, so that the physical and logical ordering is always maintained. The size of a data block is established by the BLOCK CONTAINS clause of the Data Division.

In addition to data blocks, indexed files have index blocks created and maintained by the system to hold information about data block location and contents. Indexes are arranged in levels: as long as the number of index levels does not increase, record access time does not increase when the number of records in the file increases.

Alternate record keys can be defined for indexed files. The system indexes the alternate record keys on a separate file that must be preserved by the job that creates or updates the file. Only the primary key determines record position in the data blocks.

Indexed files offer advantages for processing large files that are to be accessed randomly by key as well as sequentially in sorted order. They can be structured so that large inserts do not increase the time needed to access a file record. When an indexed file is being created, the sequential access mode should be used for most efficient processing.

The File-Control entry skeleton for an indexed file is shown in figure 3-15.

```
SELECT clause
[; ACCESS MODE clause]
[; ALTERNATE RECORD KEY clause]
 ; ASSIGN clause
[; FILE STATUS clause]
 ; ORGANIZATION IS INDEXED
 ; RECORD KEY clause
[; USE clause] .
```

Figure 3-15. Indexed Organization File-Control Entry Skeleton

Direct File Organization

When Advanced Access Methods (AAM) direct files are installed, one of two types of direct file organization is selected: initial direct or extended direct. The type of direct file to be used is specified through the ORG parameter of the USE clause or the FILE control statement. ORG=OLD indicates that a file is in initial direct file format; ORG=NEW indicates that a file is in extended direct file format.

If extended direct files have been installed, ORG=NEW, which is the default, can be specified to indicate that the file is in the extended direct file format. If a file in initial direct format is to be created or accessed and extended direct files have been installed, ORG=OLD must be specified.

If extended direct files have not been installed, the ORG parameter must not be specified.

Records in a direct file can be fixed or varying length in any combination. Each record has a unique primary key and might have alternate record keys within it. The primary key item is defined by the RECORD KEY clause; alternate record key items are defined by ALTERNATE RECORD KEY clauses.

A primary key for a direct file can be any data item. Usually, a key value would be a name or number that has logical meaning to the program or programmer. The system, however, considers the key value to represent a string of bits that must be hashed to the number of a home block in the file. The record location is determined by the hashed key.

Hashing is the process of manipulating the key contents according to some constant formula. The system supplies a hashing routine, although the programmer has the option of supplying his own hashing routine through the USE FOR HASHING declarative. The same hashing routine must be used for all records for the life of a file. A key analysis utility is available to determine the distribution of records in home blocks for particular key values. (The utility is documented in the AAM reference manual.) An optimal hashing routine distributes all records evenly among home blocks so that no overflow records need exist.

Home blocks are preallocated storage, with the number of blocks established by the BLOCK COUNT clause that is required in the File-Control entry. The size of all home blocks is the same and is established by a BLOCK CONTAINS clause in the FD entry in the Data Division. When a primary key value hashes to a home block that is filled, an overflow record exists. Overflow records are accommodated in another home block or in an overflow block, with the system following AAM parameters to select the block. Overflow records require another mass storage access before they can be returned to the program.

Direct files offer processing advantages when records are always accessed randomly by key. The file can be read sequentially from beginning to end, but the records are not sorted by key. Direct files are useful for rapid access to large files.

Alternate record keys can be defined for direct files. The system indexes the alternate record keys on a separate file that must be preserved by the job that creates or updates the file of home blocks. Only the primary key determines the home block the record occupies.

The skeleton of a File-Control entry for a direct file is shown in figure 3-16.

```
SELECT clause
[; ACCESS MODE clause]
[; ALTERNATE RECORD KEY clause]
 ; ASSIGN clause
 ; BLOCK COUNT clause
[; FILE STATUS clause]
 ; ORGANIZATION IS DIRECT
 ; RECORD KEY clause
[; USE clause] .
```

Figure 3-16. Direct Organization File-Control Entry Skeleton

Actual-Key File Organization

When Advanced Access Methods (AAM) actual-key files are installed, one of two types of actual-key file organization is selected: initial actual-key or extended actual-key. The type of actual-key file to be used is specified through the ORG parameter of the USE clause or the FILE control statement. ORG=OLD indicates that a file is in initial actual-key file format; ORG=NEW indicates that a file is in extended actual-key file format.

If extended actual-key files have been installed, ORG=NEW, which is the default, can be specified to indicate that the file is in the extended actual-key file format. If a file in initial actual-key format is to be created or accessed and extended actual-key files have been installed, ORG=OLD must be specified.

If extended actual-key files have not been installed, the ORG parameter must not be specified.

NOTE

Because of anticipated changes in this product, use of actual-key files is not recommended. For guidelines, see appendix F.

As with indexed and direct files, records in actual-key files can be fixed or varying length. Each record has a unique primary key and might have alternate record keys within it. A primary key for an initial actual-key file is an integer that represents a block number and a record slot number within that block. For extended actual-key files the primary key is a record number which AAM converts to the storage location of the record. Primary key values do not have logical meaning, but rather are the physical location of the record. The primary key item is defined by the RECORD KEY clause; alternate record key items are defined by ALTERNATE RECORD KEY clauses.

Each data block has a fixed number of record slots. The size of the data blocks and the number of slots per block are established in the Data Division FD entry with the RECORD clause and BLOCK CONTAINS clause. Since record lengths can vary, overflow records can exist when a data block based on average record size calculations is full. The system performs all overflow record handling internally, so that the program always accesses records by the primary key.

The program can, but need not, generate primary keys when a file is being created or records are being added to an existing file. If the key item has a zero value when the record is written, the system generates a key, writes the record, and returns the key value to the key item. The program is responsible for preserving the primary key values returned to the program for future record access by primary key values.

The system performs no indexing of primary keys. A user application might use actual-key files in conjunction with an indexed file to create an alternate key access capability when the system supplied alternate key capability is not used.

Alternate record keys can be defined for actual-key files. The system indexes the alternate record keys on a separate file that must be preserved by the job that creates or updates the file. Alternate record keys do not specify the record location; they can be items with logical meaning to the program or programmer.

The skeleton of a File-Control entry for an actual-key file is shown in figure 3-17.

```
SELECT clause
[; ACCESS MODE clause]
[; ALTERNATE RECORD KEY clause]
 ; ASSIGN clause
[; FILE STATUS clause]
 ; ORGANIZATION IS ACTUAL-KEY
 ; RECORD KEY clause
[; USE clause] .
```

Figure 3-17. Actual-Key Organization File-Control Entry Skeleton

Initial Actual-Key File Key Format

Values assigned to actual-key file primary keys must specify a block number and a slot within that block. The key is considered as a bit string of length determined by the number of digits in the description multiplied by 6. The lower bits are assumed to indicate a position in a block; the upper bits are the block number. Block numbers and record slot numbers begin with zero. Block 0, record 0 is created and reserved for AAM. The first record position in the file has a key representing block 0, record 1. New blocks must be created in order; that is, block 2 cannot be created before block 1. All record positions need not be filled before the next block is created.

User key generation is usually simpler when the blocking factor determined by the BLOCK CONTAINS clause is a power of 2, although any factor can be used. If the blocking factor is 16, for instance, the key value for the first record in the second block must be 16 to obtain the bit pattern representing block 1, record 0, as shown below.

$$\underbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}}_{\text{block number}}\underbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{0}}_{\text{slot number}}$$

Extended Actual-Key File format

Values assigned to extended actual-key file primary keys represent a record number. AAM converts the record number to the storage location of the record. When CYBER Record Manager is supplying the keys, it returns sequential record numbers to the primary key. The user need not be concerned with block and slot numbers. When supplying the keys, the user must remember to create new blocks in order. For instance, block 2 cannot be created before block 1. All record positions need not be filled, however, before the next block is created.

Word-Address File Organization

A key for a record in a word-address file is the number of the word in the file at which the record begins. These files are considered to be a set of contiguous 60-bit, 10-character-per-word words numbered from 1 through the end of the file. A key value of 457 causes a record to be read or written from word 457 through the record end.

The system reads or writes starting at the word specified by the key; it does no checking to determine if valid information exists at that location. The WORD-ADDRESS KEY clause defines the key item.

All records need not be the same length, but fixed length records offer ease of address calculations in the program. The record type for word-address files is always U (undefined). The user is responsible for setting the record length before reading or writing any variable length records. After a read or write operation, the word-address key is updated to the next available word by using the provided record length and the last key.

If the first record written to the file has a value of 4567, the file on mass storage has, at that time, a length of at least 4567 words plus the length of the current record. The program is responsible for determining the validity of any data read and for specifying the correct record boundaries.

Word-address files are most often used in system application situations. They offer immediate access and low system overhead. No file preallocation occurs. The File-Control entry skeleton for a word-address file is shown in figure 3-18.

```
SELECT clause
[; ACCESS MODE clause]
 ; ASSIGN clause
[; FILE STATUS clause]
 ; ORGANIZATION IS WORD-ADDRESS
[; RESERVE clause]
[; USE clause]
 ; WORD-ADDRESS KEY clause
```

Figure 3-18. Word-Address Organization File-Control Entry Skeleton

ACCESS MODE Clause

The ACCESS MODE clause (figure 3-19) specifies the manner in which records are to be operated upon within the system. If the clause is omitted, ACCESS IS SEQUENTIAL is assumed.

```
                  ( SEQUENTIAL )
ACCESS MODE IS    { RANDOM     }
                  ( DYNAMIC    )
```

Figure 3-19. ACCESS MODE Clause Format

For relative files, the ACCESS MODE clause must immediately precede any RELATIVE KEY clause.

For files with sequential organization, only SEQUENTIAL can be specified. For files being created with indexed organization, SEQUENTIAL should be specified for most efficient processing. For all other organizations, access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

SEQUENTIAL

Allows access by sequential position only.

RANDOM

Allows access by key values only. The programmer controls the sequence in which records are accessed by specifying the key value for the record needed.

DYNAMIC

Allows access by sequential position and by key values intermixed. The programmer changes from sequential access to random access at will by using appropriate formats of the input-output statements.

The ACCESS MODE clause, in conjunction with the OPEN statement of the Procedure Division, establishes the input-output statements that can be used for a given file. Table 3-2 shows ACCESS MODE clause and OPEN statement interaction.

ALTERNATE RECORD KEY Clause

The ALTERNATE RECORD KEY clause (figure 3-20) specifies a key item that defines an alternate record key for indexed, direct, or actual-key files. The number and description of alternate record keys can change from program to program when the IXGEN utility of AAM is used with initial AAM file organizations. The MIPGEN utility can be used for extended AAM file organizations. The clause must be repeated for each alternate record key. As many as 255 alternate record keys can be defined for a given file.

Data-name-1 must specify a data item defined within a Record Description entry for the file. The relative location of the item within the record and the item description must not change for the life of the file.

An item of category alphanumeric or numeric can be specified as an alternate record key; that is, the item must be described by a PICTURE clause containing at least one X or with a PICTURE clause containing a combination of the symbols 9, S, V, and P. If data-name-1 specifies an elementary numeric item, its picture must not contain an S unless the item is also described by USAGE IS COMP-1. Alternate keys cannot exceed 255 characters in length, or, if numeric, a numeric value that can be stored in a single memory word.

```
ALTERNATE RECORD KEY IS data-name-1 [WITH DUPLICATES [ASCENDING]]

[ ( OMITTED )                                                  ]
[ { USE     }  WHEN data-name-2 CONTAINS CHARACTER FROM literal ]

[                       ( SPACES ) ]
[ OMITTED WHEN KEY IS   { ZEROS  } ]
```

Figure 3-20. ALTERNATE RECORD KEY Clause Format

TABLE 3-2. INPUT-OUTPUT STATEMENTS ALLOWED BY ACCESS MODE AND OPEN MODE

File Access Mode		Statement \ Open Mode	File Organization												
			Sequential				Relative			Indexed, Direct, or Actual-Key			Word-Address		
			I	O	I-O	E	I	O	I-O	I	O	I-O	I	O	I-O
File Access Mode	Sequential	READ	X		X		X		X	X		X	X		X
		WRITE		X		X		X			X			X	
		REWRITE			X				X			X			
		START					X		X	X		X			
		DELETE							X			X			
	Random	READ					X		X	X		X	X		X
		WRITE						X	X		X	X		X	X
		REWRITE							X			X			
		START													
		DELETE							X			X			
	Dynamic	READ					X		X	X		X	X		X
		WRITE						X	X		X	X		X	X
		REWRITE							X			X			
		START					X		X	X		X			
		DELETE							X			X			

I	INPUT specified in OPEN statement.
O	OUTPUT specified in OPEN statement.
I-O	I-O specified in OPEN statement.
E	EXTEND specified in OPEN statement.
X	Statement is allowed for organization and open mode.
blank	Statement is not allowed for organization and open mode.

Data-name-1 can reference an item described with the OCCURS clause; however, a subscript must not be specified. Such an item is called a repeating group, whether or not it is a group item. This facility allows the alternate key in a record to have more than one value. When the record is written, each occurrence of the repeating group becomes an alternate key value for the record.

When a READ or START statement specifies the repeating group alternate key as the key of reference, the record area value that is used for comparison is the first occurrence of the repeating group. This value need not be the first occurrence of the repeating group in the record being read or in the record at which the file is positioned; it can be any occurrence of the alternate key. See the READ and START statements in section 5.

Data-name-1 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item named in the RECORD KEY clause or of any other data-name item in an ALTERNATE RECORD KEY clause associated with this file; that is, an alternate record key cannot begin in the same location as the primary key or any other alternate record key. In non-ANSI usage, however, an alternate record key can begin in the same character position as the primary key or another alternate record key if the keys are not the same length; that is, no two keys can define exactly the same storage location.

DUPLICATES Phrase

The DUPLICATES phrase specifies that more than one record can contain the same value for a specific alternate record key. If the DUPLICATES option of the ALTERNATE RECORD KEY clause is not specified, alternate key values must be different for a specific key for each record in the file. If not, a duplicate value encountered on a write causes execution of the INVALID KEY clause.

The order of primary key values within a set of duplicates depends on the presence or absence of the ASCENDING option of the DUPLICATES phrase. The ASCENDING option specifies that records are to be returned in ascending primary key order when records within a set of duplicates are read sequentially. The ascending order is established by the collating sequence in effect for alphanumeric keys. If the ASCENDING option is omitted, primary key values associated with a given alternate key value are maintained in the order in which they were written to the file (first in, first out). Refer to the Multiple-Index Processor (MIP) user's guide for further discussion on index file structure.

The ASCENDING option must be specified if the alternate key is within a repeating group and the file organization is the extended AAM format. Performance in file updating is considerably enhanced when ASCENDING is specified; however, its usage is non-ANSI.

OMITTED/USE Phrases

The OMITTED/USE WHEN and the OMITTED WHEN KEY phrases define specific conditions that determine when an alternate record key is to be included or is not to be included in the alternate key index. Alternate key values that are not necessarily included in the alternate key index are called sparse keys. Refer to the Multiple-Index Processor user's guide for a further discussion of sparse keys.

The OMITTED/USE WHEN and the OMITTED WHEN KEY phrases can be used to reduce storage space and processing time as follows:

Keys that are never used to access data records can be excluded. These keys are the result of sparse control characters.

Keys that have values of blank or zero are excluded. These keys are the result of null suppression.

Both phrases apply to extended AAM file organizations only; they are not applicable to initial AAM file organizations. When neither phrase is included, all alternate key values are included in the alternate key index.

The USE WHEN CONTAINS phrase specifies that an alternate record key is to be included in the alternate key index if data-name-2 contains a character included in the literal. Data-name-2 is part of the data record and is the control character. If the control character indicated by data-name-2 is not present in the literal, the key is included in the index. Conversely, the OMITTED WHEN CONTAINS phrase specifies that an alternate record key is not to be included in the alternate key index if data-name-2 specifies a character included in the literal. If the control character indicated by data-name-2 is not present in the literal, the key is included in the index. The item referenced by data-name-2 must specify a data item defined within the fixed length portion of the Record Description entry for the file. The item must be described as alphanumeric and must be one character in length. Data-name-2 can appear in more than one ALTERNATE RECORD KEY clause for a file as long as the same data-name-2 is used in each reference. The literal must be one to 36 characters in length. Only the characters A through Z and digits 0 through 9 can be used. No character can appear more than once within the literal.

The OMITTED WHEN KEY phrase is used to specify that a key is not to be included in the alternate key index if it contains all spaces and has a usage of DISPLAY or if it contains all zeros and has a usage of COMP-1 or COMP-2.

When both the USE WHEN and OMITTED WHEN KEY phrases are used in an ALTERNATE RECORD KEY clause, the condition specified in the OMITTED WHEN KEY phrase has precedence over the condition specified in the USE WHEN phrase.

ASSIGN Clause

The ASSIGN clause (figure 3-21) specifies the name by which the operating system identifies the file. This clause is required for all files.

```
ASSIGN TO implementor-name-1

     [,implementor-name-2] . . .
```

Figure 3-21. ASSIGN Clause Format

Implementor-name-1 specifies the name by which the operating system identifies the file-name used in the SELECT clause. It must be one through seven letters or digits beginning with a letter. Implementor-name-1 is the logical file name parameter used on control statements for functions such as permanent file catalog or magnetic tape request. Files can be assigned special dispositions and other characteristics by using implementor-names that correspond to the special-named files of the operating system. See the operating system reference manual for information about files with logical file names such as INPUT, OUTPUT, PUNCH, PUNCHB, P80C, and P8.

Since INPUT and OUTPUT are reserved words, they should be enclosed in quotes if they are used as implementor-names (for strict ANSI usage). However, they are accepted as legal implementor-names if they are specified without the quotes. Any implementor-name that duplicates the file-name or that duplicates any other data-name in the program must be enclosed in quotes. See the ACCEPT and DISPLAY statements for information about implementor-names associated with the terminal or operator console.

Implementor-name-2 is the logical file name of the index file for an indexed, direct, or actual-key file that has alternate record keys. It must be one through seven letters or digits beginning with a letter. The job is responsible for preserving the index file between jobs and for making it available on mass storage to any job that updates the file specified by file-name or to any job that reads the file specified by file-name using alternate record keys. Implementor-name-2 is the logical file name used on control statements for permanent file functions.

Any additional implementor-names are checked for syntax, but they do not affect program execution.

More than one file can be assigned to the same implementor-name and the same implementor-name can be assigned in a main program and a subprogram. Only one of the files assigned to an implementor-name can be open at a given time, however.

BLOCK COUNT Clause

The BLOCK COUNT clause (figure 3-22) specifies the number of home blocks to be allocated when a file with direct organization is being created. The clause is required when the clause ORGANIZATION IS DIRECT appears and the file is opened for OUTPUT. It is ignored at all other times.

```
BLOCK COUNT IS { integer   }
               { data-name }
```

Figure 3-22. BLOCK COUNT Clause Format

The data item referenced by data-name or the integer must specify an unsigned integer. The value defines the number of home blocks in the file. During processing, records are assigned to one of the blocks by a user-supplied or system-supplied hashing algorithm. When the system hashing routine is used, best results are obtained if the number of blocks is a prime number.

FILE STATUS Clause

The FILE STATUS clause (figure 3-23) specifies a data item in which the system returns a status code during execution of the statements: CLOSE, DELETE, OPEN, READ, REWRITE, START, and WRITE. When the clause is omitted, the program can detect file-related errors and the at-end condition through a USE AFTER ERROR PROCEDURE declarative procedure or through an AT END or an INVALID KEY phrase imperative-statement in the Procedure Division, depending on the particular error or condition involved.

```
FILE STATUS IS data-name
```

Figure 3-23. FILE STATUS Clause Format

The data-name specified must be a two-character alphanumeric category data item that is defined in other than the File Section or Report Section. If the file is an External file, data-name must be defined in the Common-Storage Section. Status values that might be returned and their meanings are in table 3-3.

The use of FILE STATUS for detection of errors or file position is limited for data base files. The only file position returned by CDCS is end-of-file. CDCS errors are not reflected in the FILE STATUS data item. Refer to section 14, Sub-Schema, for further information.

TABLE 3-3. FILE STATUS VALUES

Value	Meaning
00	Successful execution.
02	Successful execution: READ — Key value for the current key of reference is equal to the value of that same key in the next record within the current key of reference. WRITE or REWRITE — Record just written created a duplicate key value for at least one alternate record key.
10	At-end condition.
21	Invalid key condition from a sequence error: WRITE — Indexed file must be created with ascending prime key values. REWRITE — Primary key cannot change between format 1 READ and REWRITE.
22	Invalid key condition because duplicate key values are not allowed.
23	Invalid key condition because no record with a specified key exists in the file.
24	Invalid key condition because of a boundary violation of a relative or indexed file.
30	Permanent error, such as parity error, transmission error, or mass storage unavailable.
34	Permanent error because of boundary violation; limit established by the FLM parameter of a FILE control statement reached.
90	CYBER Record Manager error other than those listed above.
99	COBOL detected error.

AT END, Status 10

The at-end condition might occur as a result of the execution of a READ, RETURN, or SEARCH statement. In general, the condition indicates the end of a file, but can also indicate the end of a table.

When the at-end condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected. The following actions take place:

> The FILE STATUS data item, if any, is set to 10.
>
> Control transfers to any imperative-statement specified by an AT END phrase in SEARCH or the input-output statement; any procedure specified by a format 1 USE statement declarative for the file is not executed.

When an AT END phrase is not included in the input-output statement, any procedure specified by a format 1 USE statement declarative executes.

If neither an AT END phrase nor a USE AFTER ERROR PROCEDURE statement declarative is specified, and if no FILE STATUS data item is associated with the file, the program aborts when an at-end condition occurs.

INVALID KEY, Status 2n

The invalid key condition might occur as a result of DELETE, READ, REWRITE, START, or WRITE statement execution when a file is being written or updated by key. In general, the condition indicates an illegal operation, such as a write of two records with the same key, but could also indicate that the key data item is incorrectly formatted.

When the invalid key condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected. The following actions take place:

> The FILE STATUS data item, if any, is set to one of the values associated with an invalid key condition.
>
> Control transfers to any imperative-statement specified by an INVALID KEY phrase in the input-output statement; any procedure specified by a format 1 USE statement declarative for the file is not executed.

When an INVALID KEY phrase is not included in the input-output statement, any procedure specified by a format 1 USE statement declarative executes.

If neither an INVALID KEY phrase nor a USE AFTER ERROR PROCEDURE statement declarative is specified, and if no FILE STATUS data item is associated with the file, the program aborts when an invalid key condition occurs.

CYBER Record Manager Error, Status 90

An error detected by CYBER Record Manager, the system routines that execute input-output statements, is not reported in detail in the FILE STATUS data item. To learn the specific error, the following statement must be executed:

```
ENTER "C.IOST" USING file-name, data-name-1,
data-name-2.
```

File-name is the name of the file in which the error was detected. Data-name-1, which must be a four-digit integer described as USAGE IS COMP-1 in its Data Description entry, receives one of the detailed error codes listed in the CYBER Record Manager reference manuals. COBOL returns the decimal equivalent of the error code (an octal value) listed in the CYBER Record Manager reference manuals; for example, for the CYBER Record Manager error code 142 (octal), COBOL returns the code 142 (decimal). Therefore, the user can use without translation the code returned in data-name-1. Data-name-2, which must be a one-character alphanumeric data item, receives a letter to indicate the severity of the error as classified by COBOL. The letters returned to data-name-2 can be F or T. The following items indicate the significance of the classification:

F	Fatal error. Usually, this code is returned only when the C.IOST routine is called in a USE AFTER STANDARD ERROR declarative procedure; execution of the program is aborted after execution exits from the declarative. To help prevent the abort of program execution, the ENTER "C.IOENA" statement can be executed before exiting the USE AFTER STANDARD ERROR declarative procedure; however, execution of the C.IOENA routine cannot always prevent the abort of execution. If the program error limit as indicated by the ERL field of the file information table (FIT) is exceeded, CYBER Record Manager overrides the routine and aborts program execution.
T	Trivial error. Further input-output operations are possible for the file unless CYBER Record Manager aborts program execution because the trivial error limit as indicated by the ERL field of the FIT is exceeded. Even if program execution continues, trivial errors might have serious impact on file integrity or results needed, and, in general, should not be ignored. COBOL treats CYBER Record Manager error codes 040, 052, 060, 135, 137, and 176 as trivial errors. The parity error codes, 135 and 137, which a user may want to be treated as trivial, are usually fatal errors. These parity error codes are treated as trivial errors only if the error option (EO) field of the FIT is set (via a FILE control statement or USE clause) to a value other than the default value. (For further information about the EO field see the CYBER Record Manager Basic Access Methods reference manual.)

COBOL Detected Error, Status 99

COBOL detected an error, such as open input on a nonexisting file. An appropriate error message has been output to the dayfile. To learn the specific error, the following statement must be executed:

```
ENTER "C.IOST" USING file-name, data-name-1,
data-name-2.
```

Execution of the above statement returns an error code that is dependent on the dayfile messages as listed in table 3-4. Refer to CYBER Record Manager Error, Status 90 in this section for a description of file-name, data-name-1, and data-name-2.

TABLE 3-4. COBOL-DETECTED ERROR CODES

Error Codes	Execution Diagnostics
2001	RECORDING MODE BINARY ON ADVANCING FILE
2002	RECORD TYPE NOT PRINTABLE ON AN ADVANCING FILE
2003	ATTEMPT TO OPEN FILE WITH SAME NAME AS AN OPEN FILE
2004	FILE OPENED INPUT OR I-O DOES NOT EXIST
2005	ATTEMPT TO OPEN A LOCKED FILE
2006	HOME BLOCK COUNT NOT SPECIFIED FOR A NEW DA FILE
2007	RECORD LENGTH ON OLD RELATIVE FILE DIFF THAN PROGRAM
2010	RELATIVE FILE HEADER NOT VALID
2011	OLD RELATIVE FILE EMPTY - NO HEADER
2012	DELETE OR REWRITE ON SEQ ACCESS FILE WITH NO VALID READ PRECEDING
2013	READ ... NEXT WITH UNDEFINED CURRENT REC POINTER
2014	REWRITE RL DOES NOT EQUAL OLD RL
2015	ILLEGAL RECORD TYPE FOR REWRITE
2016	MULTI-FILE FILE NOT ASSIGNED TO TAPE
2017	READ ON FILE OPENED FOR OUTPUT
2020	ATTEMPT TO OPEN FILE OPENED BY ACCEPT OR DISPLAY AND PROCESSING DIRECTION DIFFERS
2021	DELETE FILE ON OPEN FILE - IGNORED
2022	AT END WHILE TRYING TO ACCEPT FROM FILE
2023	RELATIVE FILE CREATED WITH PRUF=YES BUT REOPENED AS NON PRUF
2024	MRL OR FL SET IN FILE STATEMENT IS LARGER THAN RECORD AREA-TRUNCATED TO RECORD AREA SIZE

If the declarative USE AFTER STANDARD ERROR is selected for the file in error, executing an ENTER "C.IOENA" statement prevents the abort (resulting from the current error) that would normally occur after the declarative is executed. To prevent aborts resulting from subsequent errors in the same file, the ENTER "C.IOENA" statement must be reexecuted. The appropriate error message continues to appear in the dayfile when the ENTER "C.IOENA" statement is used.

ORGANIZATION Clause

The ORGANIZATION clause (figure 3-24) specifies the structure of a file. It is required for any organization other than sequential. When the clause is omitted, sequential organization is assumed.

```
                         ⎧ SEQUENTIAL   ⎫
                         ⎪ RELATIVE     ⎪
   ORGANIZATION IS       ⎨ INDEXED      ⎬
                         ⎪ DIRECT       ⎪
                         ⎪ ACTUAL-KEY   ⎪
                         ⎩ WORD-ADDRESS ⎭
```

Figure 3-24. ORGANIZATION Clause Format

RECORD KEY Clause

The RECORD KEY clause (figure 3-25) specifies an item that defines the primary key for records in indexed, direct, or actual-key files. The clause is required in any program that accesses a file with these organizations.

```
RECORD KEY IS data-name
```

Figure 3-25. RECORD KEY Clause Format

During execution, the data item referenced by data-name specifies the value of the primary key. Values assigned must be unique among records of a file.

Data-name must specify a fixed length item within the Record Description entry associated with the file or an item defined in the Working-Storage Section. The relative position of the item within the record and the item description cannot change for the life of the file. The description of data-name is affected by the file organization.

For indexed files, data-name can be a category alphanumeric or numeric elementary or group item. If data-name specifies an elementary numeric item, its picture must not contain an S unless the item is also described by USAGE IS COMP-1. A group item described with the DEPENDING ON phrase of an OCCURS clause cannot be specified. An alphanumeric item cannot exceed 255 characters.

For actual-key files, data-name must be defined as an elementary COMP-1 or COMP-4 integer of one through eight digits.

For direct files, data-name can be a category alphanumeric elementary or group item, as long as the group item is not described with the DEPENDING ON phrase of the OCCURS clause. Data-name can also be a category numeric unsigned elementary item. If the item is described as COMP-1, it can be signed only if the sign is positive. An alphanumeric item cannot exceed 255 characters.

RELATIVE KEY Clause

The RELATIVE KEY clause (figure 3-26) specifies the key item for a file with relative organization. During execution, the key item specifies a relative record number within the file. The ordinal of the first record in the file is 1. The clause must be specified for relative files that are open in the random or dynamic access mode or that are referenced by a START statement.

```
RELATIVE KEY IS data-name
```

Figure 3-26. RELATIVE KEY Clause Format

When the clause is specified, it must immediately follow an ACCESS MODE clause.

The data item referenced by data-name must be an unsigned positive integer. It cannot be defined in the Record Description entry associated with the file. The data item must not be described by, nor be subordinate to, an OCCURS clause. If the file is an External file, data-name must be defined in the Common-Storage Section.

When the access mode is sequential, the RELATIVE KEY clause is optional. If the clause is omitted, the system reads or writes the records in order by ascending ordinal value. If the clause is specified for sequential access, the value of the relative key for the record just read or written is returned to data-name by the system.

RESERVE Clause

The RESERVE clause (figure 3-27) affects the size of the central memory buffer used during internal processing of sequential, relative, and word-address files. The clause is ignored for files with other organizations. If the clause is omitted, buffer allocation is under system control.

```
                    [AREA ]
RESERVE integer     [AREAS]
```

Figure 3-27. RESERVE Clause Format

In the clause, integer must be an unsigned integer in the range 1 through 4095. Specification of a large value might result in an abort during execution if the required memory is not available.

The system uses the specified integer to calculate buffer size.

Buffer size is affected by the file organization and the block size as described in the following list. (For more information see BLOCK CONTAINS clause in the FD entry of the Data Division.)

Sequential File

Five times maximum block size, plus two words if clause omitted; otherwise, integer times maximum block size, plus two words.

Relative File

Two times 64 words if clause omitted; otherwise integer times 64 words.

Word-Address File

Two times maximum block size, plus two words if clause omitted; otherwise, integer times maximum block size, plus two words.

Indexed, Direct, Actual-Key File

Buffer size allocated by system.

If any file is accessed in a very random manner, use of this clause might result in system performance degradation.

SELECT Clause

The SELECT clause (figure 3-28) identifies the file associated with the File-Control entry in which the clause appears. SELECT must be the first clause in the entry.

```
SELECT [OPTIONAL] file-name
```

Figure 3-28. SELECT Clause Format

In the clause, file-name specifies a file named in an FD entry or an SD entry; only one SELECT clause can reference a given file. If file-name specifies a file in an SD entry only the ASSIGN clause can follow the SELECT clause.

The OPTIONAL phrase can only be specified for input files with sequential organization. It is required for input files that are not necessarily present each time the program executes.

USE Clause

The USE clause (figure 3-29) specifies file information that cannot otherwise be specified through COBOL language statements. It can also be used to override processing specified by Procedure Division statements and statements in the Input-Output Section and the File Section of the program.

```
USE literal
```

Figure 3-29. USE Clause Format

The clause must contain a nonnumeric literal that duplicates the parameter format of the FILE control statement processed by CYBER Record Manager. Parameter format is:

keyword=value [, keyword=value] . . .

Keywords correspond to the symbolic names of fields in the file information table (FIT) used to govern CYBER Record Manager processing. Values can be symbolic or integer.

Parameters that can be specified through the USE clause are:

BBH	EFC	IBL	PC
BCK	EO	IP	RMK
BFS	ERL	MUL	RT
BT	FLM	NL	SBF
DFC	FWI	ORG	SPR
DP	HB	OVF	TRC

A complete description of the meaning and use of these parameters appears in the CYBER Record Manager documentation.

In addition to the CYBER Record Manager parameters, the following can be specified:

PRINTF=YES

Identifies file as a print file. File structure is set to RT=Z and BT=C, and the record area is adjusted such that the first character supplied by the program is a data character, not a carriage control character. If the BEFORE/AFTER ADVANCING phrase of WRITE is not used, all lines are single spaced. If the phrase of WRITE is used, the phrase establishes the line spacing. PRINTF=YES is assumed on Report Writer files, and files with a LINAGE clause in the FD entry.

PRUF=YES

Applicable to relative files only. PRUF=YES causes each record to begin on a PRU boundary and to be written and read as a full PRU. (A PRU is 640 characters and is the smallest unit transferred to or from mass storage for each read or write issued by the system.) When file records are exactly 10 characters less than a PRU multiple, very efficient processing results. (COBOL uses 10 characters in the PRU at the beginning of each record.) When this parameter is specified, the record area is used as a buffer; any RESERVE clause is ignored.

Examples of the USE clause are:

USE "IP=25"

Causes 25 percent padding in index blocks for an indexed file.

USE "RT=R,RMK=62B"

Specifies that record type for the file is R and that each record is terminated by the character]. Overrides the record type established by the FD entry for the file.

NOTE

Refer to appendix F for recommendations on the use of record types.

WORD-ADDRESS KEY Clause

The WORD-ADDRESS KEY clause (figure 3-30) specifies the beginning position of a record within a file with word-address organization. The clause is required to access a file with this organization.

```
WORD-ADDRESS KEY IS data-name
```

Figure 3-30. WORD-ADDRESS KEY Clause Format

The data item specified by data-name must be defined as an unsigned numeric integer. It cannot be described by, or be subordinate to, an OCCURS clause. If the file is an External file, data-name must be defined in the Common-Storage Section.

During execution, the program is responsible for setting the item to the number of the word in the file at which reading or writing is to occur. See the Basic Access Methods (BAM) reference manual for information pertinent to different record types.

After a read or write operation, the key is updated to the next available word following the record just written or read. The length of the record depends on the record descriptions and the RECORD clause.

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph is used to specify the points at which rerun is to be established, the memory area that is to be shared by different files, and the location of files on a multifile reel. The paragraph is optional and the clauses can appear in any order. The skeleton for the I-O-CONTROL paragraph is shown in figure 3-31. The paragraph header is:

I-O-CONTROL.

```
I-O-CONTROL.  [APPLY clause]
[; MULTIPLE FILE TAPE clause] . . .
[; RERUN clause] . . .    [; SAME clause] . . . .
```

Figure 3-31. I-O-CONTROL Paragraph Skeleton

APPLY Clause

The APPLY clause is documentary only, although the clause syntax is checked. Input-output-technique can be any word that conforms to the rules for forming a procedure-name. Clause format is:

```
APPLY input-output-technique ON file-name-1
    [, file-name-2 ] . . .
```

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause (figure 3-32) specifies that a file named in the program shares the same physical tape reel or set of reels with other files; it also indicates the position of the specified file within the multifile tape. The clause is required when a file that resides on a multifile tape is referenced by a program, unless the file position is specified by control statement parameters.

The tape files must have labels described with LABEL RECORDS ARE STANDARD in their FD entries.

More than one MULTIPLE FILE TAPE clause can appear to describe each file involved. The position of the file is specified by the position of the file-name or pseudo-file-name within the sequence of file-names and/or pseudo-file-names or by an associated POSITION phrase.

File-name-1 and file-name-2 name files that are referenced in an OPEN statement and have an FD entry associated with the file; pseudo-file-name-1 and pseudo-file-name-2 name files on a multifile tape that are not referenced within the program. A combination of file-names and pseudo-file-names can be specified to indicate the ordinal position of the files referenced by the program.

The multifile name (MFN) is the name used to assign a multiple file tape. The MFN depends on several factors and differs for each operating system, as described in the following paragraphs.

On the NOS operating system, the first six characters of the implementor-name (as specified in the ASSIGN clause) are used as the MFN. The implementor-name can be from one to seven characters in length. The seventh character can differ for each file in a set. If FILE-SET-ID is specified in the FD entry for the file, it is used to check the multifile set name field in the label. If FILE-SET-ID is not specified, MFN is used to check the multifile set name field in the label.

On the NOS/BE operating system, if FILE-SET-ID is specified in the FD entry for the file, it is used as the MFN. If FILE-SET-ID is not specified, the first six characters of the implementor-name are used. In this case, implementor-name must be seven characters in length, and the last character should differ for each file. The REQUEST or LABEL control statement, specifying MFN, must contain the MF parameter and must specify U labels. The multifile set name field in the label is not checked on input, but is set to MFN on creation.

Regardless of the operating system, the MFN for file-name-1 and file-name-2 must be the same and must conform to the rules for file names in the operating system being used.

The POSITION phrase of the MULTIPLE FILE TAPE clause specifies the ordinal position of the file on the multifile tape. It must be used if all files on a multifile tape are not specified by a file-name or a pseudo-file-name, or if the file-names or pseudo-file-names are not specified in consecutive ordinal order corresponding to their physical placement on the tape. The first file on the tape has position 1. Integers must be unsigned and positive with a value greater than zero.

RERUN Clause

The RERUN clause (figure 3-33) specifies the conditions under which the system is to checkpoint the program. When a checkpoint occurs, the current state of the program is copied to a checkpoint tape along with information about files attached to the job. In the event the program terminates abnormally, the RESTART capability of the operating system can be used with the checkpoint tape to continue the program from the situation existing at the last checkpoint.

Any change made to a permanent random access file since the last checkpoint dump will remain in the file, and attempts to update the changed records can result in errors. Therefore, it is recommended that the RERUN clause not be used when updating permanent random access files.

More than one RERUN clause can be specified as long as no RERUN clause duplicates an integer-1 RECORDS condition specified for a particular file-name.

ON Phrase

The ON phrase is documentary only, but should be specified. When the job requests a tape and designates it as a checkpoint tape, that tape holds the checkpoint dumps. Otherwise, a tape with the name CCCCCCC is requested by the system and used for the checkpoint.

EVERY Phrase

The EVERY phrase specifies the conditions under which a checkpoint is to occur.

```
                         { file-name-1        }
MULTIPLE FILE TAPE CONTAINS { pseudo-file-name-1 } [POSITION integer-1]

   [, { file-name-2        } [POSITION integer-2]] . . .
      { pseudo-file-name-2 }
```

Figure 3-32. MULTIPLE FILE TAPE Clause Format

```
                                                  {  {            { REEL } }                  }
RERUN [ON { file-name-1         }]  EVERY  {  { [END OF] { UNIT } }  OF file-name-2  }
          { implementor-name    }           {  { integer-1 RECORDS   }                  }
                                                  {  condition-name                             }
```

Figure 3-33. RERUN Clause Format

END OF REEL/UNIT

Specifies that checkpoint is to occur each time an end-of-reel or unit condition is encountered on file-name-2. (The terms REEL and UNIT are synonymous and interchangeable.) For tape files, the end-of-reel condition can occur either as the result of executing a CLOSE statement with a REEL/UNIT phrase or as a result of reaching the physical end of a tape reel. For mass storage sequential files, the condition only occurs as the result of executing a CLOSE statement with a REEL/UNIT phrase. File-name-2 cannot reference a file described with an SD entry or with the EXTERNAL clause in the FD entry; file-name-2 cannot be referenced in more than one RERUN clause.

Integer-1 RECORDS

Specifies that checkpoint is to occur at the end of processing each integer-1 records. Integer-1 must be positive and unsigned with a value greater than zero but not exceeding 2^{15}-1, or 32767.

Condition-name

Specifies that checkpoint is to occur when a particular switch status exists at the time the program resumes execution after halting from STOP literal statement execution. Condition-name must be defined in the SWITCH-n clause of the SPECIAL-NAMES paragraph and be associated with a mnemonic-name and an ON or OFF status. An external switch can be manipulated by a SWITCH control statement between job steps, by a central site operator at any time, or by a terminal operator following a STOP literal statement. An internal or external switch can be manipulated in the program by execution of a format 4 SET statement. The system interrogates the switch as soon as operation resumes after the STOP statement. A checkpoint occurs when the switch condition corresponds to that defined in the RERUN clause.

SAME AREA Clause

The SAME AREA clause (figure 3-34) can be specified by four phrases. The first three phrases (SAME AREA, SAME SORT AREA, SAME SORT-MERGE AREA) indicate that the named files share the same buffer. The fourth phrase (SAME RECORD AREA) indicates that the named files share the same record area within the program.

Files referenced in the SAME AREA clause cannot be described with the EXTERNAL clause in the FD entry.

Buffer Sharing

The SAME AREA, SAME SORT AREA, and SAME SORT-MERGE AREA clauses are equivalent and are used to specify that different files can use the same internal buffer area. Only one file named in one of these clauses can be open at any given time. The buffer area is allocated by the system when the file is opened and is released when the file is closed. The released area is available as buffer space for other files or for sorting or merging operations whether or not the SAME AREA clauses are present.

A file-name defined by an SD entry can appear only in a SAME SORT AREA or a SAME SORT-MERGE AREA clause. No file-name can appear more than once in each type of phrase of the SAME AREA clause. If a file-name appears in two or more types of SAME AREA clauses, all of the file-names in those clauses must be duplicated in each clause.

Record Area Sharing

The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing the current record. All of the files can be in the open mode at the same time, and a record in the SAME RECORD AREA is considered as a record of each opened input and output file whose file-name appears in a SAME RECORD AREA clause.

A file-name must not appear in more than one SAME RECORD AREA clause. However, a file-name can appear in both a SAME RECORD AREA clause and one or more of the SAME AREA clauses. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in the SAME RECORD AREA clause. Additional file-names not appearing in the SAME AREA clause can also appear in the SAME RECORD AREA clause.

Although all files mentioned in a SAME RECORD AREA clause can be opened at any given time, when the files are also mentioned in a SAME AREA clause the rule that only one of the files mentioned in a SAME AREA clause can be opened at one time takes precedence over the rule for files named in a SAME RECORD AREA clause.

```
       [( RECORD     )]
  SAME [{ SORT       }]  AREA FOR file-name-1  {, file-name-2} . . .
       [( SORT-MERGE )]
```

Figure 3-34. SAME AREA Clause Format

DATA DIVISION

The Data Division describes all data referenced in the program, whether it is data read or written to files, data that is developed during program execution, or data that is assigned a value which remains constant during program execution.

The Data Division is made up of seven optional sections. Within each section, different types of entries, which begin with a level indicator or level number, describe the physical representation and logical use of the data.

A skeleton of the Data Division is shown in figure 4-1.

The division header is:

DATA DIVISION.

The header must appear on a separate line, beginning in area A. The separator period must terminate the header.

DATA DIVISION SECTIONS

The six sections of the Data Division each describe a different use of data:

File Section
: Describes data in input or output files.

Common-Storage Section
: Describes data stored in common memory areas for use by independently compiled subprograms.

Working-Storage Section
: Describes constants or data developed within a program.

Secondary-Storage Section
: Describes data stored in extended memory.

Linkage Section
: Describes data passed to a subprogram by a CALL statement.

Communication Section
: Describes data passed to and from MCS. The section can be used only under NOS.

Report Section
: Describes content and format of report to be generated by the Report Writer facility.

```
DATA DIVISION.
[FILE SECTION.
    [{file-description-entry           }  [record-description-entry] ... ] ... ]
    [{sort-merge-file-description-entry}
[COMMON-STORAGE SECTION.
    [77-level-description-entry]
    [record-description-entry  ]  ... ]
[WORKING-STORAGE SECTION.
    [77-level-description-entry]
    [record-description-entry  ]  ... ]
[SECONDARY-STORAGE SECTION.
    [record-description-entry]  ... ]
[LINKAGE SECTION.
    [77-level-description-entry]
    [record-description-entry  ]  ... ]
[COMMUNICATION SECTION.
    [communication-description-entry [record-description-entry]. . .]. . .]
[REPORT SECTION.
    [report-description-entry      {report-group-description-entry} ... ] ... ]
```

Figure 4-1. Data Division Skeleton

All sections are optional. When the section is omitted, the section header can, but need not, be omitted. When included, sections must be in the order shown in figure 4-1. This figure also shows the different types of entries that can appear in each section. Figure 4-2 further defines terminology of entries.

FILE SECTION

The File Section defines the structure of data files. Each file used for input, output, or sort or merge operations must be defined in this section. Files referenced in the Procedure Division only by the ACCEPT or DISPLAY statements are not defined in the File Section, with the exception of the file with the implementor-name "OUTPUT" which can, but need not, be defined.

The File Section is composed of a section header followed by a number of File Description entries or Sort Description entries. The section header is:

<u>FILE SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

File Description entries, which begin with an FD level indicator, define input, output, and report files. For input and output files, Record Description entries follow the FD entry to describe the data. For report files, however, the data is described in Report Description entries within the Report Section, as discussed in section 6 of this manual.

Sort-Merge-File-Description entries, which begin with an SD level indicator, define the record and the sort keys, as discussed in section 7 of this manual. Record Description entries follow the SD entry.

When the File Section is omitted, no files can be referenced in the program, except through ACCEPT and DISPLAY statements that assume default file and data descriptions. See the discussion of these statements in section 5 of this manual for details.

COMMON-STORAGE SECTION

The Common-Storage Section describes data that is shared by independently compiled subprograms. See section 15, Inter-Program Communication Facility.

WORKING-STORAGE SECTION

The Working-Storage Section describes data that is not part of a record in a file. Data items defined in this section can be assigned values in this section or can be assigned values developed during program execution.

The Working-Storage Section is composed of the section header followed by any number of level 77, level 66, or level 88 Data Description entries and/or Record Description entries. The section header is:

<u>WORKING-STORAGE SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

The VALUE clause can be used to set the initial value of an item in the Working-Storage Section. If an initial value is not specified for a data item, its initial content is undefined.

SECONDARY-STORAGE SECTION

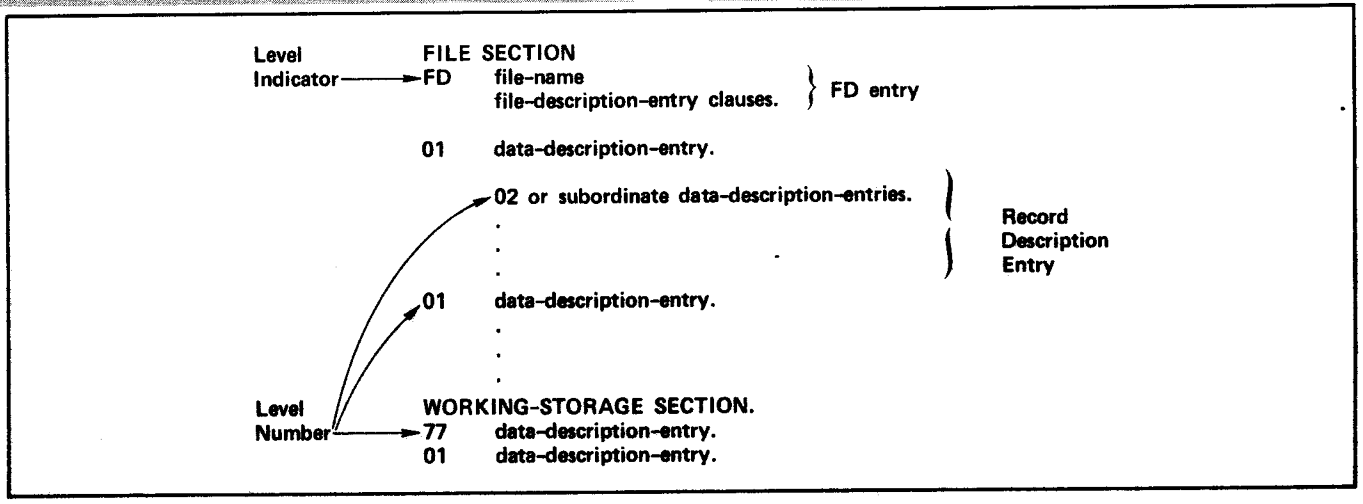
The Secondary-Storage Section describes group items that are allocated to extended memory as an alternative to central memory. Only nonedited alphanumeric moves (including the implicit moves associated with the INTO and FROM options of the READ, WRITE, RELEASE, and RETURN statements) can be performed on data defined in the Secondary-Storage Section. References to Secondary-Storage items can be subscripted, but Secondary-Storage items cannot be used as subscripts.

The Secondary-Storage Section is composed of a section header followed by any number of Record Description entries. The section header is:

<u>SECONDARY-STORAGE SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

Only group items with level numbers 01 through 49 can be specified in this section; otherwise, Secondary-Storage items are like Working-Storage items. Items in Secondary-Storage cannot be described with an OCCURS clause having a DEPENDING ON phrase.



Level Indicator
FILE SECTION
FD file-name
file-description-entry clauses.
FD entry
01 data-description-entry.
02 or subordinate data-description-entries.
Record Description Entry
01 data-description-entry.
Level Number
WORKING-STORAGE SECTION.
77 data-description-entry.
01 data-description-entry.

Figure 4-2. Data Division Terminology Summary

Usage of Secondary-Storage requires the use of the EC parameter on the control statement. For interactive mode, NOS and NOS/BE require RFL,EC=n. For batch mode, NOS requires ECn on the job control statement and RFL,EC=n prior to execution; NOS/BE requires ECn on the job control statement.

LINKAGE SECTION

The Linkage Section is valid only in a called subprogram. See section 15, Inter-Program Communication Facility.

COMMUNICATION SECTION

The Communication Section is required in programs that use the Message Control System (MCS). The Communication Section can be used only under the NOS operating system. The Communication Section defines the two Communication Description (CD) areas that communicate with MCS. These CD areas are referenced by the six statements (ACCEPT MESSAGE COUNT, SEND, RECEIVE, DISABLE, ENABLE, PURGE) in the Procedure Division that interact with MCS under the NOS operating system.

The Communication Section is composed of the section header and two CD areas, one CD area for input-type functions, and one CD area for output-type functions. The section header is:

COMMUNICATION SECTION.

The header must appear on a separate line, beginning in area A, and it must be terminated by a separator period.

REPORT SECTION

The Report Section describes reports that are generated through the Report Writer. See section 6, Report Writer Facility.

DATA DIVISION ENTRIES

Entries within the Data Division begin with a level indicator or a level number.

A level indicator consists of two characters: FD, SD, RD, or CD.

FD and SD entries must appear only in the File Section.

RD entries must appear only in the Report Section.

CD entries must appear only in the Communication Section and only under NOS.

The level indicators FD or SD must be immediately followed by a file-name specified in a SELECT clause of the Environment Division. The level indicator CD must be immediately followed by a cd-name defined in the Communication Section of the Data Division. The CD level indicator can be used only under the NOS operating system. The level indicator RD must be immediately followed by a report-name specified in the Report Section of the Data Division. The level indicator must begin in area A of a line; the file-name, report-name, or cd-name should begin in area B, but can begin in area A. Clauses within the entry define general characteristics and data items in each group. The entry is terminated by the separator period.

Entries that begin with a level indicator are followed by Data Description entries that begin with a level number. Entries of level 01 through 49 describe the hierarchy of items: that is, they describe records, group items, and elementary items. Entries of level 66, 77, and 88 have special purposes and do not define the hierarchy of the items described.

The remainder of this section describes File Description entries, Record Description entries, Data Description entries, and Communication Description entries. Data Division clauses that can appear only in RD entries and SD entries are discussed in sections 6 and 7, respectively.

FILE DESCRIPTION ENTRY

A File Description entry defines the physical structure and either the record names or report names pertaining to a given file. It also specifies the manner in which data is recorded on the file, the size of the logical and physical records, and the labels on the file. A skeleton of a File Description entry is shown in figure 4-3.

```
FD file-name
    [; BLOCK CONTAINS clause]
    [; CODE-SET clause]
    [; DATA RECORDS clause]
    [; EXTERNAL clause]
     ; LABEL RECORDS clause
    [; LINAGE clause]
    [; RECORD clause]
    [; RECORDING MODE clause]
    [; REPORT clause ]
```

Figure 4-3. FD Entry Skeleton

The level indicator FD must be immediately followed by a file-name that has been specified in a SELECT clause and equated to an implementor-name in the FILE-CONTROL paragraph of the Environment Division. The clauses in the FD entry can be in any order; only the LABEL RECORDS clause is required. Punctuation between clauses is optional.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause (figure 4-4) specifies information the system uses to determine the physical size of a block in the file. The clause is valid only for files with sequential, indexed, direct, and actual-key organizations; it is ignored for files with relative and word-address organization.

```
BLOCK CONTAINS  [integer-1 TO]

                integer-2 [ { RECORDS    } ]
                          [ { CHARACTERS } ]
```

Figure 4-4. BLOCK CONTAINS Clause Format

Integer-1 and integer-2 are unsigned numeric literals indicating the number of records or characters to be used in system calculations. Integer-2 must be greater than integer-1. If both RECORDS and CHARACTERS are omitted, CHARACTERS is assumed.

The physical block size that results from this clause or the omission of this clause depends on the file organization; and, in the case of sequential files, depends also on the device on which the file resides. The number of records that exist in each block is affected by actual record size (see the RECORD clause), by sequential file block type, and by system use of space within the block. Indexed file data blocks, for example, contain key information as well as user records.

(See the CYBER Record Manager manuals for information about choosing an appropriate block structure, about other parameters that can affect block structure, and about alternative means of specifying file structure.)

Indexed Files

Indexed files have index blocks and data blocks. The BLOCK CONTAINS clause affects only the size of data blocks. For indexed files in the extended indexed sequential format, the size of a data block is always a multiple of PRU size (640 characters) less 50 characters. For indexed files in the initial indexed sequential format, the size of a data block is always a multiple of PRU size (640 characters) less 10 characters. The value, in characters, that the system rounds upward for block size is:

Maximum record size if the clause is omitted.

Maximum record size multiplied by integer-2 when RECORDS is specified.

Integer-2 when CHARACTERS is specified.

For indexed files in initial indexed sequential format, the AAM utility ESTMATE should be used to select an efficient data block size for a file with a given average record size, key type, and key length. For indexed files in extended indexed sequential format, the AAM utility FLBLOK should be used to choose the block size.

Direct Files

For direct files in the extended direct format, the size of a block is always a multiple of PRU size (640 characters) less 50 characters. For direct files in the initial direct format, the size of a block is always a multiple of PRU size (640 characters) less 10 characters. The value, in characters, that the system rounds upward for block size is:

Two average size records when the clause is omitted.

Maximum record size multiplied by integer-2 when RECORDS is specified.

Integer-2 when CHARACTERS is specified.

Actual-Key Files

For actual-key files in the extended actual-key format, the size of a block is always a multiple of PRU size (640 characters) less 50 characters. For actual-key files in the initial actual-key format, the size of a block is always a multiple of PRU size (640 characters) less 10 characters. The value, in characters, that the system rounds upward for block size is:

Maximum record size multiplied by eight, plus 90 characters, when the clause is omitted.

Sum of (maximum record size multiplied by integer-2) and ((integer-2 plus 1) multiplied by 10) when RECORDS is specified.

Integer-2 when CHARACTERS is specified.

Sequential Files

A block in a sequential file has a type and a size that depends on the device.

When the file is on a mass storage device, the block is always a C type block of 640 characters.

When the file is on a tape in SI or I format, the block is always a C type block of PRU size. PRU size for coded tape is 1280 characters; for binary tape PRU size is 5120 characters.

When the file is on a tape with S or L format, block type and size vary according to the clause specification:

Omitted

Block type K with one record per block.

Integer-2 RECORDS

Block type K with the size of each actual block varying as the sizes of the particular integer-2 records vary.

Integer-1 TO integer-2 RECORDS

Block type E with minimum block size of integer-1 multiplied by minimum record size, and with maximum block size of integer-2 multiplied by maximum record size.

Integer-2 CHARACTERS

Block type E with integer-2 characters per block.

Integer-1 TO integer-2 CHARACTERS

Block type E with minimum block size of integer-1 characters and maximum block size of integer-2 characters.

K type blocks and E type blocks always have an even number of characters, so the system adds a padding character to the block if necessary. K type blocks have the same number of records in each block; if the size of the records in the block varies, the block size varies accordingly. E type blocks have as many records as can be accommodated between the minimum and maximum block sizes. W type records cannot be used with K or E type blocks.

CODE-SET Clause

The CODE-SET clause (figure 4-5) is used for internal code conversion between an external alphabet and the display code used internally by the system.

```
CODE-SET IS alphabet-name
```

Figure 4-5. CODE-SET Clause Format

The alphabet-name specified must be associated with an external alphabet in the ALPHABET clause of the SPECIAL-NAMES paragraph of the Environment Division.

The CODE-SET clause provides a way of assisting the user in reading and/or writing information for processing on another manufacturer's system. Information is usually transferred via magnetic tape or punched cards.

The CODE-SET clause cannot be specified if the EXTERNAL clause is included in the FD entry.

Tape Processing

The CODE-SET clause can be used for 7-track tapes with odd parity written on a UNIVAC 1100 series system. Tapes are read or written in UNIVAC 1100 Series FIELDATA code if the alphabet-name in the CODE-SET clause is associated with the external alphabet UNI in the SPECIAL-NAMES paragraph. Specifying the CODE-SET clause causes automatic translation between FIELDATA code and the internal display code of the system.

The operating system provides normal read/write processing for 9-track tapes in ASCII or EBCDIC code and 7-track tapes in external BCD code.

Card Processing

Card decks consisting of characters from the 64-character ASCII or EBCDIC subset can be processed as input, but cannot be punched out unless the operating system is installed in ASCII mode. The CODE-SET clause must be used to punch a deck using the full 64-character subsets of ASCII and EBCDIC. The clause must also be used to read a deck coded in one of these subsets plus the lower case alphabet.

The alphabet-name must be associated with the external alphabet ASCII-64, STANDARD-1, or EBCDIC, depending on the code translation desired.

If the CODE-SET clause is specified, the object time input-output routines translate code in free-form binary format to the internal display code used by the system. (A card in free-form binary format contains 12 bits-per-column and a maximum of 16 words-per-card.)

To create an input file in free-form binary format from an ASCII or EBCDIC deck under the NOS operating system, the deck must be preceded and followed by cards with a 5/7/9 punch in column 1 and a 4/5/6/7/8/9 punch in column 2. The deck must be in a record by itself.

To create an input file in free-form binary format from an ASCII or EBCDIC deck under the NOS/BE operating system, the deck must be preceded and followed by cards with punches in all 12 rows of both column 1 and column 2. The deck must be in a record by itself.

A file created using this procedure is punched by specifying a disposition code of P8 on a ROUTE or DISPOSE control statement. For more detailed information, consult the appropriate operating system reference manual.

If binary information is to reside on either the file INPUT or the file PUNCH, a FILE control statement must be used to override the default block type and record type for these files. The FILE statement must specify F type records and C type blocks.

DATA RECORDS Clause

The DATA RECORDS clause (figure 4-6) associates the names of data records with their files. It is documentary only, but should be specified.

```
DATA { RECORD IS   } data-name-1
     { RECORDS ARE }

          [, data-name-2] . . .
```

Figure 4-6. DATA RECORDS Clause Format

The data-names must specify level 01 Record Description entries that are subordinate to the FD entry containing the DATA RECORDS clause.

EXTERNAL Clause

The EXTERNAL clause specifies that the file is an External file and can be shared by several programs in a run unit. All External files must be described in the main program; a subprogram describes only those External files referenced by that subprogram. Clause format is:

EXTERNAL

When the EXTERNAL clause is specified, all other clauses associated with the file must be the same in any program referencing the file. It is best to use the COPY statement or Update common decks to ensure that all descriptions of the file are the same.

Certain features are prohibited with External files. The Report Writer facility cannot be used. The LABEL RECORDS clause cannot specify STANDARD. The RERUN clause and the SAME AREA clause cannot be specified for an External file. The CODE-SET clause cannot be specified.

In the FD entry and the File-Control entry for an External file, data-names specified in the clauses must be defined in the Common-Storage Section of the Data Division. That is, data-names specified in clauses such as the RECORD KEY clause, the BLOCK COUNT clause, the FILE STATUS clause, and the RECORD KEY clause must be defined in the Common-Storage Section. However, the data-names described within the Record Description entries of levels 01 through 49 must not be defined in the Common-Storage Section and must follow the FD entry.

LABEL RECORDS Clause

The LABEL RECORDS clause (figure 4-7) specifies whether or not standard labels exist on the file. It can also specify values for standard label fields. The clause is required in every FD entry.

OMITTED

Specifies that no explicit labels exist for the file. Must be specified for all files that reside on mass storage or are External files.

STANDARD

Specifies that tape file labels conform to the standard system format, which is that defined in American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange. STANDARD is valid only for magnetic tape files and files that are not External files. If STANDARD is specified for a mass storage file, the labels are ignored. It must be specified when the MULTIPLE FILE TAPE clause is used to position a multiple file tape.

The VALUE OF phrase is valid only for tape files with labels specified as STANDARD. It specifies the value of label record items to be written during output file processing or to be checked during input file processing. Label processing occurs at the time the file is opened during execution.

In the phrase, each implementor-name specifies a label record item. These names correspond to fields in standard HDR1 labels with similar names. Table 4-1 defines the implementor-names that can be specified and the meaning of the fields. The contents of the items should conform to the length and type of character the standard defines; larger data items are truncated.

Both data-name and literal specify values for the items. Literals for all except implementor-names FILE-ID and FILE-SET-ID must be numeric. A figurative constant can be substituted for the literal. Data-names must be defined in the Working-Storage Section and must be described as USAGE IS DISPLAY; they cannot reference an item described with an OCCURS clause. Data-names can be qualified when necessary, but cannot be subscripted or indexed.

See the operating system reference manual for more information about standard tape labels.

```
LABEL {RECORDS ARE} {STANDARD}
      {RECORD IS  } {OMITTED }

    [; VALUE OF implementor-name-1 IS {data-name-1}
                                      {literal-1  }

       [, implementor-name-2 IS {data-name-2}] . . . ]
                                {literal-2  }
```

Figure 4-7. LABEL RECORDS Clause Format

TABLE 4-1. STANDARD LABEL ITEMS FOR TAPE FILES

Implementor-name	Contents
FILE-ID	1-17 character file identifier.
FILE-SET-ID	1-6 character identifier of multifile set. See discussion of MULTIPLE FILE TAPE clause for usage of this field.
FILE-SECTION-NUMBER†	1-4 digits indicating number of this reel in a set. Normally starts with 1 and is incremented by the system for each new reel.
FILE-SEQUENCE-NUMBER	1-4 digits indicating number of file in a multifile set.
GENERATION-NUMBER†	1-4 digits.
GENERATION-VERSION-NUMBER	1-2 digits indicating edition of file.
CREATION-DATE	5 digits: 2 digits for year followed by 3 digits for day of year.
EXPIRATION-DATE	5 digits: 2 digits for year followed by 3 digits for day of year. Indicates the first date the file might be overwritten. 99999 is indefinite retention.
ACCESSIBILITY†	1 character. The tape cannot be read unless the value matches that written in the label. Default is blank.

†Items can be specified, but this label field is not processed by the system. See the operating system reference manual for more information about standard tape labels.

LINAGE Clause

The LINAGE clause (figure 4-8) specifies the number of lines in a logical page of a file to be printed. The values specified by the clause are used during opening and writing of the file to control vertical positioning of output lines. The clause can specify top and bottom margins and a footing area within the page.

The clause cannot appear in an FD entry that has a REPORT clause.

The LINAGE clause defines logical pages; the programmer is responsible for reconciling the physical page size and top-of-form alignment for a particular printer with the values in this clause. The logical page defined by the clause need not have any relation to physical page size.

A separate special register LINAGE-COUNTER is generated for each file whose FD entry contains a LINAGE clause. The special register is an implicit unsigned integer item. The value in the special register for a given file is the current line number within one page body. The value is set to 1 when the file is opened because the first line on a logical page is line 1. If neither LINES AT TOP nor LINES AT BOTTOM is specified, blank lines are written to position the page. If either LINES AT TOP or LINES AT BOTTOM is specified, the page must be positioned by other means. The special register can be referenced within the Procedure Division, but it cannot be directly changed by the program. When more than one file is described with a LINAGE clause, all references to LINAGE-COUNTER must be qualified by file-name.

All data-names specified in the LINAGE clause must reference elementary unsigned numeric integer data items. Integers must be unsigned and positive. If the file is an External file, all data-names must be defined in the Common-Storage Section.

Data-name-1 or integer-1 specifies the number of lines that can be written and/or spaced on a logical page. The value of data-name-1 or integer-1 must be greater than zero.

Footings and margins are specified by the following phrases. The values in these phrases must be logically consistent with the page size. A value of zero can be specified for the margins.

WITH FOOTING AT

Specifies the line number within the page body at which the footing area begins. The value of data-name-2 or integer-2 must be greater than zero and less than the value specified by data-name-1 or integer-1. A footing ends at line data-name-1 or integer-1. Default value is integer-1.

LINES AT TOP

Specifies the number of lines at the top of a logical page. Zero lines can be specified. The default value is zero.

LINES AT BOTTOM

Specifies the number of lines in the bottom margin of a logical page. Zero lines can be specified. The default value is zero.

If either LINES AT TOP or LINES AT BOTTOM are specified in the file, when the file is opened, a blank line with a Q in the carriage control position (to suppress auto-eject) is written, followed by a blank line with a 1 in the carriage control position (to eject to the top of the page). Thereafter, no page eject carriage control characters are generated. Instead, a number of blank lines are written to position the page properly. These lines can contain double, triple, or suppress space printer control characters. If neither clause is specified, the positioning is done with the normal page eject carriage control mechanism (that is, 1 in the first position of a line). Therefore, for normal printer output, neither clause is used. The usual use for these clauses is for terminal output.

RECORD Clause

The RECORD clause (figure 4-9) specifies information the system uses to determine record size (for all file organizations) and record type (for all file organizations, except word-address and relative). This same file organization information is determined by the Record Description entry when the RECORD clause is omitted.

For word-address files the record type is always U (undefined). The program must set the record size correctly before a read operation can be performed. For read operations, the largest record description is used if the program has multiple record descriptions and no DEPENDING ON clauses (either OCCURS or RECORD). Write operations use the size of the record specified in the WRITE statement. For relative files the record type is always F (fixed-length).

Table 4-2 shows the minimum and maximum record sizes that result from clause specifications. Sizes are calculated in 6-bit characters.

NOTE

Refer to appendix F for recommendations on the use of the RECORD clause.

```
LINAGE IS { data-name-1 } LINES [, WITH FOOTING AT { data-name-2 } ]
          { integer-1   }                         { integer-2   }

   [, LINES AT TOP { data-name-3 } ]  [, LINES AT BOTTOM { data-name-4 } ]
                   { integer-3   }                       { integer-4   }
```

Figure 4-8. LINAGE Clause Format

Format 1

```
RECORD CONTAINS [integer-1 TO]
integer-2 CHARACTERS  [DEPENDING ON data-name]
```

Format 2

```
RECORD IS VARYING IN SIZE  [[FROM integer-1]
[TO integer-2] CHARACTERS]  [DEPENDING ON data-name]
```

Figure 4-9. RECORD Clause Format

TABLE 4-2. RECORD SIZE AND TYPE

RECORD Clause	Sequential, Indexed, Direct, Actual-Key				Relative[†]	Word-Address[†]
	CYBER Rec. Mgr. Record Type	Maximum Record Size[††]	Minimum Record Size[††]	Actual Record Size[††]	Actual Record Size	Actual Record Size
Omitted						
All Record Descriptions same size	F	Largest record	Largest record	Largest record	Largest record	Largest record
Record Description not all same size; no OCCURS/ DEPENDING ON	W	Largest record plus 10	Smallest record plus 10	Named record size plus 10	Largest record	Named record size
OCCURS/DEPENDING ON in the Record Description entry	T	Largest record	Largest record	Calculated from data-name	Largest record	Calculated from data-name
OCCURS/DEPENDING ON not in Record Description entry	W	Largest record plus 10	Smallest record plus 10	Calculated from data-name plus 10	Largest record	Calculated from data-name
RECORD CONTAINS						
Integer-2 CHARACTERS	F[†††]	Integer-2 characters	Integer-2 characters	Integer-2 characters	Integer-2 characters	Integer-2 characters
Integer-1 TO integer-2 CHARACTERS	W[§]	Integer-2 characters plus 10	Integer-1 characters plus 10	Named record size plus 10	Integer-2 characters	Named record size
Integer-1 TO integer-2 CHARACTERS DEPENDING ON data-name in Record Description	D	Integer-2 characters	Integer-1 characters	Data-name value	Integer-2 characters	Data-name value
Integer-1 TO integer-2 CHARACTERS DEPENDING ON data-name not in Record Description	W	Integer-2 characters plus 10	Integer-1 characters plus 10	Data-name value plus 10	Integer-2 characters	Calculated from data-name

TABLE 4-2. RECORD SIZE AND TYPE (Contd)

RECORD Clause	Sequential, Indexed, Direct, Actual-Key				Relative[†]	Word-Address[†]
	CYBER Rec. Mgr. Record Type	Maximum Record Size[††]	Minimum Record Size[††]	Actual Record Size[††]	Actual Record Size	Actual Record Size
RECORD VARYING FROM integer-1 CHARACTERS	W	Largest record plus 10	Integer-1 characters plus 10	Named record size plus 10	Largest record	Named record size
TO integer-2 CHARACTERS	W	Integer-2 characters plus 10	Smallest record plus 10	Named record size plus 10	Integer-2 characters	Named record size
FROM integer-1 TO integer-2 CHARACTERS	W[§]	Integer-2 characters plus 10	Integer-1 characters plus 10	Named record size plus 10	Integer-2 characters	Named record size
DEPENDING ON data-name in Record Description						
FROM integer-1 CHARACTERS	D	Largest record	Integer-1 characters	Data-name value	Largest record	Data-name value
TO integer-2 CHARACTERS	D	Integer-2 characters	Smallest record	Data-name value	Integer-2 characters	Data-name value
FROM integer-1 TO integer-2 CHARACTERS	D	Integer-2 characters	Integer-1 characters	Data-name value	Integer-2 characters	Data-name value
DEPENDING ON data-name not in Record Description						
FROM integer-1 CHARACTERS	W	Largest record plus 10	Integer-1 characters plus 10	Data-name value plus 10	Largest record	Calculated from data-name
TO integer-2 CHARACTERS	W	Integer-2 characters plus 10	Smallest record plus 10	Data-name value plus 10	Integer-2 characters	Calculated from data-name
FROM integer-1 TO integer-2 CHARACTERS	W	Integer-2 characters plus 10	Integer-1 characters plus 10	Data-name value plus 10	Integer-2 characters	Calculated from data-name

[†] Word-address files are always set to RT=U. Relative files are, in effect, always set to RT=F.

[††] If RT=Z, size in storage is increased if necessary to place a zero-byte terminator in bits 0 through 11 of the last word.

[†††] Record type Z if the file name is INPUT, OUTPUT, or PUNCH.

[§] Record type T if data-name is in Record Description for entry with OCCURS DEPENDING ON.

The size of the record indicated in this clause must be the number of character positions required to store the record, regardless of the types of characters used to represent the items within the logical record. For instance, items described by a USAGE clause containing COMP-1 or INDEX occupy a full memory word and have a size of 10 characters, no matter what the maximum number of digits in the value of the item; any item described with a SYNCHRONIZED clause must take slack characters into account; and so forth. The size of a record is determined by the sum of the number of characters in all elementary data items. The size of the maximum number of table elements, if any, described in the record is included in the summation. This sum might be different from the actual size of the record.

In format 1, integer-2 must not be used by itself unless all the data records in the file have the same size. In this case it specifies the exact number of characters in each record. If integer-1 and integer-2 are both specified, integer-1 is the minimum number of characters in any record and integer-2 is the maximum number of characters. Integer-2 must be greater than integer-1. Unsigned positive integers are required.

In order to provide compatibility with future ANSI standards, format 2 can be used to specify variable length records. If integer-2 is not specified in format 2, the minimum record size is the size of the smallest record defined by the record descriptions. If integer-2 is not specified, the maximum record size is the size of the longest record defined by the record description.

The DEPENDING ON phrase specifies the data item that contains, during execution, the exact number of characters in the record being referenced. Data-name, which can be qualified, must be described as an unsigned integer item. The data item can, but need not, be within the record. When it is within the record, data-name must be within the minimum size, must be described with USAGE IS DISPLAY, COMP-1, or COMP-4; its PICTURE clause can describe a maximum of 6 digits. If the file is an External file, data-name must be defined in the Common-Storage Section.

When a record described by this phrase is being written by execution of a RELEASE, REWRITE, or WRITE statement, the program must set the data item to the number of characters in the record before the output statement is executed for a given record. When such a record is read by a READ or RETURN statement, the system returns to the data item the number of characters in the record just read.

For word-address files, the program is responsible for setting the data item prior to each READ or WRITE statement for a file record.

RECORDING MODE Clause

The RECORDING MODE clause (figure 4-10) has meaning only for tape files. It specifies the data conversion, if any, that occurs between internal system codes and the external code written to the tape.

```
RECORDING MODE IS { DECIMAL }
                  { BINARY  }
```

Figure 4-10. RECORDING MODE Clause Format

When the clause is omitted, the default is determined by the Record Description entry. BINARY is the default for files with block type I and record type W (see table 4-2 and the BLOCK CONTAINS clause). DECIMAL is the default for all other files.

DECIMAL

Conversion occurs. For 7-track tape files, translation is made with External BCD codes. For 9-track tape files, translation is made with ASCII or EBCDIC codes, depending on the parameters on the control statement that requests the tape.

BINARY

No conversion occurs. BINARY should be specified for 7-track tape files when any of the records contains binary zero character codes or items described by USAGE IS INDEX, COMP-1, COMP-2, or COMP-4. BINARY must be specified for tapes in I format.

NOTE

Because of anticipated changes in this product, use of the RECORDING MODE clause is not recommended. For guidelines, see appendix F.

REPORT Clause

The REPORT clause names the reports that are to be produced. If the file is an External file, the REPORT clause cannot be specified. See section 6, Report Writer Facility.

RECORD DESCRIPTION ENTRY

A Record Description entry consists of a set of Data Description entries that furnish information about the physical structure and identification of a record. It begins with a level 01 Data Description entry and ends immediately before the next level 01 Data Description entry or the end of the section. A Record Description entry can appear in any section of the Data Division except the Report Section.

The general format is shown in figure 4-11. Data-description-entry-1 must have level number 01; data-description-entry-2 must have a level number greater than 01. Data Description entries are discussed on the following pages.

```
data-description-entry-1
[data-description-entry-2] . . .
```

Figure 4-11. General Record Description Entry Format

In the File Section, Record Description entries are associated with files. Each file can have records with different elementary and group items intermixed. A file can have more than one level 01 entry; each level 01 entry is a redefinition of the memory area associated with that file.

In sections other than the File Section, data can be grouped into logical records and defined by a series of Record Description entries, each of which describes a unique memory area, unless the entry is redefined by another Record Description entry.

DATA DESCRIPTION ENTRY

A Data Description entry specifies the characteristics of a particular item of data. Each entry consists of the following ordered elements:

Level-number

Data-name or the keyword FILLER that names the data item, or condition-name

Series of independent clauses that describe the data item

Terminating separator period

Figure 4-12 shows the skeletons of the three formats of the Data Description entry. Punctuation between format elements is optional.

Format 1 is the full entry to describe data of level 01 through 49 and level 77.

Format 2 is an entry with a 66 level-number that renames prior group or elementary items.

Format 3 is an entry with an 88 level-number that assigns condition-name values.

Level-number is the first element of a Data Description entry. Level-number consists of two digits in the range 01 through 49, 66, 77, or 88. Level-number values 1 through 9 can be written either as a single digit or as a zero followed by a digit 1 through 9.

Levels 01 through 49 describe the hierarchy of data items: that is, they define record, group, and elementary items. Levels 66, 77, and 88 have special meaning and do not define the hierarchy of the item described.

Entries with level-number 01 or 77 must begin in area A of a line; the following data-name or FILLER should begin in area B although a word in area A is accepted. If data-name and FILLER are both omitted, FILLER is assumed.

```
Format 1

        level-number  [ { data-name-1 } ]   [; REDEFINES clause]
                      [ { FILLER      } ]

             [; BLANK WHEN ZERO clause]  [; JUSTIFIED clause]  [; OCCURS clause]

             [; PICTURE clause]  [; SIGN clause]  [; SYNCHRONIZED clause]

             [; USAGE clause]  [; VALUE clause] .

Format 2

        66   data-name-1;  RENAMES clause.

Format 3

        88   condition-name; VALUE clause.
```

Figure 4-12. Data Description Entry Format Skeletons

Successive entries can be indented to improve readability of the source listing. Indentation does not affect the magnitude of a level-number.

The maximum size of any group or elementary entry is 131 071 characters.

Table 4-3 summarizes Data Description entry clauses and the level-number of the entry in which they can appear. The individual clause descriptions contain additional information about clause interactions and restrictions on clause use.

Format 1 Data Description Entry

In format 1, level-number can be 77 or in the range 01 through 49. Clauses can be in any order, except that any REDEFINES clause must immediately follow data-name or FILLER. Clauses are presented below in alphabetical order.

Level 01 through 49

Data Description entries that begin with level-number 01 through 49 can appear in any section of the Data Division. A level 01 entry with a REDEFINES clause, however, cannot appear in the File Section.

A Data Description entry with level-number 01 is a Record Description entry. Levels 02 through 49 indicate the organization of elementary and group items within the record.

An elementary item is defined as an item whose level-number is equal to or greater than the level-number of the next item. The PICTURE clause must be specified for every elementary item, except for data items described as USAGE IS INDEX or COMP-2. The clauses PICTURE, JUSTIFIED, and BLANK WHEN ZERO are valid only for elementary items.

TABLE 4-3. DATA DESCRIPTION ENTRY CLAUSE USE

Clause	Level of Entry				
	01	02-49	$77^{\dagger}$	$66^{\dagger\dagger}$	$88^{\dagger\dagger}$
BLANK WHEN ZERO	yes	yes	yes	no	no
JUSTIFIED	yes	yes	yes	no	no
OCCURS	no	yes	no	no	no
PICTURE	yes	yes	yes	no	no
REDEFINES	$yes^{\dagger\dagger\dagger}$	yes	yes	no	no
RENAMES	no	no	no	yes	no
SIGN	yes	yes	yes	no	no
SYNCHRONIZED	yes	yes	yes	no	no
USAGE	yes	yes	yes	no	no
VALUE	$yes^{\dagger\dagger\dagger}$	$yes^{\dagger\dagger\dagger}$	$yes^{\dagger\dagger\dagger}$	no	yes

†Applicable only in Common-Storage, Working-Storage, and Linkage Sections.

††Not applicable in Secondary-Storage or Report Section.

†††No in File Section.

A group item is defined as an item whose level-number is less than that of the next item. A group item includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered.

NOTE

Refer to appendix F for recommendations on the use of group items.

Level-numbers within a group item need not be consecutive; however they must be ordered so that the higher the level-number the lower the entry in the hierarchy. Items which are immediately subordinate to a group item need not have an identical level-number, but the subordinate items must have a level-number greater than that of the group item. An item whose level number is less than or equal to the level number of a given preceding item is not subordinate to that item.

A data-name or the keyword FILLER must be the first word following level-number, for strict ANSI usage. If data-name and FILLER are both omitted, FILLER is assumed.

Data-name

Specifies the name of the data item being described. Syntax must conform to the rules for user-defined words. Data-names can be referenced explicitly elsewhere in the program.

FILLER

Specifies an elementary or group item that cannot be referenced explicitly. The contents of a group item named with FILLER can be referenced only by referencing the data-names or condition-names of its constituent items.

An example of a group item with two elementary items is:

```
03  FULL-NAME.
    25 LAST-NAME PICTURE X(15).
    25 F-INITIALS PICTURE X(3).
```

Level 77

A Data Description entry with level-number 77 identifies an independent, noncontiguous data item that is not a subdivision of another data item. A level 77 entry cannot itself be subdivided. The entry is allowed only in the Common-Storage, Working-Storage, and Linkage Sections of the Data Division.

NOTE

Because of anticipated changes in this product, use of level 77 items is not recommended. For guidelines, see appendix F.

A data-name or the keyword FILLER must be the first word following 77 (for strict ANSI usage) as described in the preceding discussion for level 01 through 49 items. If data-name and FILLER are both omitted, FILLER is assumed.

Examples of level 77 entries are:

```
77 RECS-READ PICTURE 9(5).
77 CONSTANT-MULTIPLIER USAGE IS COMP-2
   VALUE IS 3.6.
```

Format 2 Data Description Entry

In format 2, the Data Description entry begins with level-number 66. Data-name must follow level 66, as described for format 1. Level 66 items rename elementary and group items. Only the RENAMES clause is valid.

The RENAMES clause is required in, and only permitted in, a level 66 entry. The entry must follow all entries of the highest group level of which the renamed item is a subentry. The RENAMES clause cannot rename a level 66, 77, 88, or 01 item.

An example of a level 66 item specification is:

```
20 IN-GROUP-1 PICTURE X(20).
20 IN-GROUP-2.
25 ITEM-2 PICTURE XXX.
25 ITEM-3 PICTURE XX.
66 OUT-GROUP RENAMES IN-GROUP-1 THRU
   IN-GROUP-2.
```

Format 3 Data Description Entry

In format 3, the Data Description entry begins with level-number 88. A condition-name must follow level 88. Level 88 entries specify condition-names that are associated with particular values of a conditional variable. (A conditional variable is a data item that is followed by one or more level 88 entries.) The conditional variable can be FILLER.

Level 88 entries must immediately follow the conditional variable to which they apply. A condition-name can be associated with any Data Description entry except: another condition-name, a level 66 item, an item described by USAGE IS INDEX, or a group item containing items specified with the JUSTIFIED or SYNCHRONIZED clause or usage other than display. The description of the conditional variable implicitly describes condition-name.

An example of level 88 item specification is:

```
05 PASS-OR-FAIL PICTURE 999.
88 PASS-EM VALUE IS 70 THRU 100.
88 FAIL-EM VALUE IS 0 THRU 69.
```

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause (figure 4-13) blanks a numeric item when a zero value is moved to that item. The clause is an alternative to a PICTURE clause with editing symbols to cause blanking. The clause can be used only for an elementary item described by USAGE IS DISPLAY and whose picture defines a numeric or numeric edited item. The category of the item is always considered to be numeric edited.

The clause cannot be used when the PICTURE clause of an item includes an asterisk for zero suppression.

<u>BLANK</u> WHEN <u>ZERO</u>

Figure 4-13. BLANK WHEN ZERO Clause Format

JUSTIFIED Clause

The JUSTIFIED clause (figure 4-14) specifies nonstandard positioning of data in a receiving data item. It can only be specified for a nonnumeric elementary item for which no editing is specified. JUST is an abbreviation for JUSTIFIED.

```
{JUSTIFIED}
{JUST     }  RIGHT
```

Figure 4-14. JUSTIFIED Clause Format

When the JUSTIFIED clause is specified for a receiving item smaller than the sending item, the leftmost characters of the sending item are truncated.

When the JUSTIFIED clause is specified for a receiving item larger than the sending item, the data is aligned at the rightmost position in the receiving item and the unused leftmost character positions of the receiving item are filled with spaces.

When the JUSTIFIED clause is omitted from a receiving item, the standard rules for aligning data within an elementary item apply.

OCCURS Clause

The OCCURS clause (figure 4-15) specifies the number of occurrences of a repeated data item and supplies information required for the application of subscripts and indexes. It is used in defining tables and other homogenous sets of repeated data items. The clause has two formats:

Format 1 specifies the exact number of times an item repeats.

Format 2 specifies the range of the number of times an item repeats and identifies the data item that determines the exact repetition during execution.

All Data Description entries subordinate to an item whose description includes an OCCURS clause apply to each repetition of the item described. An item containing both OCCURS and SYNCHRONIZED clauses is synchronized at each occurrence.

The OCCURS clause must not be specified in a Data Description entry that contains any of the following:

Level-number 01, 66, 77, or 88

Subordinate variable-occurrence data item

VALUE clause in either the entry itself or a subordinate entry

Whenever an OCCURS clause is specified for a Data Description entry, the entry and any subordinate entries must be either subscripted or indexed whenever they are referenced, except in a SEARCH statement, in a USE FOR DEBUGGING statement, or as the object of a REDEFINES clause.

All data-names in the clause can be qualified. Integers must be positive. The value of integer-2 cannot be zero or greater than the number of occurrences of the repeated data item. The maximum number of occurrences allowed for a repeated data item is 65535.

Neither integer-1 nor integer-2 can exceed 131 071 characters. The product of the number of character positions subordinate to this item and integer-1 or integer-2 cannot exceed 131 071.

Format 1 OCCURS

In format 1, the value of integer-1 represents the exact number of occurrences of the repeated data item. Data-name cannot appear in more than one OCCURS clause.

```
Format 1

    OCCURS integer-1 TIMES

        [{ASCENDING }                                        ]
        [{DESCENDING} KEY IS data-name-1 [, data-name-2] . . .] . . .

        [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] . . .]
```

Figure 4-15. OCCURS Clause Format

The KEY IS phrase indicates the order in which repeated data items are sequenced. This phrase must be included when a SEARCH ALL statement is used to search the table for an element satisfying a particular condition. The data-names are listed in their descending order of significance. Data-name-1 can be either the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the clause. Data-name-2 must be the name of an entry subordinate to the entry containing the OCCURS clause.

If data-name-1 is not the name of the entry containing the OCCURS clause, then:

> All of the items identified by the data-names in the KEY IS phrase must be within the group item that is the subject of this entry
>
> Items identified by the data-names in a KEY IS phrase must not contain an OCCURS clause
>
> No entry containing an OCCURS clause can appear between the items identified by the data-names in the KEY IS phrase and the subject of the entry. That is, a key item cannot be subordinate to an occurring item which is subordinate to the subject of this entry.

The INDEXED BY phrase specifies the indexes that are used to index repeated data items. It is required if the subject of the entry containing the OCCURS or one of its subordinate entries is to be referred to by indexing. The index-name must be a unique name that is not specified elsewhere in the program; the item is allocated and formatted by the compiler and cannot be associated with any data hierarchy.

Format 2 OCCURS

In format 2, the subject of the entry has a variable number of occurrences. Integer-1 is the minimum number of occurrences of the repeated item; integer-2, which must be greater than integer-1, is the maximum number of occurrences.

Data-name-1 is the item that contains the number of occurrences at any given time. If it is within the Record Description containing the OCCURS clause, it must be within the fixed portion; it must not be within the entry containing the OCCURS clause itself. The value of data-name-1 must fall within the range of integer-1 and integer-2, inclusive, during execution and must be unsigned. An attempt to reference an occurrence outside the limits set by the integer-1 and integer-2 range causes unpredictable results.

When a format 2 clause is subordinate to a group item, a reference to the group item involves only the part of the table area that is specified by the value of data-name-1. Data names are evaluated once before the reference operation takes place.

The KEY IS and INDEXED BY phrases are the same as for format 1.

A format 2 OCCURS clause can only be followed by subordinate entries of higher levels, not by Data Description entries of the same or lower levels. Some example of OCCURS clause usage are shown in figure 4-16.

PICTURE Clause

The PICTURE clause (figure 4-17) describes the general characteristics and editing requirements for an elementary item. (The physical representation of data within memory is influenced by the USAGE clause.) It must be specified for every elementary item, except that it must not be specified for an item with a USAGE clause specifying INDEX or COMP-2.

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

Figure 4-17. PICTURE Clause Format

The character-string specifies, by its combination of allowable characters from the COBOL character set, the category of the elementary item. It also specifies the size, location of decimal point, and the presence or absence of a sign.

a. The table PARTS-LIST contains 1000 entries, 100 lower level entries for each 03 level entry and 10 entries at the 03 level. To reference the table with subscripts it must be redefined using OCCURS:

```
01     PARTS-TABLE REDEFINES PARTS-LIST.
       03     LIST OCCURS 10 TIMES.
              05     PART PICTURE 9(10) OCCURS 100 TIMES.
```

b. A table to be searched by the SEARCH ALL statement can be described as:

```
02     DATA-LIST OCCURS 25 TIMES INDEXED BY LIST-INDEX,
       DESCENDING KEY IS ITEM-KEY, ITEM-KEY-2.
       04     ITEM-KEY PICTURE X.
       04     ITEMA OCCURS 10 TIMES INDEXED BY ITEM-INDEX.
              06     AVALUE PICTURE 9(7).
              88     VALA VALUES ARE 1 THRU 999999.
       04     ITEM-KEY-2 PICTURE IS 99.
```

Figure 4-16. Example of OCCURS Clause

The maximum number of symbols allowed in the character-string of the clause format is 30. Since repeating symbols can be indicated by an unsigned nonzero integer enclosed in parentheses after the symbol to be repeated, the character-string limit does not limit the value of the data or the number of character positions in the item in storage.

The six categories of data that can be specified by a PICTURE character-string and the class to which each belongs are shown in table 4-4. Table 4-5 specifies the symbols that can appear in the character-string for each category and the contents allowed in the data item.

Every elementary item belongs to both a category and a class. Any group item is treated during execution as if it belonged to the alphanumeric class, regardless of the class of elementary items subordinate to that group item.

When specifying a PICTURE character-string, the following apply:

The number of character positions described cannot exceed 131 071.

TABLE 4-4. CLASSES AND CATEGORIES OF DATA ITEMS

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Boolean	Boolean
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Group	Alphanumeric	Alphabetic Boolean Numeric Numeric Edited Alphanumeric Edited Alphanumeric

TABLE 4-5. PICTURE CLAUSE SYMBOLS ALLOWED BY ITEM CATEGORY

Category		Allowable Symbols in PICTURE Character-String	Contents of Item	Usage Examples		
				PICTURE of Item	Source Data	Item
Alphabetic		A B	Letters and spaces	AAAAA or A(5)	COSTS	C O S T S
Boolean		1	Numerals 0 and 1	111 or 1(3)	B "011"	0 1 1
Numeric	Unsigned	9 V P	Digits	999	123	1 2 3
				99V999	12.345	1 2 ↑ 3 4 5
				PPP9999	.0001234	↑0 0 0 1 2 3 4
	Signed	9 V P S	Digits + -	S99V99	+12.34	1 2 ↑ 3 4
				SPPP9999	-.0001234	↑0 0 0 1 2 3 4
				S999PPP	-123000.	1 2 3 0 0 0↑
Alphanumeric (without editing)		A X 9	All characters in COBOL character set	XXXXXXXX or X(8)	ABCD-***	A B C D - * * *
				XXXXXXXX or X(8)	123.4567	1 2 3 . 4 5 6 7
				AAAA999 or A(4)9(3)	ABCD123	A B C D 1 2 3
Alphanumeric Edited		A X 9 B 0 /	All characters in COBOL character set	AA00AA	WXYZ	W X 0 0 Y Z
				XXBBXX	A1B1	A 1 Δ Δ B 1
				X0X0X0	ZXY	Z 0 X 0 Y 0
				XB(3)X0(4)X	N15	N Δ Δ Δ 1 0 0 0 0 5
Numeric Edited		B / P V Z 0 9 , . * + - CR DB Currency symbol	Digits	See the discussions below: Floating Insertion Editing Zero Suppression and Replacement Editing		

↑ assumed decimal point
Δ blank

The size of an elementary item is determined by allowable symbols that represent character positions. Multiple repeating symbols can be indicated by an unsigned nonzero integer enclosed in parentheses following the symbols A , X 9 P Z / * 0 + - B 1 or the currency symbol.

The symbols S V . CR and DB can appear only once in a given picture.

Numeric and numeric edited items are limited to a maximum of 18 digit positions. A sign described with a SIGN IS SEPARATE phrase occupies a character position but is not included in the digit position count.

A numeric edited item must contain at least one of the symbols 0 B / Z * + , . - CR DB or the currency symbol.

An alphanumeric edited item must contain at least one of the following combinations of symbols:

B and X	0 and A
0 and X	/ and A
/ and X	

<u>Symbols Used in Character-Strings</u>

Each of the symbols used to describe an elementary item has a specific function.

A Represents a character position that can contain either a letter or a space.

B Represents a character position into which the space character is to be inserted.

P Indicates an assumed decimal scaling position and the location of the assumed decimal point when the point is not within the number that appears in the data item.

The scaling position character P:

Is not counted in the size of the data item, but is counted in determining the maximum number of digit positions (18) in numeric edited and numeric items.

Is considered to contain the value zero.

Can only appear as the leftmost or rightmost character or continous character string of P's within a PICTURE description, implying an assumed decimal point to the left or to the right, respectively.

Is redundant when used with the V character.

Cannot appear in the same character-string as the symbol . (period).

S Indicates the presence of an operational sign. It does not necessarily specify either the position or the representation of the sign. See the SIGN clause.

The S is not counted in determining the size of the item unless the entry includes a SIGN clause that specifies the optional SEPARATE CHARACTER phrase.

V Indicates the location of the assumed decimal point. It does not represent a character position and is not counted in the size of the data item.

Only one V can appear in a character-string. It is redundant in a character-string that has the scaling position character P and redundant as the rightmost character in a character-string.

X Represents a character position that can contain any allowable character.

Z Represents a leftmost leading numeric character position that is to be replaced by a space character when the contents of that character position is zero.

1 Represents a boolean character position that can contain either a zero or a one. It represents a standard data format character.

9 Represents a digit position that can contain a numeral.

0 Represents a character position into which a zero is to be inserted.

/ Represents a character position into which the slash character is to be inserted.

, Represents a character position into which the comma is to be inserted.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the function of the period and comma are exchanged, causing the rules for the period to apply to the comma and the rules for the comma to apply to the period.

. Represents the decimal point for alignment purposes and a character position into which the period is to be inserted.

If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the function of the period and comma are exchanged, causing the rules for the period to apply to the comma and the rules for the comma to apply to the period.

CR Represents two character positions into which the editing sign CR is to be inserted. Only one of the editing sign control symbols + - CR or DB can be used in a given character-string.

DB Represents two character positions into which the editing sign DB is to be inserted. Only one of the editing sign control symbols + - CR or DB can be used in a given character-string.

+ Represents a character position into which the plus sign is to be inserted. Only one of the editing sign control symbols + - CR or DB can be used in a given character-string.

- Represents a character position into which the minus sign is to be inserted. Only one of the editing sign control symbols + - CR or DB can be used in a given character-string.

* Represents a leading numeric character position into which an asterisk is to be inserted when the content of the position is zero.

cs The currency symbol represents a character position into which the designated currency sign is to be inserted.

The currency symbol can be either the currency sign ($ and #, or its equivalent graphic character) or the single character specified in the CURRENCY SIGN clause of the SPECIAL-NAMES paragraph.

The symbol $ or # or the character designated by CURRENCY SIGN clause are valid currency symbols in the same program, but not in the same picture.

Table 4-6 summarizes the symbols of the clause, the number of times they can be used in a single PICTURE clause character-string, and the resulting size.

Editing Rules

Two types of editing can be specified in the PICTURE clause:

Insertion editing, which places a character in the position indicated. Different types of insertion editing are: simple insertion, special insertion, fixed insertion, and floating insertion.

Replacement editing, which replaces leading zeroes in the data with spaces or asterisks.

The type of editing that can be defined for a given data item depends on the category to which the item belongs.

TABLE 4-6. PICTURE CLAUSE EDITING SYMBOL FUNCTIONS AND LIMITS

Symbol	Function Symbol Represents	Number of Times Symbol Can Occur in Picture	Number of Positions In Total Item Size
A	Letter or space	no limit	1 for each symbol
B	Space	no limit	1 for each symbol
P	Assumed decimal scaling position	no limit†	none
S	Operational sign	1	1 if SEPARATE CHARACTER; otherwise none
V	Location of assumed decimal if P omitted	1†	none
X	Any language character	no limit	1 for each symbol
Z	Blank leading zero indicator	no limit†	1 for each symbol
9	Digit position	18†	1 for each symbol
1	Boolean character	no limit†	1 for each symbol
0	Zero insertion position	no limit†	1 for each symbol
/	Slash insertion position	no limit†	1 for each symbol
,	Comma insertion position	no limit†	1 for each symbol
.	Decimal point position if P omitted	1	1
CR + DB -	Sign position	1 of 4 symbols	2 for CR or DB; 1 for + or -
*	Asterisk replace leading zero indicator	no limit†	1 for each symbol
currency symbol	Currency symbol position	1	1

† If total of digits and editing characters exceeds 18 for numeric and numeric edited items, truncation occurs during item use.

Category	Editing Allowed
Alphabetic	Simple insertion of space
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion of 0 space and/or /
Numeric edited	All
Boolean	None

Simple Insertion Editing - Simple insertion editing is specified in a PICTURE character-string by an insertion symbol B 0 , or / . An insertion symbol represents the position in an item into which a space, zero, comma, or slash is inserted when data is placed in that item.

When the comma is the rightmost symbol in a PICTURE character-string, the PICTURE clause must be the last clause of the Data Description entry. This results in either the combination of , . or, if the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, two consecutive periods appearing in the Data Description entry.

Special Insertion Editing - Special insertion editing is specified in a PICTURE character-string by the period. The period, which represents the decimal point for alignment purposes, is counted in the size of the item.

When a decimal point is the rightmost symbol in a PICTURE character-string, the PICTURE clause must be the last clause of the Data Description entry. This results in either two consecutive periods or, if the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the combination of , . appearing in the Data Description entry.

Fixed Insertion Editing Fixed insertion editing is specified in a PICTURE character-string by a currency symbol or one of editing sign control symbols + - CR or DB . The rules for fixed insertion editing are:

Only one currency symbol and one editing sign control symbol can appear in a given PICTURE character-string.

The symbols CR and DB represent the two rightmost character positions of the item.

The symbol + or - can be either the leftmost or the rightmost character position of the item.

The currency symbol must be the leftmost character position to be counted in the size of the item, except that it can be preceded by a + or - symbol.

The final contents produced by the editing sign control symbols is affected by the value of the data item being edited:

Editing Symbol	Result When Data Item Positive or Zero	Result When Data Item Negative
+	+	-
-	1 space	-
CR	2 spaces	CR
DB	2 spaces	DB

Floating Insertion Editing Floating insertion editing is specified in a PICTURE character-string by a string of at least two floating insertion symbols. Floating insertion editing results in characters appearing in the data item in a position appropriate for each particular data value, within the limits of the editing specification.

The floating insertion editing symbols are the currency symbol, and the + and - signs. If the fixed insertion symbols CR or DB are part of the string, they must be the rightmost symbols. The currency symbol and the + or - symbol are mutually exclusive when they are used as floating insertion symbols.

The leftmost character of the floating string represents the leftmost limit of the floating symbol in the data item, and the rightmost character represents the rightmost limit. The second leftmost character of the floating string represents the limit of the numeric data that can be stored in the item.

The floating string can be structured in two ways: either any or all of the leading numeric character positions to the left of the decimal point are represented by the insertion character, or all of the numeric character positions are represented by the insertion character.

When the insertion characters are only to the left of the decimal point, a single floating insertion character is placed in the character position immediately preceding either the decimal point or the first nonzero digit in the item, whichever is farther to the left in the PICTURE character-string. The characters preceding the placement of the floating insertion character are replaced with spaces.

When all numeric characters in the character string are represented by the floating insertion character and the entire data item is zero, the entire data item will be replaced with spaces. If the data is not all zeros, floating insertion editing is handled the same as if the insertion characters were not placed to the right of the decimal point.

Simple insertion editing characters , B 0 and / can appear among the floating string characters, but cannot be the leftmost symbols in the string. Floating string processing takes precedence in this instance: spaces always appear to the left of the properly placed floating character. Simple insertion characters that are not enclosed in the floating string appear as indicated in the character-string of the clause.

Some examples of floating insertion editing are:

Picture	Source Data	Item
$$$9.99	0008↑24	△△$8.24
---9.99	-005↑26	△△-5.26
+++9.99	034↑56	△+34.56
$$$.99	32↑65	$32.65
$$$$9	1234	$1234
$$$$9	0123	△$123
$$$,$ZZ.99	000001	Illegal Picture
$$$,$$$.99	55↑55	△△△△$55.55

Zero suppression and replacement editing is specified in the PICTURE character-string by a string of one or more of the symbols Z or * . Any simple insertion characters that are either embedded in the string or to the immediate right of the string are part of the replacement string. If Z is specified, a leading zero in that position in the receiving field is replaced by a space; if the asterisk is specified, the leading zero is replaced by an *.

The replacement string can be structured in two ways: either any or all of the leading numeric character positions to the left of the decimal point are represented by the replacement symbol, or all of the numeric character positions are represented by the replacement symbol.

When replacement symbols are designated to the left of the decimal point, the replacement character is placed into all character positions immediately preceding either the decimal point or the first nonzero digit in the item, whichever is further to the left in the PICTURE character-string.

When all numeric characters in the character-string are represented by Z and the entire receiving item is zero, the entire data item is set to spaces. When all numeric characters in the character string are represented by * and the entire receiving item is zero, the receiving item is set to * except for the decimal point. If the field is not zero, replacement insertion is handled as if the replacement characters were designated only to the left of the decimal point.

Some examples of zero suppression and replacement editing are:

Picture	Source Data	Item
ZZ999	00923	ΔΔ923
ZZZ99	00923	ΔΔ923
****.**	000000	****.**
$***.99	009‸23	$**9.23
ZZ9999	123456	123456
ZZ9999	000005	ΔΔ0005
ZZZZZ	00010	ΔΔΔ10
Z/Z/9	023	ΔΔ2/3

Editing Precedence Rules

Table 4-7 shows the order in which symbols can appear in the PICTURE clause character-string. The character-string must contain one of the following:

At least one symbol A X Z 9 or *

At least one string of floating insertion symbols

In the table, several symbols appear twice: the leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

REDEFINES Clause

The REDEFINES clause (figure 4-18) gives a new data-name and description for a previously specified item of the same level. Storage is not allocated for the new data-name: only one item physically exists. The item can be referenced by either data-name. Any use of the data is in accordance with the description associated with the data-name by which the item is referenced.

NOTE

Refer to appendix F for recommendations on the use of the REDEFINES clause.

<u>REDEFINES</u> data-name-2

Figure 4-18. REDEFINES Clause Format

The clause, when it is specified, must be the first one in the Data Description entry. The Data Description entry with the new name must immediately follow the entry for the item being redefined, with no intervening items of lower level number. The entry might appear in the Working-Storage Section, for example, as:

```
05  NAME-ORIG PICTURE 9(5).
05  NAME-NEW REDEFINES NAME-ORIG.
    06  NEW-1 PICTURE 9(4).
    06  NEW-2 PICTURE 9(1).
05  NAME-NEW2 REDEFINES NAME-ORIG
    PICTURE 9999V9.
```

Multiple redefinitions of the same item are possible. All redefinitions of the same storage area should reference the original data-name, although a redefinition of an intervening redefinition is permitted.

In the clause, data-name-2 is the item that is being redefined. (Data-name-1 is the data-name of the new item.) The level number associated with both data-names must be identical. Level 01 entries in the File Section, level 66 items, and level 88 items cannot be referenced in this clause.

Data-name-2 must not be described by an OCCURS clause, but can be subordinate to an item that contains an OCCURS clause. Data-name-2 cannot be subscripted, indexed, or qualified.

Data-name-1, the new data-name, cannot be an elementary item with a SYNCHRONIZED clause nor a group item whose first elementary item has a SYNCHRONIZED clause unless data-name-2 has the proper boundary alignment. The new description must not contain a VALUE clause nor the DEPENDING ON phrase of an OCCURS clause.

The sizes of data-name-1 and data-name-2 need not be the same, since storage is allocated for the larger of two descriptions. Redefinition ends when all data referenced by data-name-2 is redefined.

TABLE 4-7. PICTURE CLAUSE SYMBOL PRECEDENCE RULES

Second Symbol	First Symbol	Nonfloating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	{+ -}	{+ -}	{CR DB}	cs	{Z *}	{Z *}	{+ -}	{+ -}	cs	cs	9	A X	S	V	P	P	1
Nonfloating Insertion Symbols	B	X	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			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	X	X	
	cs						X																
Floating Insertion Symbols	(Z *)	X	X	X	X		X			X	X												
	(Z *)	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								X								
	cs	X	X	X	X	X	X								X	X				X		X	
Other Symbols	9	X	X	X	X	X	X			X	X		X		X		X	X	X	X		X	
	A X	X	X	X													X	X					
	S																						
	V	X	X	X	X		X			X	X		X		X		X		X		X		
	P	X	X	X	X		X			X	X		X		X		X		X		X		
	P						X			X									X	X		X	
	1																						X

X	The symbol at the top of the column can precede the symbol at the left of the row.
{ }	Symbols within the braces are mutually exclusive.
cs	Currency symbol.

RENAMES Clause

The RENAMES clause (figure 4-19) specifies an alternative grouping of elementary items, with possible overlap, under a new name. The clause must be specified in a level 66 item; it cannot reference level 77, 88, or 01 items.

```
RENAMES data-name-2  [ {THRU   }  data-name-3 ]
                       {THROUGH}
```

Figure 4-19. RENAMES Clause Format

When RENAMES is specified, the Data Description entry cannot contain other clauses. The full entry in which the clause appears is:

66 data-name-1 ; RENAMES data-name-2 .

One or more RENAMES clauses can be written for each record. The clauses must immediately follow the last entry of the associated record.

NOTE

Refer to appendix F for recommendations on the use of the RENAMES clause.

In the Data Description entry data-name-1 specifies the name of the alternate grouping of one or more elementary items. Data-name-1 cannot be used as a qualifier; however, it can be qualified by the name of the associated level 01, FD or SD entry.

In the RENAMES clause, data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same record. They cannot have the same name, but the names can be qualified. Neither data-name-2 nor data-name-3 can be described by an OCCURS clause or be subordinate to an item described by an OCCURS clause.

If the THRU phrase is not specified, data-name-1 can be either an elementary item or a group item since it has all the attributes of data-name-2.

When the THRU phrase is specified, data-name-2 and data-name-3 designate a range of elementary items that make up the group item data-name-1. None of the items can be described by the DEPENDING ON phrase of the OCCURS clause. Data-name-1 includes all items within the range, beginning with the first, or only, elementary item specified by data-name-2 and ending with the last, or only, elementary item specified by data-name-3.

When data-name-3 is specified, it cannot name an item that is defined before data-name-2 nor one that is subordinate to data-name-2; it can, however, start within the item specified by data-name-2.

SIGN Clause

The SIGN clause (figure 4-20) specifies the position and the representation of the operational sign for an item. It can only be specified for a numeric or numeric edited item described by a PICTURE clause containing the symbol S and a USAGE IS DISPLAY clause, for a group item containing at least one such entry, or for both if the SIGN clauses are not contradictory.

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

Figure 4-20. SIGN Clause Format

If the SIGN clause is not specified for a numeric item, the SIGN CONTROL clause of the SPECIAL-NAMES paragraph, if present, specifies sign control for the item. If neither is present, SIGN IS TRAILING is assumed.

LEADING or TRAILING indicates whether the sign is at the beginning or the end of the item, respectively. Default is TRAILING.

SEPARATE specifies that the positive and negative signs are the characters + and - respectively. When SEPARATE is stated, the symbol S in the PICTURE clause character-string results in one character added to the size of the item.

If SEPARATE is omitted, the sign is combined with the first or last digit of the item. When the item is displayed in output or is received in input as a card image, the signed digit appears as specified in the second column of table 4-8. When the item is to be received as input from a card, the signed digit must be punched as specified in the third column of table 4-8. When input data is positive or unsigned, output data is the same as the input data. The negative sign is represented by a - overpunch in row 11; the positive sign by the absence of an overpunch or the presence of a + overpunch in row 12.

TABLE 4-8. SIGN OVERPUNCH REPRESENTATION

Sign and Digit	Output Representation	Hollerith Punch
+9	I	12-9
+8	H	12-8
+7	G	12-7
+6	F	12-6
+5	E	12-5
+4	D	12-4
+3	C	12-3
+2	B	12-2
+1	A	12-1
+0	<	12-0[†]
-0	V	11-0[†]
-1	J	11-1
-2	K	11-2
-3	L	11-3
-4	M	11-4
-5	N	11-5
-6	O	11-6
-7	P	11-7
-8	Q	11-8
-9	R	11-9

[†]Under NOS, the 029 keypunch cannot be used to make the Hollerith punch patterns that represent +0 or -0.

SYNCHRONIZED Clause

The SYNCHRONIZED clause (figure 4-21) specifies that an elementary item is to be aligned on word boundaries of computer memory. The alignment is such that no other data item occupies any of the character positions in the word in which the elementary item is synchronized. If the number of character positions in the elementary item is less than a full word, or multiple of a full word, the unfilled positions are known as slack character positions.

```
{ SYNCHRONIZED }  [ LEFT  ]
{ SYNC         }  [ RIGHT ]
```

Figure 4-21. SYNCHRONIZED Clause Format

The SYNCHRONIZED clause, or its abbreviation SYNC, should only appear with an elementary item; If it appears with a group item, it is interpreted as though it were written for each subordinate elementary item.

All level 01 Record Description entries are automatically aligned to begin on word boundaries. COMP-1, COMP-2, and INDEX items are always aligned within word boundaries regardless of whether or not the SYNCHRONIZED clause is specified.

If SYNCHRONIZED LEFT is specified for a COMPUTATIONAL or DISPLAY item, the item is left justified in the computer word; if SYNCHRONIZED RIGHT is specified, the item is right justified within the word. When SYNCHRONIZED is not followed by either LEFT or RIGHT, LEFT is assumed for all items except level 77 items. Level 77 items are right justified unless the ANSI=77LEFT parameter, which causes left synchronization, is specified on the compiler call.

The operational sign of an item appears in the normal operational sign position regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

Slack character positions are the unused positions in computer words for synchronized items of less than a multiple of 10 characters. The contents of the slack characters are undefined and can be changed by any store into the associated data item.

The slack characters are included in the size of any group item to which the elementary item belongs. The slack characters are also counted in the size of a redefined item and in determining any action on an item that depends on size, such as justification, truncation, or overflow.

Slack characters are included as follows:

> If the item being synchronized immediately follows an elementary item, the slack character positions are inserted as a filler item immediately before the item being synchronized.
>
> If the item being synchronized immediately follows a group item, the slack character positions are inserted as a filler item between the immediately preceding elementary item and the following group item.

The number of slack character positions inserted for left synchronization, right synchronization, or group items described with an OCCURS clause is determined as follows:

> The total number of character positions occupied by all elementary items preceding the item to be synchronized, including any slack character positions previously added, are added together.
>
> This sum is divided by 10, giving a remainder R1.
>
> If the remainder R1 is equal to zero, no slack character positions are required. Otherwise, 10 - R1 slack character positions are required.

USAGE Clause

The USAGE clause (figure 4-22) specifies the manner in which a data item is stored in memory. Storage form does not restrict the use of the data, although some Procedure Division statements might require data items to be declared by a particular USAGE clause. If the USAGE clause is omitted for a data item or for any group to which the data item belongs, the default is USAGE IS DISPLAY.

```
              ⎧ COMPUTATIONAL   ⎫
              ⎪ COMP            ⎪
              ⎪ COMPUTATIONAL-1 ⎪
              ⎪ COMP-1          ⎪
              ⎪ COMPUTATIONAL-2 ⎪
[USAGE IS]   <  COMP-2           >
              ⎪ COMPUTATIONAL-4 ⎪
              ⎪ COMP-4          ⎪
              ⎪ DISPLAY         ⎪
              ⎩ INDEX           ⎭
```

Figure 4-22. USAGE Clause Format

The USAGE clause can be written at any level. If it is written at a group level, it applies to each elementary item within the group. The USAGE clause of further elementary and group items cannot conflict with the USAGE clause of their group level items.

COMP, COMP-1, COMP-2, and COMP-4 are used as abbreviations for the words COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-4, respectively.

> COMPUTATIONAL
>
> A COMPUTATIONAL item must be of the numeric class and have decimal numeric values. (See the PICTURE clause for the attributes of a numeric item.) Each digit is stored by its display code representation, unless the CCl parameter is used on the compiler call. The SYNCHRONIZED clause can be used with a COMPUTATIONAL item to control data placement.
>
> If a group item is described as COMPUTATIONAL, the elementary items within the group item are COMPUTATIONAL; however, the group item itself is considered to be DISPLAY and cannot be used in computations.
>
> A COMPUTATIONAL item whose PICTURE clause character-string does not contain an operational sign is assumed to be positive when used as a sending item and is made positive when it is the receiving item.
>
> Although a COMPUTATIONAL item provides efficient storage, it requires conversion to integer or floating point format for multiplication, division, and exponentiation.
>
> COMPUTATIONAL-1
>
> A COMP-1 item is of the numeric class. Its PICTURE clause character-string must not contain more than 14 symbols 9 and P.
>
> A COMP-1 item occupies a full computer word and is represented internally as a 48-bit binary integer, right-aligned in the word. The item is capable of holding larger values than indicated by a 14-symbol PICTURE character-string, but any test for overflow condition is based on the maximum value implied by the PICTURE clause of the result field.

When a COMP-1 item is used as a receiving field, no truncation is performed unless the result exceeds 48 bits; the PICTURE character-string for the receiving field is ignored in truncation.

When the PICTURE clause character-string of a COMP-1 item does not contain an operational sign character, the compiler assumes the value is positive if it is a sending field and makes the sign positive if it is a receiving field.

COMPUTATIONAL-2

A COMP-2 item is represented internally as a single precision floating point number, which occupies a full word of memory. No PICTURE clause can be included in a Data Description entry whose USAGE IS COMP-2. The compiler assumes that the value of a COMP-2 item is a signed normalized floating point number.

COMP-2 items maintain accuracy to 14 significant digits. Their value must be:

$$-10^{322} \leq \text{value} \leq -10^{-293}$$

$$0$$

$$10^{-293} \leq \text{value} \leq 10^{322}$$

COMPUTATIONAL-4

A COMP-4 item is of the numeric class. Its PICTURE clause character-string must not contain more than 14 symbols 9 and P.

A COMP-4 item is represented internally as a binary integer with a maximum size of 48 bits. The item is capable of holding larger values than indicated by a 14-symbol PICTURE character-string, but any test for overflow condition is based on the maximum value implied by the PICTURE clause of the result field.

A COMP-4 item can be signed or unsigned. If it is signed, the sign is represented in the leftmost bit of the item. If the item is unsigned, the compiler assumes the value is positive if it is a sending field and makes the sign positive if it is a receiving field.

When a COMP-4 item is used as a receiving field and truncation is necessary, it is performed in binary and without regard to the sign.

Table 4-9 shows the number of bits or bytes required to represent the maximum value for any number of digits 1 through 14 specified in the PICTURE clause character-string. Table entries have the following significance:

> Bits required for unsigned items is the number of bits required to store the maximum decimal value which can be represented by the specified number of digits in the PICTURE clause.
>
> Bits required for signed items, is the number of bits required for unsigned items, plus one.
>
> Bits rounded is the smallest multiple of six which is greater than or equal to the number of bits required.
>
> Bytes required is the number of bits rounded divided by six. The number of bytes must be used for data storage, because the smallest addressable unit is one byte.

For example, if the COMP-4 item has PICTURE 99, then the number of digits in the PICTURE clause is 2. Seven bits are required to store the maximum numeric value which can be represented by PICTURE 99, or decimal 99. That is, seven bits can store numeric values up to decimal 127; whereas, six bits can only store numeric values up to decimal 63. A multiple of six bits must be used for data storage. The smallest multiple of six which is greater than or equal to 7 is 12. Therefore, the minimum number of bits rounded for this example is twelve bits or two bytes.

DISPLAY

A DISPLAY item is stored internally in display code. The category of the item can be alphabetic, alphanumeric, alphanumeric edited, numeric, numeric edited, or boolean.

TABLE 4-9. STORAGE REQUIREMENTS FOR COMP-4 ITEMS

Digits in PICTURE Clause	Unsigned Items			Signed Items		
	Bits Required	Bits Rounded	Bytes Required	Bits + 1 Required	Bits + 1 Rounded	Bytes Required
1	4	6	1	5	6	1
2	7	12	2	8	12	2
3	10	12	2	11	12	2
4	14	18	3	15	18	3
5	17	18	3	18	18	3
6	20	24	4	21	24	4
7	24	24	4	25	30	5
8	27	30	5	28	30	5
9	30	30	5	31	36	6
10	34	36	6	35	36	6
11	37	42	7	38	42	7
12	40	42	7	41	42	7
13	44	48	8	45	48	8
14	47	48	8	48	48	8

INDEX

An INDEX item is represented internally in binary form and occupies a full computer word in memory. No PICTURE, JUSTIFIED, VALUE, or BLANK WHEN ZERO clauses can be included in the Data Description entry when USAGE IS INDEX. An item described with this clause is known as an index data item.

An index data item contains a value that corresponds to an occurrence number of a table element. It cannot be a conditional variable.

If a group item is described with the USAGE IS INDEX clause, the elementary items within the group are index data items, but the group itself is not. The group item cannot be used as an index data item.

An index data item can only be referenced explicitly in a SEARCH statement, a SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

An index data item is not represented internally in the same format as an index-name that is defined in the INDEXED BY phrase of an OCCURS clause. (See section 6 of the COBOL user's guide for further detail.)

VALUE Clause

The VALUE clause (figure 4-23) specifies the initial value of an item at the start of the object program or the value of a level 88 condition-name.

Format 1 either specifies a Report Section printable item or specifies an elementary or group item in the Working-Storage, Common-Storage, or Secondary-Storage Section.

Format 2 specifies a condition-name in a level 88 item in the Working-Storage Section.

```
Format 1

   VALUE IS literal

Format 2

  { VALUE IS   }              [ { THRU    } literal-2 ]
  { VALUES ARE }  literal-1   [ { THROUGH }           ]

        [ , literal-3  [ { THRU    }  literal-4 ] ]  . . .
        [              [ { THROUGH }            ] ]
```

Figure 4-23. VALUE Clause Format

Format 1 VALUE

The VALUE clause defines the value of Report Section printable items, the initial value of Working-Storage items, and the initial value of data items in the Common-Storage Section.

If the clause is not specified in an item description, the initial value of the item is unpredictable.

The VALUE clause must not be specified for:

An entry containing an OCCURS clause

An entry containing a REDEFINES clause

An entry subordinate to an OCCURS clause or a REDEFINES clause

A group entry containing items with descriptions including a JUSTIFIED or SYNCHRONIZED clause, or a USAGE clause that does not specify DISPLAY

In format 1, the literal specified can be a numeric literal, nonnumeric literal, a boolean literal, or a figurative constant. The following rules apply:

If the PICTURE clause defines a boolean item, all literals must be boolean and adhere to the PICTURE character-string specifications.

If the PICTURE clause defines a numeric item, all literals must be numeric and adhere to the PICTURE character-string specifications.

If the PICTURE clause defines an alphabetic, alphanumeric, alphanumeric edited, numeric edited item, all literals must be nonnumeric. The literal will be aligned as if the item were alphanumeric.

If the item is a group level entry, the literal must be a figurative constant or a nonnumeric literal. The group item is initialized without consideration for the individual elementary and group items contained within the group. The elementary and group items within a group item specifying a VALUE clause must not themselves contain a VALUE clause.

When the initial value set by the VALUE clause remains unchanged during program execution, the item is known as a constant.

Format 1 can be used to specify a level 88 item.

In a report group, the VALUE clause is one of the three clauses that can be used for the required definition of the purpose of each elementary item. The discussion of the GROUP INDICATE clause in section 6 further explains the effects of the VALUE clause on a Report Section item.

Format 2 VALUE

Format 2 must only be used to specify the range of values for a condition-name in a level 88 entry. Only the condition-name and the VALUE clause are allowed in the entry.

The literals can be numeric, nonnumeric, or a figurative constant. If the PICTURE clause describes a boolean item, the literal can be a boolean literal. Literal-2 must be greater than literal-1; literal-4 greater than literal-3, and so forth.

COMMUNICATION DESCRIPTION ENTRY

A Communication Description (CD) entry defines the two types of Message Control System (MCS) interface areas in a COBOL program. The two interface areas, CD input and CD output, are referenced by six statements in the Procedure Division to perform input-type and output-type functions for MCS. The data in these two areas is used for communicating information about the message being handled between MCS and the program. Interface with MCS is only allowed under NOS.

CD Input

The CD input area (figure 4-24) specifies the CD level indicator, a cd-name (communication description name), and a series of independent clauses. The independent clauses include the symbolic queue and sub-queue names, message date, message time, source, text length, end key, status key, message count, and the user-defined data-names. The independent clauses can be written with only the data-names.

These independent clauses are used to execute the statements RECEIVE, ACCEPT MESSAGE COUNT, DISABLE INPUT, and ENABLE INPUT. A COBOL program that receives messages must define at least one CD input area.

Each CD input record area is 87 contiguous character positions. The record area consists of symbolic name clauses that are interpreted as data-names with picture descriptions and level-numbers. The record area is defined to MCS as follows:

SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item with a field length of 12 characters that occupies positions 1-12 in the record.

SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item with a field length of 12 characters that occupies positions 13-24 in the record.

SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item with a field length of 12 characters that occupies positions 25-36 in the record.

SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item with a field length of 12 characters that occupies positions 37-48 in the record.

MESSAGE DATE clause defines data-name-5 as the name of a data item that internally describes a 6-digit integer without an operational sign and that occupies character positions 49-54 in the record.

MESSAGE TIME clause defines data-name-6 as the name of a data item that internally describes an 8-digit integer without an operational sign and that occupies character positions 55-62 in the record.

SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters that occupies positions 63-74 in the record.

TEXT LENGTH clause defines data-name-8 as the name of an elementary data item that internally describes a 4-digit integer without an operational sign and that occupies character positions 75-78 in the record.

END KEY clause defines data-name-9 as the name of an alphanumeric data item of one character that occupies position 79 in the record.

STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of two characters that occupies positions 80-81 in the record.

```
Format 1

CD  cd-name;            FOR      [INITIAL]      INPUT

  ┌ ┌                                                              ┐
  │ │[ ;  SYMBOLIC QUEUE IS data-name-1 ]                          │
  │ └                                                              │
  │      [ ; SYMBOLIC SUB-QUEUE-1        IS      data-name-2 ]     │
  │      [ ; SYMBOLIC SUB-QUEUE-2        IS      data-name-3 ]     │
  │      [ ; SYMBOLIC SUB-QUEUE-3        IS      data-name-4 ]     │
  │      [ ; MESSAGE DATE                IS      data-name-5 ]     │
  │      [ ; MESSAGE TIME                IS      data-name-6 ]     │
  │      [ ; SYMBOLIC SOURCE             IS      data-name-7 ]     │
  │      [ ; TEXT LENGTH                 IS      data-name-8 ]     │
  │      [ ; END KEY                     IS      data-name-9 ]     │
  │      [ ; STATUS KEY                  IS      data-name-10 ]    │
  │      [ ; MESSAGE COUNT               IS      data-name-11 ] ]  │
  │[data-name-1, data-name-2, . . . . . . data-name-11]            │
  └                                                                ┘
```

Figure 4-24. CD Input Area

MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item that internally describes a 6-digit integer without an operational sign and that occupies character positions 82-87 in the record.

INITIAL Clause

The INITIAL clause specifies a communication description (CD) input area that is updated by MCS before program execution. The INITIAL clause can be specified in only one CD input area within a single COBOL program. The INITIAL clause must not be used in a program that specifies the USING phrase in the Procedure Division header.

Each COBOL program containing an INITIAL CD input area automatically calls MCS prior to the execution of its first Procedure Division statement. If MCS scheduled the job stream and this is the first call to MCS, then MCS fills in the queue name fields of the INITIAL CD input area. If MCS did not initiate the job stream or this is not the first call to MCS, then MCS initializes the queue name fields of the INITIAL input CD input area to spaces. Therefore, if a job stream scheduled by MCS contains more than one COBOL program, the first program that uses MCS should contain an INITIAL CD input area so the queue names from MCS are not lost.

Whenever a program is scheduled for execution by MCS, the symbolic names of the queue structure that demanded this activity are placed in the data items for QUEUE, SUB-QUEUE-1, SUB-QUEUE-2, and SUB-QUEUE-3 of the CD input area containing the INITIAL clause prior to the execution of the first Procedure Division statement.

The execution of a subsequent RECEIVE statement naming the same contents of the QUEUE, SUB-QUEUE-1, SUB-QUEUE-2, and SUB-QUEUE-3 data items returns the actual message that caused the program to be scheduled. Only at that time is the remainder of the area updated.

An INITIAL CD input area can be used in all cases where an input CD area needs to be referenced. Its fields are used and updated in the same way as those of any other CD input area.

Symbolic Queues

The symbolic queues contain four names that are used to communicate between the MCS and a COBOL program. The symbolic queue names are SYMBOLIC QUEUE, SYMBOLIC SUB-QUEUE-1, SYMBOLIC SUB-QUEUE-2, and SYMBOLIC SUB-QUEUE-3. The data-names given to the symbolic names of a COBOL program must be defined in the application definition for the MCS application that communicates with the COBOL program. An application definition uses the Application Definition Language to define and describe application components to MCS. (Refer to the Message Control System reference manual.)

The characters used to form a symbolic queue name are A-Z, 0-9, and - (hyphen). The first character must be a letter and the last character must not be a hyphen. The maximum length is 12 characters. Sybmbolic queue names that are less than 12 characters are left justified and space filled.

The symbolic queue can be either simple or compound. The simple queue is a queue that does not contain sub-queues. A compound queue contains sub-queues. The order of the sub-queues of a compound queue determines the queue selection ordering used when a COBOL program executes a RECEIVE statement.

To access a simple queue, one to four symbolic queue names must be entered (depending on the number of levels that identify a given simple queue) in the appropriate symbolic queue name fields. If all symbolic queue name fields are not needed, or if a compound queue is accessed, all unused symbolic queue name fields must contain spaces during execution of the communication facility statement that references that communication description area. (See Queue Hierarchy in the Message Control System reference manual.)

MESSAGE DATE

The MESSAGE DATE is the date that MCS acquires the final portion of the message transferred to the COBOL program that executed the RECEIVE statement. The MESSAGE DATE is updated with the execution of the RECEIVE statement if message text is transferred. The format for MESSAGE DATE is yymmdd (year, month, day).

MESSAGE TIME

The MESSAGE TIME is the time that MCS acquires the final portion of the message transferred to the COBOL program that executed the RECEIVE statement. The MESSAGE TIME is updated with the execution of the RECEIVE statement if message text is transferred. The format for MESSAGE TIME is hhmmsstt (hours, minutes, seconds, hundredths of a second). The value of hh can be 0 through 23; mm and ss can be 0 through 59; tt must be zeros.

SYMBOLIC SOURCE

The SYMBOLIC SOURCE is the symbolic name that MCS associates with the source of the message being transferred when a COBOL program executes a RECEIVE statement. Execution of the ENABLE INPUT or DISABLE INPUT statements, along with the keyword TERMINAL, can establish or break the logical paths between the source of the message being transferred and all input queues receiving messages from the source.

Symbolic source names that are less than 12 characters long are left-justified and space-filled. Symbolic source names used in an application must be defined in the application definition. (See the Message Control System reference manual.) The association of a symbolic name with a given name is part of the definition.

TEXT LENGTH

The TEXT LENGTH is the number of message characters transferred to the COBOL program with the RECEIVE statement. The TEXT LENGTH is updated with a successful execution of the RECEIVE statement. If no message characters are transferred and control is returned to the program, TEXT LENGTH is set to zero.

END KEY

The END KEY indicates the part of a message that has been transferred. When a RECEIVE statement with the SEGMENT phrase is executed, the END KEY assumes one of the following values:

A 0 indicates that less than a message segment has been transferred to the COBOL program that executed the RECEIVE statement.

A 1, which indicates a complete segment or an end-of-segment indicator (ESI), terminates the message text transferred.

A 2, which indicates an end-of-message indicator (EMI), terminates the message text transferred. An EMI implies an ESI.

A 3, which indicates an end-of-group indicator (EGI), terminates the message text transferred. An EGI implies an EMI and an ESI.

When a RECEIVE statement is executed with the MESSAGE phrase, the END KEY assumes the values 0, 2, or 3.

STATUS KEY

The STATUS KEY indicates if any exception conditions were detected during execution of an MCS statement that references a CD input area. Table 4-10 indicates the codes that the STATUS KEY can assume for particular exception conditions that occur with a statement execution.

TABLE 4-10. INPUT CD AREA STATUS KEY CODES

Column Legend							Exception Conditions
1	2	3	4	5	6	7	
X	X	X	X	X	X	00	The communication facility statement executed successfully.
		X	X	X	X	15	One or more of the queue/source paths were already disabled or enabled. Action applied to other paths.
X	X	X		X		20	One or more symbolic queue names invalid or unknown. No action was taken while executing the communication facility statement.
			X		X	20	Symbolic source name invalid or unknown. No action was taken while executing the communication facility statement.
		X	X	X	X	40	Password invalid. No action was taken while executing the communication facility statement.
X	X	X	X	X	X	90	System error. The results of the communication facility statement are unpredictable and the program should terminate processing.
X	X	X	X	X	X	91	MCS is not running or MCS is not defined as a System Control Point. No action was taken while executing the communication facility statement.
X	X	X	X	X	X	92	No valid application name parameter specified on the program call statement or the specified application is not running. This code can only occur if the program was not initiated by MCS. No action was taken while executing the communication facility statement.
X	X	X	X	X	X	93	The program-name of the program is unknown or a program with the same program-name is already established with MCS. No action was taken while executing the communication facility statement.
X	X	X	X	X	X	94	MCS or the application is shutting down. No action was taken while executing the communication facility statement.
X	X	X	X	X	X	96	Sufficient resources (e.g., central memory) are not available to satisfy the request. No action was taken while executing the communication facility statement. The program request can be repeated.
X	X	X	X	X	X	97	MCS encountered a CIO error when accessing a mass storage queue. The results of the request are unpredictable and the program should terminate processing. Further attempts to access the queue by any part of the application results in the application being closed down by MCS.

COLUMN LEGEND

Column 1 - RECEIVE
Column 2 - ACCEPT MESSAGE COUNT
Column 3 - DISABLE INPUT (without the keyword TERMINAL)
Column 4 - DISABLE INPUT TERMINAL
Column 5 - ENABLE INPUT (without the keyword TERMINAL)
Column 6 - ENABLE INPUT TERMINAL
Column 7 - STATUS KEY codes

MESSAGE COUNT

The MESSAGE COUNT is the number of complete messages that exist in the message queues. The number represents the complete messages enqueued by MCS into a simple or compound queue when an ACCEPT MESSAGE COUNT statement is successfully executed.

If a compound queue is specified, the MESSAGE COUNT indicates the number of complete messages in all simple queues that are subordinate to the specified compound queue.

Usage of CD Input Areas

Table 4-11 summarizes the usage of the CD input areas. The symbols I, +, *, $, U, O, and a blank indicate how the field must be used or updated.

TABLE 4-11. USAGE CD INPUT AREA

Fields	Column Legend						
	1	2	3	4	5	6	7
SYMBOLIC QUEUE	I	I	I		I		$
SYMBOLIC SUB-QUEUE-1	*	+	+		+		$
SYMBOLIC SUB-QUEUE-2	*	+	+		+		$
SYMBOLIC SUB-QUEUE-3	*	+	+		+		$
MESSAGE DATE	O						
MESSAGE TIME	O						
SYMBOLIC SOURCE	O			I		I	
TEXT LENGTH	O						
END KEY	O						
STATUS KEY	U	U	U	U	U	U	
MESSAGE COUNT		O					

COLUMN LEGEND

Column 1 - RECEIVE
Column 2 - ACCEPT MESSAGE COUNT
Column 3 - DISABLE INPUT
(without the keyword TERMINAL)
Column 4 - DISABLE INPUT TERMINAL
Column 5 - ENABLE INPUT
(without the keyword TERMINAL)
Column 6 - ENABLE INPUT TERMINAL
Column 7 - INITIAL input CD areas at the beginning of the Procedure Division

An I indicates that a symbolic name must be entered in that input area before executing the communication facility statement that references the area.

A + indicates that either a symbolic queue name or spaces must be entered in the area before executing the communication facility statement. Entering a queue name or spaces is dependent on whether a compound or simple queue is referenced. For a simple queue, the choice depends on the number of levels required to identify the simple queue in a queue hierarchy.

An * indicates that either a symbolic queue name or spaces must be entered before executing the communication facility statement. If message text was transferred and a compound queue was referenced, MCS enters symbolic queue names in as many of the space-filled * areas as necessary to identify the simple queue containing the returned message.

A $ indicates that either the field is filled with spaces or a symbolic queue name before executing the first Procedure Division statement of the program. The following criteria determine whether a symbolic queue name is entered into a field:

If the job stream containing the program was scheduled for execution by MCS.

If the threshold of the queue was exceeded.

If an INITIAL input CD preceded the program that used MCS.

If this is the first program in the job stream containing an INITIAL input CD area.

The number of levels necessary to specify the simple or compound queue whose message count caused the job to be scheduled by MCS.

A U indicates that MCS updates the area during execution of the communication facility statement, regardless of any exception conditions.

An O indicates that the area is conditionally updated by MCS during execution of the communication facility statement.

The absence of an entry indicates that the area is not used or updated.

CD Output

The CD output area (figure 4-25) specifies the CD level indicator, a cd-name (communication description name), and a series of independent clauses. The independent clauses include the fields destination count, text length, status key, destination table occurs, indexed, error key, symbolic destination, and the user-defined names of these fields. These clauses are used to execute the statements SEND, PURGE, DISABLE OUTPUT, and ENABLE OUTPUT. A COBOL program that sends messages must define at least one CD output area.

DESTINATION COUNT

The DESTINATION COUNT specifies the number of destinations that applies to a specific communication facility statement. The DESTINATION COUNT must have at least a value of one, and must not exceed integer-2. The destination order for the communication facility statement is determined by the content of the SYMBOLIC DESTINATION; the ordering continues up to and including the number of destinations given by the content of the DESTINATION COUNT.

```
Format 2

CD   cd-name;        FOR    OUTPUT

  [ ;  DESTINATION  COUNT         IS          data-name-1 ]

  [ ;  TEXT LENGTH                IS          data-name-2 ]

  [ ;  STATUS KEY                 IS          data-name-3 ]

       [ ;  DESTINATION TABLE OCCURS    integer-2       TIMES

         [ ;  INDEXED     BY    index-name-1    [ , index-name-2 ]   ... ]]

  [ ;  ERROR KEY                              IS          data-name-4 ]

  [ ;  SYMBOLIC DESTINATION                   IS          data-name-5 ]   .
```

Figure 4-25. CD Output Area

TEXT LENGTH

The TEXT LENGTH indicates to MCS the number of character positions transferred during execution of the SEND statement. Before executing a SEND statement, the number of leftmost character positions of the message to be transferred must be entered in the TEXT LENGTH.

STATUS KEY

The STATUS KEY indicates any exception conditions detected during execution of a communication facility statement that references a CD output area. Table 4-12 indicates the codes that the STATUS KEY can assume for particular exception conditions that occur with the execution of each statement.

ERROR KEY

The ERROR KEY indicates any exception condition detected for a specific destination. MCS updates the ERROR KEY as part of the execution of a communication facility statement, whether or not any exception conditions were detected. Table 4-13 indicates the codes that the ERROR KEY can assume for particular exception conditions that occur with the execution of each statement.

SYMBOLIC DESTINATION

The SYMBOLIC DESTINATION indicates to MCS the destination that applies to a communication facility statement. The symbolic name that MCS associates with each destination must be entered in the SYMBOLIC DESTINATION.

Symbolic destination names that are less than 12 characters long must be left justified and space filled. Symbolic destination names and the associated given destinations used in an application must be defined in the application definition language. (See the Message Control System reference manual.) Symbolic destinations can be terminals, queues, journals, or a broadcast list. The ERROR KEY field for a broadcast list cannot completely describe for all list members the results of the statement executed.

Usage of CD Output Areas

Table 4-14 summarizes the usage of the CD output areas. The symbols I, U, O, and a blank indicate how the area must be used or updated.

An I indicates that the field shown on that line must be set before executing the communication facility statement that references the field.

A U indicates that the field is always updated by MCS, regardless of any exception conditions detected during execution of the communication facility statement.

An O indicates that the field is conditionally updated by MCS during execution of the communication facility statement.

The absence of an entry indicates that the field is not used or updated.

TABLE 4-12. OUTPUT CD AREA STATUS KEY CODES

Column Legend					Exception Conditions
1	2	3	4	5	
X	X	X	X	00	The communication facility statement executed successfully.
X	X			10	The communication facility statement executed successfully for all destinations referenced; however, one or more of them are currently disabled.
		X	X	15	One or more destinations already disabled or enabled. Action applied to other destinations.
X	X	X	X	20	One or more symbolic destination names invalid or unknown. Action successfully completed for all valid destinations.
X	X	X	X	30	DESTINATION COUNT invalid or exceeds the maximum value possible. No action was taken while executing the communication facility statement.
		X	X	40	Password invalid for one or more destinations referenced. Disabled or enabled action applied to other destinations.
X				50	TEXT LENGTH invalid or exceeds the maximum value possible. No action was taken while executing the SEND statement.
X				60	A portion of a segment or message is to be sent, but no message area is referenced and/or TEXT LENGTH is set to zero. No action was taken while executing the SEND statement.
	X			70	One or more destinations do not have partial messages associated with them. PURGE statement successfully completed for other destinations.
X	X	X	X	80	Two or more of the conditions designated by the STATUS KEY codes 10, 15, 20, 40, 70 and 95 have occurred. Action successfully completed for those destinations for which no exception condition exists.
X	X	X	X	90	System error. The results of the communication facility statement are unpredictable and the program should terminate processing.
X	X	X	X	91	MCS is not running or MCS is not defined as a System Control Point. No action was taken while executing the communication facility statement.
X	X	X	X	92	No valid application name parameter specified on the program call statement or the specified application is not running. This code can only occur if the program was not initiated by MCS. No action was taken while executing the communication facility statement.
X	X	X	X	93	The program-name of the program is unknown or a program with the same program-name is already established with MCS. No action was taken while executing the communication facility statement.
X	X	X	X	94	MCS or the application is shutting down. No action was taken while executing the communication facility statement.
X				95	Output queue threshold for one or more destinations exceeded. SEND statement successfully completed for other destinations.
X	X	X	X	96	Sufficient resources (e.g., central memory) are not available to satisfy the request. No action was taken while executing the communication facility statement. The program may repeat the request.
X	X	X	X	97	MCS encountered a CIO error when accessing a mass storage queue. The results of the request are unpredictable and the program should terminate processing.

<u>COLUMN LEGEND</u>

Column 1 - SEND	Column 3 - DISABLE OUTPUT	Column 5 - STATUS KEY
Column 2 - PURGE	Column 4 - ENABLE OUTPUT	

TABLE 4-13. OUTPUT CD AREA ERROR KEY CODES

Column Legend					Exception Conditions
1	2	3	4	5	
X	X	X	X	0	No exception condition for this destination.
X	X	X	X	1	Symbolic destination name invalid or unknown.
X	X			2	Destination or associated output queue is disabled.
		X	X	3	Password invalid for this destination.
	X			4	No partial message associated with this destination.
		X	X	5	Destination already is disabled or enabled.
X				6	Output queue threshold for this destination exceeded.
X				A	No queue is defined for this destination.

COLUMN LEGEND

Column 1 - SEND
Column 2 - PURGE
Column 3 - DISABLE OUTPUT
Column 4 - ENABLE OUTPUT
Column 5 - ERROR KEY

TABLE 4-14. USAGE CD OUTPUT AREA

Fields	Column Legend			
	1	2	3	4
DESTINATION COUNT	I	I	I	I
TEXT LENGTH	I			
STATUS KEY	U	U	U	U
ERROR KEY	0	0	0	0
SYMBOLIC DESTINATION	I	I	I	I

COLUMN LEGEND

Column 1 - SEND
Column 2 - PURGE
Column 3 - DISABLE OUTPUT
Column 4 - ENABLE OUTPUT

PROCEDURE DIVISION

Procedure Division is required in every source program to specify the processing that is to take place. It is the last division in the program. Program execution begins with the first statement of this division, excluding any declaratives. Statements are then executed in the order in which they are presented for compilation, except where the statements themselves cause another order.

DIVISION STRUCTURE

The division consists of the division header and the body structure. When the program is segmented or declaratives are included, the lines in the division must be grouped into sections. Sections are optional when declaratives are omitted and the program is not segmented. Division structure is shown in figure 5-1.

DIVISION HEADER

The division header must appear on a separate line, beginning in area A, and must be terminated by a separator period. The division header format is:

```
PROCEDURE DIVISION  [USING data-name-1
     [, data-name-2]   . . . .]
```

The USING phrase is applicable only in a subprogram that is to execute under control of a CALL statement that also contains a USING phrase. The order of the operands in the header must be the same order as the operands in the CALL statement. A data-name must not appear more than once in the division header, although it can appear more than once in the CALL statement. See section 15, Inter-Program Communication Facility.

DECLARATIVES

Declaratives are bounded by the keywords DECLARATIVES and END DECLARATIVES. These words must begin in area A, appear on a separate line, and terminate with the separator period.

The declaratives body consists of one or more sections. The sentence following the declaratives section header must be a declarative-sentence consisting of a USE statement terminated by a period separator. This statement defines the conditions under which any immediately following paragraphs are executed. One or more paragraphs should follow all USE statements.

The USE statement has several formats, each concerned with a different type of declarative. See the USE statement in this section.

Declaratives in a segmented program are discussed in section 8, Segmentation Facility.

PROCEDURES

A procedure consists either of a paragraph or group of successive paragraphs or of a section or group of successive sections. If one paragraph in the division is in a section, then all paragraphs must be in sections.

Format 1

```
[DECLARATIVES.
{ section-name SECTION  [segment-number].  declarative-sentence.
[ paragraph-name.  [sentence] . . .] . . .} . . .
END DECLARATIVES.]
{ section-name SECTION  [segment-number].
[ paragraph-name.  [sentence] . . .] . . .} . . .
```

Format 2

```
{ paragraph-name.  [sentence] . . .} . . .
```

Figure 5-1. Procedure Division Skeleton

A procedure-name is used to refer to a paragraph or section in the source program. It consists of a paragraph-name or a section-name.

Section-names must be unique within the division.

Paragraph-names need be unique only within a given section. References to a paragraph-name can be made unique by qualifying the paragraph-name with the name of the section in which it appears; the keyword SECTION is not included in the qualifier. Paragraph-names need not be qualified when referred to from within the same section.

Both section-name and paragraph-name can be composed of 30 characters selected from the letters A through Z, the digits 0 through 9, and the hyphen. A letter is not required in either a section-name or paragraph-name.

Sections

A section begins with the section header that begins in area A and is terminated by the separator period. A section header has the following format:

section-name <u>SECTION</u> [segment-number] .

The segment-number is described in section 8, Segmentation Facility.

A section ends with the appearance of another section header, the end of the division, or the end of the declaratives.

Zero, one, or more successive paragraphs can be contained in a section.

Paragraphs

A paragraph begins with the paragraph-name, which begins in area A and is terminated by a separator period. Paragraph-name can, but need not, appear on a separate line.

A paragraph ends with the appearance of another paragraph-name, a section header, the end of the division, or the end of the declaratives.

Zero, one, or more successive sentences can be contained in a paragraph.

STATEMENTS AND SENTENCES

A sentence is a series of one or more statements that is terminated by a separator period. Statements can, but need not, be written as individual sentences unless specifically noted in the statement discussion.

A sentence is categorized by the statements it contains. Three types of statements and sentences are:

Conditional

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the executing program is dependent on this truth value. A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by a period.

Compiler-directing

A compiler-directing statement consists of a compiler-directing verb and its operands. A compiler-directing sentence is a single compiler-directing statement terminated by a period.

Imperative

An imperative statement indicates a specific unconditional action is to be taken by the executing program. It can consist of a consecutive sequence either of imperative statements terminated by a period or of an ELSE phrase associated with a previous IF statement or of a WHEN phrase associated with a previous SEARCH statement. An imperative sentence is an imperative statement terminated by a period.

An imperative statement can also be a delimited scope statement. A delimited scope statement is any statement that includes its explicit scope terminator. The explicit scope terminators are END-IF, END-PERFORM, and END-SEARCH.

Table 5-1 shows the type of statement for each of the verbs used in the Procedure Division.

PROCEDURE DIVISION STATEMENTS

All statements of the Procedure Division are presented alphabetically in this section.

ACCEPT STATEMENT

The ACCEPT statement (figure 5-2) makes data available to the specified data item. Statement format determines the source and extent of the data.

Format 1 reads low volume data from a file.

Format 2 sets a data item with the value of a conceptual date or time.

Format 3 returns the number of messages in an MCS queue; can be used only under NOS.

In the statement, identifier specifies the item to receive data.

Format 1

<u>ACCEPT</u> identifier [<u>FROM</u> mnemonic-name]

Format 2

$$\textbf{\underline{ACCEPT} identifier FROM} \left\{\begin{array}{l}\underline{\text{DATE}}\\ \underline{\text{DAY}}\\ \underline{\text{DAY-OF-WEEK}}\\ \underline{\text{TIME}}\end{array}\right\}$$

Format 3

<u>ACCEPT</u> cd-name MESSAGE <u>COUNT</u>

Figure 5-2. ACCEPT Statement Format

TABLE 5-1. STATEMENT CLASSIFICATION

Statement and Phrase	Type of Statement		
	Condi-tional	Compiler Directing	Imper-ative
ACCEPT			X
ADD:			
with SIZE ERROR	X		
without SIZE ERROR			X
ALTER			X
CALL:			
with ON OVERFLOW	X		
without ON OVERFLOW			X
CANCEL			X
CLOSE			X
COMPUTE:			
with SIZE ERROR	X		
without SIZE ERROR			X
CONTINUE			X
COPY			
DELETE:		X	
with INVALID KEY	X		
without INVALID KEY			X
DISABLE			X
DISPLAY			X
DIVIDE:			
with SIZE ERROR	X		
without SIZE ERROR			X
ENABLE			X
ENTER			X
EXIT			X
GENERATE			X
GO TO			X
IF:			
with END-IF			X
without END-IF	X		
INITIALIZE			X
INITIATE			X
INSPECT			X
MERGE			X
MOVE			X
MULTIPLY:			
with SIZE ERROR	X		
without SIZE ERROR			X
OPEN			X
PERFORM:			
with END-PERFORM			X
without END-PERFORM			X
PURGE			X
READ:			
with INVALID KEY	X		
without INVALID KEY			X
with AT END	X		
without AT END			X
RECEIVE			
with NO DATA	X		
without NO DATA			X
RELEASE			X
REPLACE		X	
RETURN	X		
REWRITE:			
with INVALID KEY	X		
without INVALID KEY			X
SEARCH:			
with END-SEARCH			X
without END-SEARCH	X		
SEND			X
SET			X
SORT			X
START:			
with INVALID KEY	X		
without INVALID KEY			X
STOP			X
STRING:			
with ON OVERFLOW	X		
without ON OVERFLOW			X
SUBTRACT:			
with SIZE ERROR	X		
without SIZE ERROR			X
SUPPRESS			X
TERMINATE			X
UNSTRING:			
with ON OVERFLOW	X		
without ON OVERFLOW			X
USE		X	
WRITE:			
with INVALID KEY	X		
without INVALID KEY			X
with EOP	X		
without EOP			X

Format 1 ACCEPT

Format 1 normally specifies that data is to be accepted from a terminal or the central site console, or from a file connected to a terminal into which data will be entered as the program executes. The transfer of data is a straight character transfer; no conversion of data takes place. Consequently, if the usage of identifier is not DISPLAY, unexpected results can occur. (Accepting data from the console is inefficient in a multiprogramming environment and should be avoided.) The file from which data is accepted is assumed to have a sequential format with C type blocking and Z type records.

Mnemonic-name specifies the source of the data. It must be associated with an implementor-name in the SPECIAL-NAMES paragraph of the Environment Division. Mnemonic-name must not be associated with a file named in an FD entry, except for the implementor-name "INPUT". When the FROM phrase is omitted, data is read from the system file INPUT.

The maximum number of characters that can be accepted from a single input record depends on the implementor-name and whether or not the implementor-name refers to a connected file of an interactive terminal:

"TERMINAL" or connected file	identifier character size or 300, whichever is smaller
CONSOLE	40
Other	80, unless changed by a FILE control statement

When the data exceeds the size of the identifier, characters are truncated on the right after the receiving identifier item is filled.

When data is being accepted from an existing file, either 80 characters or the number of characters specified by the FL parameter of the FILE control statement are read. If the size of the identifier exceeds the size of the input record, multiple records are read until the identifier is filled. If the identifier size is not an even multiple of record size, the last record is truncated on the right.

Data being entered interactively through a terminal or console need not be expanded to match the size of the receiving data item. The data accepted with a single ACCEPT statement is terminated by a carriage return. The identifier item is blank filled when terminal input data does not fill the item.

When an ACCEPT identifier FROM CONSOLE statement is executed on NOS, the following message appears, but does not flash, in the B display on the central site console:

TYPE N.CFO + COBOL INPUT

The operator should be expecting the message in order to take any action. The operator must type n.CFO and then the input data; the n is the control point ordinal.

When an ACCEPT identifier FROM CONSOLE statement is executed on NOS/BE, the following message flashes in the B display on the central site console:

TYPE J.CFO + COBOL INPUT

The operator must type j.CFO and then the input data; the j is the JDT ordinal appearing on the left in the B display.

When the data is accepted from an interactive terminal, a question mark is output at the terminal before the response is expected. If the previous operation at the terminal was a DISPLAY WITH NO ADVANCING statement, a question mark is not output and the response can be entered from the current position.

When the character size of the identifier exceeds 300 characters or the FL parameter of the FILE control statement, each portion of data terminated by a carriage return represents a multiple of 300 or FL characters. The data is followed by a question mark being output to the terminal until enough multiples of 300 or FL have been received to fill the identifier. Any excess data accepted by this method is truncated from the right.

Any end-of-file or partition boundary encountered on a noninteractive file causes a diagnostic to be issued and the run to be terminated. All spaces are returned for an end-of-file or a partition boundary on an interactive file. Partition boundaries are defined by CYBER Record Manager.

Format 2 ACCEPT

Format 2 works with conceptual data items that do not have a description within the program. The items behave, however, as if they were described as elementary unsigned numeric integer data items with the number of digits indicated below.

DATE

Six digits yymmdd representing the year of the century, month of the year, and day of the month.

DAY

Five digits yyddd representing the year of the century and day of the year.

DAY-OF-WEEK

One digit d representing the day of the week. The range of values allowed is 1 through 7 corresponding to Monday through Sunday, respectively.

TIME

Eight digits hhmmsscc representing the hour, minute, second, and centisecond. The two digits representing the centisecond are always set to 00. The hour is based on elapsed time after midnight for a 24-hour clock. The range of values allowed is 00000000 through 23595900.

Examples of conceptual data item format:

Item	Example	Format Accepted
DATE	July 1, 1968	680701
DAY	July 1, 1968	68183
DAY-OF-WEEK	Thursday	4
TIME	2:41 p.m.	14410000

The conceptual data item is moved to the identifier field as if a MOVE instruction were executed.

Format 3 ACCEPT

Format 3 causes the number of complete MCS messages existing in a simple or compound queue to be returned. When an ACCEPT MESSAGE COUNT statement is executed, the communication description (CD) input area referenced by cd-name must contain one or more symbolic queue names. These names identify the simple or compound input queue that contains the message count to be returned. If the statement executes successfully, this message count is returned in the MESSAGE COUNT field of the CD input area. The message count does not include any messages in the simple or compound queue that are incomplete. Incomplete messages include messages that have not been completely acquired by MCS and messages that have partially been returned to programs executing RECEIVE statements.

The message count is accurate at the time MCS updates the MESSAGE COUNT field. The count might not be accurate if the queue name referenced is not disabled at the time that the statement following the ACCEPT MESSAGE COUNT is executed. Execution of the ACCEPT MESSAGE COUNT statement also updates the STATUS KEY field of the CD input area.

ADD STATEMENT

The ADD statement (figure 5-3) adds two or more elementary numeric data items. The data item that receives the result of the addition depends on the format used:

Format 1 stores the result in the last identifier used in the addition.

Format 2 stores the result in an identifier that was not used in the addition.

Format 3 adds elementary items in one group identifier to any corresponding items in another group identifier and stores the result in the second identifier used for the addition.

The composite size of identifiers and/or literals is limited to 18 digits. The composite size is determined by superimposing data items on their respective decimal points in a hypothetical data item. The items used in composite size calculations depend on the format:

Format 1 uses all identifiers and/or literals specified.

Format 2 uses all operands, but not the identifiers into which the result is stored.

Format 3 separately uses each pair of corresponding data items.

Format 1 ADD

Format 1 ADD occurs as follows: The value of literal-1 or identifier-1 is added to the value of all literal-2 or identifier-2 specified. The sum is then added to the current value of identifier-m and stored as the new value of identifier-m. If identifier-n is specified, the sum of all literals or identifiers specified before the keyword TO is added to, and the result stored in, each identifier-n specified. All identifiers of format 1 must be elementary numeric items.

ROUNDED Phrase

The ROUNDED phrase affects only the data item in which the result of a computation is stored; it affects the result only when that data item is described with a symbol P, V, or a decimal point. Computations are performed in

```
Format 1

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

          [; ON SIZE ERROR imperative-statement]

Format 2

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

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

Format 3

     ADD {CORRESPONDING} identifier-1 TO identifier-2  [ROUNDED] [, identifier-3  [ROUNDED] . . .]
         {CORR         }

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

Figure 5-3. ADD Statement Format

locations known only to the compiler. At the end of the computation, the result is moved to the data item or items specified in the arithmetic statement.

The result of the computation is aligned with the decimal point of the result data item before rounding is considered. If the ROUNDED phrase is omitted, truncation occurs at the right when the number of places in the fraction of the result exceeds the number of fraction places provided in the result data item.

If the ROUNDED phrase is specified, the last digit of the absolute value of the result data item is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

SIZE ERROR Phrase

The SIZE ERROR phrase affects only a data item in which the result of a computation is stored. Computations are performed in locations known only to the compiler. At the end of the computation, the result is moved to the data item or items specified in the arithmetic statement.

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the data item, a size error occurs for that item. Each result data item is treated separately.

When the SIZE ERROR phrase is omitted, the value of any data item with a size error is undefined. Exponent overflow or underflow causes program termination, unless a MODE control statement is in effect for these conditions. All other result data items are unaffected by the error condition.

When the SIZE ERROR phrase is specified, the value of any data item with a size error is not altered from its previous value. All other result data items are unaffected by the error condition. After all result data items have been considered, the imperative-statement of the phrase executes. The program is responsible for determining which result caused the size error and execution of the imperative-statement in the phrase.

If the ROUNDED phrase is specified, rounding occurs before checking for size error. Consequently, the ROUNDED phrase precludes the possibility of a size error to the right, but not to the left, of a decimal point.

A size error always exists when any of the following conditions is detected for any intermediate or final result during an ADD or other arithmetic statement operation:

Division by zero.

Zero is raised to zero or negative power.

Number less than zero is raised to a noninteger power.

Floating point exponent overflow or underflow.

Format 2 ADD

Format 2 ADD occurs as follows: The value of identifier-1 or literal-1 is added to the value of identifier-2 or literal-2, then their sum is added to all literal-3 or identifier-3 specified. The result of the addition is stored as the new value of identifier-m. If identifier-n is specified, the sum of all literals or identifiers specified before the keyword GIVING is also stored as the new value of each identifier-n specified. In format 2, all identifiers or literals to be added must be elementary numeric items; identifier-m and identifier-n can be elementary numeric or elementary numeric edited items.

The ROUNDED phrase and the SIZE ERROR phrase are the same as for format 1 ADD.

Format 3 ADD

Format 3 adds items in identifier-1 to, and stores the results in, corresponding items in identifier-2. If identifier-3 is specified, items in identifier-1 are added to, and the results stored in, corresponding items in each identifier-3 specified. All identifiers in format 3 must be group items; only corresponding elementary numeric items are added. Identifiers must not be reference modified.

See format 2 MOVE statement for the definition of corresponding data items. The ROUNDED phrase and the SIZE ERROR phrase are the same as for format 1 ADD.

ALTER STATEMENT

The ALTER statement (figure 5-4) modifies a GO TO statement so that subsequent execution transfers control to a different procedure.

```
ALTER procedure-name-1 TO

        [PROCEED TO] procedure-name-2

    [, procedure-name-3 TO

        [PROCEED TO] procedure-name-4] . . .
```

Figure 5-4. ALTER Statement Format

Procedure-name-1, which is the procedure to be altered, must be the name of a paragraph containing only a format 1 GO TO statement. Procedure-name-2, which is the procedure to be substituted in the GO TO statement, must be the name of a paragraph or section in the Procedure Division. Any procedure-name-3 is treated the same as procedure-name-1; procedure-name-4 is treated as procedure-name-2. The same procedure-name must not be altered more than once in the same ALTER statement.

The words PROCEED TO can be included for clarity. They do not change the meaning of the statement.

NOTE

Because of anticipated changes in this product, use of the ALTER statement is not recommended. For guidelines, see appendix F.

See section 8, Segmentation Facility, for use of ALTER in a segmented program.

CALL STATEMENT

The CALL statement transfers control to a COBOL subprogram. See section 15, Inter-Program Communication Facility.

CANCEL STATEMENT

The CANCEL statement removes a called program from memory. See section 15, Inter-Program Communication Facility.

CLOSE STATEMENT

The format 1 CLOSE statement (figure 5-5) has several functions depending on the phrases used. It can be used to:

Terminate processing at the end of an input or output file

Terminate processing before the end of an input file

Terminate processing of a reel of a sequential tape file

Cause an END REEL checkpoint of a sequential mass storage file

```
Format 1

                          [ {REEL} [WITH NO REWIND] ]
CLOSE file-name-1         [ {UNIT} [FOR REMOVAL   ] ]
                          [                         ]
                          [ WITH   {NO REWIND}      ]
                          [        {LOCK     }      ]

  [                 [ {REEL} [WITH NO REWIND] ] ]
  [ ,file-name-2    [ {UNIT} [FOR REMOVAL   ] ] ] . . .
  [                 [                         ] ]
  [                 [ WITH  {NO REWIND}       ] ]
  [                 [       {LOCK     }       ] ]
```

Figure 5-5. CLOSE Statement Format 1

The format 2 CLOSE statement terminates processing for all files associated with a relation specified in a sub-schema. See section 14, Sub-Schema Facility, for the format 2 CLOSE statement.

CLOSE inhibits the use of any subsequent input-output statements. If CLOSE is omitted, all open files are closed when a STOP RUN statement is executed as if a CLOSE statement with no optional phrases was executed; a warning message is issued to the dayfile. It might not be possible to close some files due to the action of CANCEL or overlays; an appropriate diagnostic is issued to the dayfile.

If the OPTIONAL phrase of the SELECT clause has been specified in the FILE-CONTROL paragraph of the Environment Division and the file is not present, the standard end-of-file processing is not performed for the file.

In the statement, file-name-1 and file-name-2 specify the files to be closed. They can be files described in a sub-schema.

The effects of CLOSE depend on the file organization.

For indexed, direct, and actual-key files, processing terminates and AAM updates the internal table that is a part of each of these files. If the file subsequently is reopened, position is at the beginning of the first record in the file.

For a relative and word-address file, processing terminates. A partition boundary exists in the file at the end of the last record. The boundary can be overwritten by adding new records at the end of the file.

For sequential files, processing depends on the device on which the file resides and the presence or absence of the REEL or NO REWIND phrase. If NO REWIND is not specified, the file is rewound before it is closed.

In all cases except when the REEL or LOCK phrase is specified, the file is closed with the CF field of the file information table set to DET. This causes buffer space to be released, along with any CYBER Record Manager capsules particular to that file. This action should be avoided if the file is to be reopened immediately. To prevent this unnecessary use of system resources, the utility routine C.FILE can be entered in the following manner:

```
ENTER "C.FILE" USING file-name, "CF=R".
```

If this statement is executed prior to the close, the buffer space is not released when the close is executed. For more information, see the discussion of utility routines in section 15, Inter-Program Communication Facility.

NO REWIND Phrase

The WITH NO REWIND phrase inhibits the rewind that is a part of CLOSE processing. It is valid only for sequential files. The phrase is ignored for all other file organizations.

LOCK Phrase

The WITH LOCK phrase closes the file and prevents it from being reopened within the job step currently executing. An operating system UNLOAD function occurs when this phrase is used, which causes any local mass storage file to be discarded. An attempt to reopen the file within the same program aborts the program.

REEL/UNIT Phrase

The REEL or UNIT phrase is valid only for sequential files. The words REEL and UNIT are synonymous and interchangeable.

If the REEL phrase is omitted, file processing terminates. An end-of-partition boundary is written to an output file. Any labels for either an input file or output file are processed as appropriate. The file is positioned in accordance with other phrases in the statement.

If the phrase is used with a sequential file on mass storage, any checkpoint established by the END REEL phrase of the RERUN clause in the I-O-CONTROL paragraph takes place. Otherwise, this phrase of CLOSE has no effect on file positioning or processing.

If the phrase is used with a sequential file on tape, any END REEL checkpoint takes place. Label rocessing occurs, if appropriate, at the end of the reel being closed. The operating system procedures for reel swap then occur. Label processing resumes at the beginning of the next reel. No end-of-partition is written on an output file when the REEL phrase is used.

A reel close is never required. If the physical delimiter of a reel is encountered during file processing, reels are swapped automatically under operating system control without the program being aware of the swap. Although the current reel is rewound and physically unloaded during the swap, the program can force a prior reel to be remounted after the file is closed by execution of an OPEN statement with the REWIND option.

Judicious use of the rewind and no rewind capabilities of the OPEN and CLOSE statements and the job control statements that designate the tape units required for a job can improve the efficiency of the physical handling of a known multireel file. Automatic swapping occurs for both a single multireel file and for a multifile multireel set, the only difference being that label processing always occurs for a multifile set.

The WITH NO REWIND phrase of CLOSE REEL inhibits the rewind and physical unload that would otherwise occur.

The FOR REMOVAL phrase is documentary only.

COMPUTE STATEMENT

The COMPUTE statement (figure 5-6) evaluates an arithmetic or boolean expression and sets the result in one or more data items.

```
Format 1

    COMPUTE identifier-1  [ROUNDED]

          [, identifier-2  [ROUNDED]]  . . .

    { FROM   }
    { =      }  arithmetic-expression
    { EQUALS }

          [; ON SIZE ERROR imperative-statement]


Format 2

    COMPUTE {,identifier-3} . . .

       { FROM   }
       { =      }  boolean-expression
       { EQUALS }
```

Figure 5-6. COMPUTE Statement Format

In format 1, the result of expression evaluation is stored in identifier-1 and also in any identifier-2 specified. These identifiers must be elementary numeric or elementary numeric edited items.

The words FROM and EQUALS are equivalent to each other and to the symbol = ; they can be used interchangeably. Arithmetic-expression can be a single identifier or literal, as well as an expression.

The ROUNDED phrase and the SIZE ERROR phrase are discussed with the ADD statement in this section. SIZE ERROR applies to all intermediate results as well as the final results. Additional information about arithmetic expressions and operations is in section 1.

In format 2, identifier must reference a boolean data item. The result of the boolean expression evaluation is stored in identifier-3. Identifier-3 references a boolean data item. The result of the boolean expression evaluation is stored in identifier-3.

CONTINUE STATEMENT

The CONTINUE statement (figure 5-6.1) is a no-operation statement in which no action is taken by the system. The statement can be used wherever a conditional statement or an imperative statement can be used. The statement is usually used to fill an unused branch of a conditional statement. It indicates that no executable statement is present and causes an implicit transfer of control to the next executable statement.

```
                CONTINUE
```

Figure 5-6.1. CONTINUE Statement Format

DELETE STATEMENT

The DELETE RECORD statement (figure 5-7) removes a record from a file with relative, indexed, direct, or actual-key organization. Once a record has been deleted, it can no longer be accessed. The use of DELETE in a program, and its result, is affected by both the file organization and the file access mode.

```
         { file-name  RECORD                                   }
DELETE   {        [; INVALID KEY imperative-statement]         }
         { FILE  {file-name}  . . .                            }
```

Figure 5-7. DELETE Statement Format

The file must be open for I-O; any access mode is allowed. Execution of DELETE updates any FILE STATUS data item defined for the file.

File-name specifies the file from which a record is to be deleted. It must be specified in an FD entry in the File Section and can be described in a sub-schema. It must not be a file with sequential or word-address organization. The contents of the record area associated with file-name is not affected by DELETE execution.

Specification of the record to be deleted is different for different access modes:

> For a file open in the sequential access mode, the last input-output statement preceding DELETE must be a format 1 READ statement. DELETE then deletes the record read by the READ statement.

For a file open in the random access mode or in the dynamic access mode, the record to be deleted is identified by the data item specified by:

> RELATIVE KEY clause associated with file-name, if the file has relative organization;
>
> RECORD KEY clause associated with file-name, if the file has indexed, direct, or actual-key organization.

The INVALID KEY phrase can be used only for files open in random access or dynamic access mode. The imperative-statement is executed when the file does not contain the record specified by the key of the record to be deleted. When the phrase is omitted, strict ANSI usage requires an appropriate format 1 USE statement for the file. See the FILE STATUS clause in section 3.

The DELETE FILE statement deletes the file from the system. The file cannot be in the open mode at the time of execution of the DELETE FILE statement. A RETURN request is issued to the operating system on the file. If the file is a local file, the space is returned to the system and all information in the file is lost. If the file is on magnetic tape, the tape is unloaded.

DISABLE STATEMENT

The DISABLE statement (figure 5-7.1) allows MCS to break logical paths that contain sources, destinations, input queues, or output queues. The DISABLE statement is only allowed under NOS.

```
          { INPUT [TERMINAL] }
DISABLE   { OUTPUT           }   cd-name

          WITH KEY { identifier-1 }
                   { literal-1    }
```

Figure 5-7.1. DISABLE Statement Format

When a DISABLE statement with the INPUT TERMINAL phrase is executed, the communication description (CD) input area referenced by cd-name must contain the symbolic name of the source that is to be disabled. Execution of this statement results in MCS breaking all logical paths between the specified source and its associated input queues. MCS no longer accepts any message data from that source.

When a DISABLE statement is executed with the INPUT phrase, but without the keyword TERMINAL, the CD input area referenced must contain one or more symbolic queue names. These names identify the simple or compound queue that is to be disabled. Execution of this statement results in MCS breaking the logical paths between the simple queue specified, or the specified queues within the compound queue, and all sources that are capable of delivering messages to the specified queue(s). As a result, MCS no longer accepts message data from any source that is destined for a disabled queue. Execution of this statement (with or without the keyword TERMINAL) updates the STATUS KEY of the CD input area.

When a DISABLE statement with the OUTPUT phrase is executed, the CD output area referenced by cd-name must contain the symbolic names and the number of destinations that are to be disabled. Execution of this statement results in MCS breaking the logical paths between the specified destinations and the output queues that can deliver messages to specified destinations. MCS halts the delivery of messages to the specified destinations. However, any message in the process of being delivered to a destination is not affected; it arrives complete at the destination. The DISABLE statement updates the STATUS KEY field and possibly the ERROR KEY field that corresponds with the specified destinations when the statement is executed.

The KEY phrase of the DISABLE statement specifies or references a password that must match the password defined in the Application Definition Language (ADL) for the source, input queue, or destinations applicable to the DISABLE statement. The DISABLE statement can affect the entire application that contains the program executing the statement. Following the successful execution of a DISABLE statement in one program, the operation of other programs of the same application is affected if the other programs reference the disabled source, queue, or destinations. If a symbolic name is an invitation or broadcast list, only the password defined in ADL for the list is validated. The passwords of the various list numbers are not validated.

DISPLAY STATEMENT

The DISPLAY statement (figure 5-8) transfers one or more literals or the contents of one or more data items to a specified device or file. It is inefficient for large volumes of data, but its use is preferable when only a few lines of information are to be written. With this statement, information can be transferred to any sequential file or to the central site operator console. When the program is executing under control of a terminal, information can also be transferred to the terminal or to a file connected to the terminal.

Each literal and the contents of each identifier specified are transferred to a device or file in the order of specification. Literal-1 and literal-2 can specify a nonnumeric literal or any figurative constant except ALL; only a single occurrence of a figurative constant is displayed. An unsigned integer, or any other numeric literal, is allowed for literal-1 and literal-2.

Multiple items in a single DISPLAY statement appear in a single line or record, to the limit of the line possible for the implementor-name associated with the mnemonic-name of the UPON phrase. The maximum number of characters that can be displayed on a single line or written as a single record follows:

> If CONSOLE is the implementor-name, 40 is the maximum number of characters.
>
> If "TERMINAL" or any connected file is the implementor-name, 72 is the maximum number of characters for NOS. The size specified by the SCREEN command is the maximum for NOS/BE; 72 characters is the default.

```
DISPLAY { literal-1    }
        { identifier-1 }

   [ , literal-2    ]
   [ , identifier-2 ] . . .  [UPON mnemonic-name]

   [WITH NO ADVANCING]
```

Figure 5-8. DISPLAY Statement Format

For all other implementor-names, 136 is the maximum number of characters. This value can be overridden by the FL parameter of a FILE control statement.

Additional lines or records are written as necessary when data exceeds these limits.

At the central site console, any character with a display code value greater than 55_8 or with a 00 display code value appears as a blank. At a terminal, all characters are possible, with the particular representation of a character dependent on the terminal used.

Items are displayed in the following formats:

Alphanumeric character-string items are displayed as in the source program.

Literal items are displayed as in the source program.

COMP-2 items are displayed in scientific exponential notation. The value of the item is the number to the left of the E raised to the power shown to the right of the E. Group items containing COMP-2 items should not be displayed.

Integer numeric items, other than COMP-2 items, are displayed as the number of digits in the PICTURE clause, with leading zeroes suppressed. The digits are preceded by a minus sign (if negative) or a space (if positive) and the decimal point is inserted.

Noninteger numeric items, other than COMP-2 items, are displayed as the numeric value defined by the PICTURE clause, with leading zeroes suppressed. The digits are preceded by a minus sign (if negative) or a space (if positive) and the decimal point is inserted.

When ANSI=NOEDIT is specified on the COBOL compiler call statement, the DISPLAY statement performs no editing of displayed items.

UPON Phrase

The UPON phrase specifies where the information is to be displayed. When the phrase is omitted, information appears on the system file OUTPUT.

The mnemonic-name of the UPON phrase must be associated with an implementor-name in the SPECIAL-NAMES paragraph of the Environment Division. An implementor-name of CONSOLE or "TERMINAL" results in information being displayed at the console or terminal, respectively. Any other implementor-name results in the information being written to the file specified.

The file to which information is written is assumed to have sequential organization with C type blocking and Z type records.

When the implementor-name is "OUTPUT", the system provides single-spacing of records displayed. (This is in contrast to display of records through a WRITE statement that references a file connected to a terminal. WRITE does not supply a carriage control character for a connected file.) The first character of the record is displayed when the DISPLAY statement references a file named "OUTPUT".

Under the NOS/BE operating system, information displayed begins with the second character of a record when the implementor-name is "TERMINAL-C" or is "OUTPUT-C" and the file is connected to a terminal. The first character of the record is used as carriage control information and the record is displayed accordingly.

WITH NO ADVANCING Phrase

The WITH NO ADVANCING phrase is used primarily with interactive terminals. The phrase causes the terminal output mechanism to be positioned after the last character displayed so that subsequent data that is accepted or displayed begins on the same line. The phrase works differently on files associated with a terminal under the NOS or NOS/BE operating system.

Through IAF under NOS, an ACCEPT statement that follows a DISPLAY WITH NO ADVANCING statement, where both statements are from a file associated with a terminal, causes a prompt (question mark and space) to be issued after the last displayed character. Data is read beginning with the position following the prompt. Similarly, a DISPLAY statement that follows a DISPLAY WITH NO ADVANCING statement under these same conditions causes data to be displayed upon the same line following the data from the DISPLAY WITH NO ADVANCING statement. However, to accomplish the WITH NO ADVANCING, IAF might add a trailing blank to any data item displayed to the terminal.

Through INTERCOM under NOS/BE, an ACCEPT statement that follows a DISPLAY WITH NO ADVANCING statement, where both statements are on a file associated with a terminal, causes data to be read beginning with the position following the last displayed character. Similarly, a DISPLAY statement that follows a DISPLAY WITH NO ADVANCING statement under these same conditions causes overprinting.

If the WITH NO ADVANCING phrase is used with any device other than an interactive terminal, regardless of the operating system, the phrase causes overprinting of the previous line. The internal software of some terminals overrides this phrase and automatically advances the line.

DIVIDE STATEMENT

The DIVIDE statement (figure 5-9) divides one elementary numeric data item into others. The data item that receives the result depends on the format used:

Format 1 stores the result in the dividend operand identifier.

Format 2 stores the result in an identifier other than the dividend or divisor.

Format 3 is logically equivalent to format 2 with the dividend and divisor reversed in the statement.

Format 4 stores the result in an identifier other than the dividend or divisor and stores the remainder in another identifier.

Format 5 is logically equivalent to format 4, with the dividend and divisor reversed in the statement.

The composite size of all receiving data items, except the REMAINDER data item, is limited to 18 digits. The composite size is determined by superimposing all receiving identifiers on their respective decimal points in a hypothetical data item.

```
Format 1

      DIVIDE { identifier-1 }  INTO identifier-2 [ROUNDED] [, identifier-3 [ROUNDED]] . . .
             { literal-1    }

          [; ON SIZE ERROR imperative-statement]

Format 2

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

          [, identifier-4 [ROUNDED]] . . . [; ON SIZE ERROR imperative-statement]

Format 3

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

          [, identifier-4 [ROUNDED]] . . . [; 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]
```

Figure 5-9. DIVIDE Statement Format

The ROUNDED phrase and the SIZE ERROR phrase are discussed with the ADD statement. Additional information about arithmetic operations is in section 1.

Division by zero always causes a size error condition.

All literals used in the DIVIDE statement must be numeric. All identifiers must be elementary numeric items, except for those in GIVING or REMAINDER phrases that can specify elementary numeric edited items.

Format 1 DIVIDE

Format 1 DIVIDE occurs as follows: The value of identifier-1 or literal-1 is divided into the current value of identifier-2. The result is stored as the new value of identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is divided into, and the result stored in, each identifier-3 specified.

Any literal must be numeric; all identifiers must specify elementary numeric items.

Format 2 DIVIDE

Format 2 DIVIDE occurs as follows: The value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2. The result is stored as the new value of identifier-3. If identifier-4 is specified, the result is stored as the new value of each identifier-4.

Any literal must be numeric; identifier-1 and identifier-2 must specify elementary numeric items. Identifier-3 and identifier-4 can specify elementary numeric or elementary numeric edited items.

Format 3 DIVIDE

Format 3 DIVIDE is logically equivalent to format 2, except that identifier-2 or literal-2 is divided into identifier-1 or literal-1.

Format 4 DIVIDE

Format 4 DIVIDE occurs as follows: The value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2. The result is stored as the new value of identifier-3. Any remainder from the division is stored as the new value of identifier-4. Any literal must be numeric; identifier-1 and identifier-2 must specify elementary numeric items. Identifier-3 and identifier-4 can be elementary numeric edited items.

The remainder is defined as the result of subtracting from identifier-2 or literal-2 the product of identifier-1 or literal-1 and identifier-3.

The accuracy of the remainder is affected by the specification of identifier-3:

> If identifier-3 is a numeric edited item, the remainder calculation uses an intermediate field containing an unedited quotient.
>
> If identifier-3 is an unsigned item, the remainder calculation uses an intermediate field containing a signed quotient.
>
> If identifier-3 is modified by the ROUNDED phrase, the remainder calculation uses an intermediate field containing a quotient in a truncated (not a rounded) form.

Use of a COMP-2 item in division always results in a zero remainder.

If a size error condition occurs on the quotient, no remainder calculation is meaningful and the contents of identifier-3 and identifier-4 are unchanged. If a size error condition occurs in the remainder, identifier-4 remains unchanged; however, as with other instances of multiple results of arithmetic statements, the programmer is responsible for any analysis to recognize which situation actually occurred.

Format 5 DIVIDE

Format 5 DIVIDE is logically equivalent to format 4, except that identifier-2 or literal-2 is divided into identifier-1 or literal-1.

ENABLE STATEMENT

The ENABLE STATEMENT (figure 5-9.1) causes MCS to establish the logical paths between a specified symbolic name and all queues that receive messages from or send messages to MCS. This statement can also cause MCS to establish logical paths between specified queues and their associated symbolic names. The ENABLE statement is only allowed under NOS.

```
ENABLE  { INPUT [TERMINAL] }  cd-name
        { OUTPUT           }

        WITH KEY  { identifier-1 }
                  { literal-1    }
```

Figure 5-9.1. ENABLE Statement Format

At the time an ENABLE statement with the INPUT TERMINAL phrase is executed, the communication description (CD) input area referenced by cd-name must contain the symbolic name of the source that is to be enabled. Execution of the statement results in MCS establishing all logical paths between the specified source and the input queues that receive the messages. MCS begins accepting message data from the specified source.

When an ENABLE statement is executed with the INPUT phrase, but without the keyword TERMINAL, the CD input area referenced must contain one or more symbolic queue names. These names identify the simple or compound queue that is to be enabled. Execution of this statement results in MCS establishing logical paths between the simple queue specified, or the specified queues within the compound queue, and all sources that are capable of delivering messages to the specified queue(s). As a result, MCS accepts message data that is destined for the enabled queue. The STATUS KEY field of the CD input area is updated during execution of an ENABLE INPUT statement (with or without the keyword terminal).

When an ENABLE statement with the OUTPUT phrase is executed, the CD output area referenced by cd-name must contain the symbolic names and the number of destinations to be enabled. Execution of this statement results in MCS establishing the logical paths between the specified destinations and the output queues that can deliver messages to these destinations. Execution of the ENABLE statement updates the STATUS KEY field of the CD output area, and possibly the ERROR KEY field, that correspond with the specified destinations.

The KEY phrase of an ENABLE statement specifies or references a password that must match the password defined in the Application Definition Language for the source, input queue, or destinations that apply to the ENABLE statement. The ENABLE statement can affect the entire application that contains the program executing the statement. Following the successful execution of an ENABLE statement in one program, the operation of other programs of the same application is affected if the programs reference the enabled source, queue, or destinations.

ENTER STATEMENT

The ENTER statement transfers control to a subprogram written in the COMPASS assembly language or a FORTRAN compiler language. See section 15, Inter-Program Communication Facility.

EXIT STATEMENT

The EXIT statement (figure 5-10) has two functions:

> EXIT provides a common exit point for a series of procedures in a main program or subprogram.
>
> EXIT PROGRAM returns control from a called program to the calling program. See section 15, Inter-Program Communication Facility.

```
EXIT [PROGRAM]
```

Figure 5-10. EXIT Statement Format

EXIT PROGRAM does not return control to the operating system. STOP RUN is the proper termination of a main program.

When only the keyword EXIT appears, the statement has no effect on program compilation or execution other than to allow a procedure-name to be assigned to a given point in a program. It can be used to provide a common end point for a series of procedures. In this case, EXIT must be the only sentence in the paragraph.

GENERATE STATEMENT

The GENERATE statement produces a report in accordance with the report description specified in the Report Section of the Data Division. See section 6, Report Writer Facility.

GO TO STATEMENT

The GO TO statement (figure 5-11) transfers control from one part of the Procedure Division to the paragraph or section identified in the statement.

> Format 1 names the procedure to which control transfers; this procedure-name can be modified by execution of an ALTER statement.

```
Format 1

    GO TO [ procedure-name-1]

Format 2

    GO TO procedure-name-1

       [, procedure-name-2] . . . , procedure-name-n

            DEPENDING ON identifier
```

Figure 5-11. GO TO Statement Format

> Format 2 transfers control to a procedure selected by the value of an identifier in the statement.

Control can be transferred to any paragraph or section in any segment of the program.

Format 1 GO TO

Format 1 transfers control to the procedure-name existing at the time the statement executes. The paragraph-name or section-name compiled into the statement can be changed by an ALTER statement that executes prior to the GO TO statement execution.

If procedure-name-1 is omitted from format 1, the GO TO statement must appear in a paragraph that contains only the GO TO statement. In this case, an ALTER statement must execute prior to the GO TO statement to establish the procedure-name for control transfer.

Format 2 GO TO

Format 2 transfers control to one of several specified procedures, depending on the value of the identifier at the time the statement executes. Identifier must be an elementary numeric item containing a positive or unsigned integer; it must be described by a USAGE clause specifying DISPLAY, COMP, or COMP-1; and it cannot be scaled.

One procedure-name must be specified for each value that the identifier might assume. When the value of the identifier is 1, control transfers to the first procedure-name specified; when the identifier value is 4, control transfers to the fourth procedure-name specified; and so forth. A value other than a positive or unsigned integer, or a value exceeding the number of procedure-names specified, transfers control to the next statement in the normal sequence of execution.

IF STATEMENT

The IF statement (figure 5-12) evaluates the specified condition. The next program action depends on whether the condition is true or false:

> If the condition is true, the statement immediately following the IF statement is executed.
>
> If the condition is false, the statement immediately following the keyword ELSE is executed.

```
IF condition;[THEN] { , statement-1  . . . }
                    { NEXT SENTENCE        }

        { ; ELSE  statement-2  . . .  [; END-IF] }
        { ; ELSE NEXT SENTENCE                   }
        { ; END-IF                               }
```

Figure 5-12. IF Statement Format

Section 1 discusses conditions.

The keywords ELSE NEXT SENTENCE can be omitted if they immediately precede the period separator at the end of a sentence. If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

Both statement-1 and statement-2 can be conditional statements or imperative statements; either can be followed by a conditional statement. (See table 5-1.) Either statement can contain an IF statement; in this case, the IF statement is said to be nested.

A nested IF statement can be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been paired with an ELSE. An example of a nested IF in which IF B and ELSE ADD are paired and IF A and ELSE GO TO are paired is:

```
IF A IS NUMERIC MULTIPLY A BY B
    IF B IS LESS THAN 50
        ADD A B TO C
    ELSE ADD A B TO D
ELSE GO TO BADCLASS.
```

The scope of an IF statement can be terminated by an END-IF at the same level of nesting. An END-IF applies to the preceding IF statement that has not been paired with an END-IF. An example of delimited IF statements follows:

```
IF A = 1
    IF B = 2
        PERFORM B2-CODE
    ELSE
        PERFORM NOT-B2-CODE
    END-IF
    PERFORM A1-CODE
ELSE
    PERFORM NOT-A1-CODE
END-IF
```

INITIALIZE STATEMENT

The INITIALIZE statement (figure 5-13) sets selected categories of data items to predetermined values.

```
INITIALIZE identifier-1  [, identifier-2] . . .

  [              ( ALPHABETIC            )        ]
  [              | ALPHANUMERIC          |        ]
  [              | NUMERIC               |        ]
  [  REPLACING  <  ALPHANUMERIC-EDITED    >  DATA ]
  [              | NUMERIC-EDITED        |        ]
  [              ( BOOLEAN               )        ]
  [                                               ]
  [              BY { identifier-3 }              ]
  [                 { literal      }              ]
```

Figure 5-13. INITIALIZE Statement Format

In the statement, identifier-1 and any identifier-2 represent receiving items; identifier-3 and literal represent the sending item. When the REPLACING phrase is omitted, a figurative constant is the sending item. INITIALIZE executes as if a series of MOVE statements had been written with the specified sending item and elementary receiving items. Any subscripting or indexing associated with the sending or receiving items is evaluated only once, immediately before any item is initialized.

Index data items and elementary FILLER data items are not affected by INITIALIZE. A group FILLER item can only be affected indirectly by way of each named elementary item within it.

Identifier-1 and identifier-2 can be group or elementary items. They have the following restrictions:

Level 66 and index data items must not be used.

Neither the identifiers nor items subordinate to them can be described by the DEPENDING ON phrase of the OCCURS clause.

Any item that is subordinate to a receiving area identifier and which contains the REDEFINES clause is excluded from this operation. Any item that is subordinate to such an item is also excluded. However, a receiving area identifier can itself have a REDEFINES clause or be subordinate to a data item with a REDEFINES clause.

If identifier-1 or any identifier-2 is a group item, it cannot be reference modified.

When the REPLACING phrase is omitted, receiving items are set as follows:

Alphabetic, alphanumeric, and alphanumeric edited items are set to the figurative constant SPACES.

Numeric and numeric edited items are set to the figurative constant ZEROES.

Boolean items are set to all boolean character zeroes.

The REPLACING phrase specifies a literal or other item that is implicitly moved to each receiving item. The category of the sending item and any elementary receiving item must be consistent with the category defined by the keyword used in the REPLACING phrase. Identifier-3 must not be an index data item.

When the receiving identifier is a group item, an elementary item within it is affected only when the elementary item category is the same as that declared in the phrase. All such elementary receiving fields, including all occurrences of table items within the group, are affected.

INITIATE STATEMENT

The INITIATE statement begins processing of a report. See section 6, Report Writer Facility.

INSPECT STATEMENT

The INSPECT statement (figure 5-14) determines the number of occurrences of one or more characters in a data item; and, depending on the format of the statement, either counts, or replaces, or counts and replaces the characters identified.

Format 1 counts the number of times a specified character string appears.

Format 2 replaces occurrences of a specified character string with a character string of equal length.

Format 3 combines the operations of format 1 and format 2.

In all formats, the data item to be inspected, identifier-1, must be described as USAGE IS DISPLAY. (Implicit DISPLAY usage exists for all group items and for elementary items not otherwise declared with a USAGE clause.) The data item to hold the tally, identifier-2, must be an elementary numeric item. All other data items in the format must be elementary items in the alphabetic or alphanumeric class, or be numeric class items specified as USAGE IS DISPLAY. Nonnumeric literals, including figurative constants except ALL, must be used in the formats.

A figurative constant is one character in length for search or tallying purposes. When it is the replacing string, a figurative constant repeats as necessary to equal the size of the string to be replaced.

Inspection Cycle

For all formats, inspection of identifier-1 occurs as a series of cycles. Each cycle is the same except for the position within identifier-1 at which the cycle begins. The string being sought in identifier-1 follows the keywords ALL, LEADING, and FIRST and is within the boundaries established by any BEFORE INITIAL or AFTER INITIAL phrases. Another appearance of ALL, LEADING, or FIRST defines a subsequent search string.

Format 1

```
                                       { { {ALL       }  {literal-1   } } [{BEFORE}         {literal-2   }] }         }
INSPECT identifier-1 TALLYING { ,identifier-2 FOR { , {LEADING   }  {identifier-3} } [{AFTER }  INITIAL {identifier-4}] } . . . } . . .
                                       { {  CHARACTERS               } }                                       }         }
```

Format 2

```
                                ( CHARACTERS BY {literal-4   } [{BEFORE}  INITIAL  {literal-5   }]                                                )
INSPECT identifier-1 REPLACING (                {identifier-6} [{AFTER }           {identifier-7}]                                                )
                                ( { {ALL    } { {literal-3   }    {literal-4   } [{BEFORE}         {literal-5   }] }       }      )
                                ( { ,{LEADING} { ,{identifier-5} BY {identifier-6} [{AFTER }  INITIAL {identifier-7}] } . . . } . . . )
                                ( { {FIRST  } {                                                                      }       }      )
```

Format 3

```
                                       { { {ALL       }  {literal-1   } } [{BEFORE}         {literal-2   }] }         }
INSPECT identifier-1 TALLYING { ,identifier-2 FOR { , {LEADING   }  {identifier-3} } [{AFTER }  INITIAL {identifier-4}] } . . . } . . .
                                       { {  CHARACTERS               } }                                       }         }

                       ( CHARACTERS BY {literal-4   } [{BEFORE}  INITIAL  {literal-5   }]                                                )
[BEFORE]  REPLACING   (                {identifier-6} [{AFTER }           {identifier-7}]                                                )
[AFTER ]              ( { {ALL    } { {literal-3   }    {literal-4   } [{BEFORE}         {literal-5   }] }       }      )
                       ( { ,{LEADING} { ,{identifier-5} BY {identifier-6} [{AFTER }  INITIAL {identifier-7}] } . . . } . . . )
                       ( { {FIRST  } {                                                                      }       }      )
```

Figure 5-14. INSPECT Statement Format

A series of inspection cycles is made until each character position in identifier-1, within the limits of any BEFORE and AFTER phrases, has been either the beginning character position for an inspection or a part of a matching character string. For the inspections:

> BEFORE begins inspection at the leftmost character position of identifier-1 and stops at the beginning of the string specified by the BEFORE phrase.
>
> AFTER begins inspection immediately to the right of the string specified by the AFTER phrase and stops at the rightmost character position of identifier-1.

The number of inspection cycles depends on the keywords used in the format:

> When the word CHARACTERS is specified, each character position in identifier-1, within the limits of any BEFORE or AFTER phrase, is considered to meet search criteria and is counted or replaced by the single character item following the keywords CHARACTERS BY.
>
> When the keyword ALL is specified, inspections are repeated until all characters in identifier-1 have been considered as the beginning character position or part of a match, within the limits of any BEFORE or AFTER phrase.
>
> When the keyword LEADING is specified, inspections are repeated only as long as the first cycle finds a match and immediately successive inspections find a match.
>
> When the word FIRST is specified for formats 2 or 3, inspections continue until one match has been found and the string or strings replaced.

The first in the cycles of inspections of identifier-1 begins at the leftmost character position of the data item or at the position specified by a BEFORE or AFTER phrase. Each string to be tallied or replaced is compared, in turn, with an equal number of characters in the data item; the comparison for each string begins in the same character position. Whenever a match occurs, tallying and/or replacing takes place and a new cycle begins.

The beginning character position is updated before the new cycle starts. If no match has been found on the previous inspection, the next inspection cycle begins at the character position immediately to the right of the beginning character position for previous inspection. If a match has been found, however, inspection begins one character immediately to the right of the matching characters in the data item.

During inspection, all items described as alphanumeric are treated as a character-string. All items that are of the category alphanumeric edited, numeric edited, boolean, or unsigned numeric are treated as alphanumeric items. Signed numeric items are treated as if they had been moved to unsigned numeric items, so that only the absolute value of the data item is relevant for comparison.

Format 1 INSPECT

Format 1 is a tallying operation. Identifier-2 is the tally item. It must be an elementary numeric data item. Identifier-2 is not initialized by INSPECT.

If the ALL phrase is specified, the value of the data item referenced by identifier-2 is incremented by one for each occurrence of literal-1 or identifier-3 matched within the contents of the data item referenced by identifier-1.

If the LEADING phrase is specified, the value of the data item referenced by identifier-2 is incremented by one for each leading contiguous occurrence of literal-1 or identifier-3 matched within the contents of the data item referenced by identifier-1.

If the CHARACTERS phrase is specified, the value of the data item referenced by identifier-2 is incremented by one for each character within the contents of the data item referenced by identifier-1.

Format 2 INSPECT

Format 2 is a replacing operation. The replacing string, literal-4 or identifier-6, must have the same number of characters as the string to be replaced. When the replacing string and the string to be replaced share a part of their storage areas, the results are undefined.

When the word CHARACTERS is specified, literal-4 and literal-5 and the data items referenced by any identifier-6 and identifier-7 must be one character in length.

The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

When the word ALL is specified, each occurrence of literal-3 or identifier-5 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4 or identifier-6. When more than one ALL phrase exists in a REPLACING phrase, each replacement is performed on the original character-string.

When the word LEADING is specified, each leading contiguous occurrence of literal-3 or identifier-5 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4 or identifier-6.

When the word FIRST is specified, the leftmost occurrence of literal-3 or identifier-5 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4 or identifier-6.

When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

Format 3 INSPECT

Format 3 is interpreted and executed as though a format 2 INSPECT and a format 1 INSPECT were executed for the same identifier-1. The syntax of formats 1 and 2 also applies to format 3.

The order in which tallying and replacing take place depends on the word immediately preceding the keyword REPLACING:

> When AFTER is specified, replacement occurs before tallying.
>
> When BEFORE is specified, tallying occurs before replacing.
>
> When neither word is specified, tallying occurs before replacing.

Examples of INSPECT statement use are:

> INSPECT WD-1 TALLYING CT-1 FOR ALL "P" BEFORE REPLACING FIRST "P" BY "Q". Assuming that the program initialized CT-1 to zero and WD-1 holds HIPPOPOTOMUS, the statement results in a value of 3 for CT-1 and a value of HIQPOPOTOMUS in WD-1.
>
> INSPECT WD2 REPLACING ALL "P" BY "Q" AFTER INITIAL "P". Assuming that WD2 holds a value LOLLIPOP, the statement results in a value of LOLLIPOQ in WD2.
>
> INSPECT WD3 REPLACING ALL "TT" BY "XX", ALL "AX" BY "YY". Assuming that WD3 holds a value FLATTERY, the statement results in a value of FLAXXERY.

MERGE STATEMENT

The MERGE statement combines two or more identically sequenced files on a set of specified keys. See section 7, Sort/Merge Facility.

MOVE STATEMENT

The MOVE statement (figure 5-15) transfers data from one data item to one or more other data items in accordance with the rules of editing.

> Format 1 transfers specified elementary or group items to other specified items.
>
> Format 2 transfers selected elementary items within the specified group item to corresponding elementary items in other specified group items.

Data remains available in the sending item after MOVE execution.

Identifier-1 and literal-1 represent the sending item; all other identifiers represent receiving items. When a sending item and a receiving item share a part of their storage areas, the results are undefined.

When more than one receiving item is specified, the data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, and so forth. Any subscripting, indexing, or reference modification associated with identifier-2, or any other receiving item, is evaluated immediately before the data is moved. Any subscripting, indexing, or reference modification associated with the ending item (identifier-1) is evaluated only once, immediately before data is moved to the first receiving item. The move occurs as if the following equivalent statements were written, where TEMP is an intermediate result with the size and type of A(B):

> MOVE A(B) TO B, C(B)
>
> is equivalent to
>
> MOVE A(B) TO TEMP
> MOVE TEMP TO B
> MOVE TEMP TO C(B)

When the sending item is smaller than the receiving item, data is aligned in the receiving item and zero or blank filled:

> Category alphanumeric, alphabetic, and alphanumeric edited items are aligned at the left unless the JUSTIFIED clause specifies right alignment. Blank fill completes the item. The ALL option of a figurative constant fills these items without blanks. Any other editing specified by the picture of the receiving item, such as $ or +, is followed during the move.
>
> Category numeric and numeric edited items are aligned by the decimal point whether the decimal point is explicitly stated or assumed. Any other editing specified by the picture of the receiving item, such as $ or +, is followed during the move.
>
> Category boolean items are aligned at the left unless the JUSTIFIED clause specifies right alignment. Zero fill completes the item.

When the sending item is larger than the receiving item, data in the receiving item is truncated, as appropriate:

> Category alphanumeric, alphabetic, and alphanumeric edited items are aligned left and truncated on the right, unless the JUSTIFIED clause is specified for the receiving item. When JUSTIFIED appears, items are aligned right and truncated on the left.
>
> Category numeric and numeric edited items are aligned by the decimal point whether the decimal point is explicitly stated or assumed. Truncation can occur at both the left and right.
>
> Category boolean items are aligned left and truncated on the right unless the JUSTIFIED clause is specified for the receiving item. When JUSTIFIED is specified, items are aligned right and truncated on the left.

When the sending item is COMP-1, COMP-2, or COMP-4, and the receiving item is DISPLAY, COMP, or numeric edited, decimal truncation of the high order digits is not attempted.

Format 1

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

Format 2

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

Figure 5-15. MOVE Statement Format

Format 1 MOVE

A format 1 MOVE moves elementary or group items. Identifier-1 or literal-1 is moved to identifier-2. If identifier-3 is specified, identifier-1 or literal-1 is also moved to each identifier-3 specified. The move to identifier-3 is independent of any prior identifier-2 or identifier-3.

Group Item Move

When format 1 is used and the sending or receiving item is a group item, the move is termed a group item move. In this instance, the move occurs as if it were an alphanumeric-to-alphanumeric elementary move. No conversion of data from one form of internal representation to another occurs. The receiving area is filled without consideration for the individual elementary or group items, contained within either the sending or receiving area, except for a group sending item that has a subordinate format 2 OCCURS clause description. For this latter type of item, only that part of the table area specified by the value of the DEPENDING ON data-name phrase is moved. When the sending item is a group item, no justification (except the normal left justification) or editing takes place. The description of the receiving field is ignored except for its size.

Elementary Item Move

When elementary items are specified for both the sending and receiving items, the move is termed an elementary move. Table 5-2 shows legal moves between items and indicates the move procedures. Additional information about moves between specific types of items follows.

Legal Operand to Signed Numeric Item - Sign representation is converted as necessary. A positive sign is generated for an unsigned sending item. If the receiving item sign is described by SEPARATE, the numeric size is one less than the actual size of the receiving item.

Legal Operand to Unsigned Numeric Item - Absolute value of the sending item is moved, with no sign generated for the receiving item. COMP-2 items are rounded, moved to an intermediate COMP item, then the absolute value is moved to the receiving item.

Alphanumeric to Numeric or Numeric Edited Item - Sending item is treated as an unsigned numeric integer. Move of any nonnumeric characters produces undefined results.

ALL Literal or Figurative Constant to Numeric or Numeric Edited Item - Sending string, which must be only digits, is repeated on the right until the string size equals the size of the integer portion of the receiving item.

Legal Operand to Alphabetic Item - Any nonalphabetic characters are moved without detection.

Signed Numeric to Alphanumeric or Alphanumeric Edited Item - Operational sign is not moved. If the operational sign is described as SEPARATE, the size of the sending item is considered to be one less than its actual size.

COMP-1 or COMP-4 to Alphanumeric Item - Item is moved to an intermediate COMP item with the same picture but no sign, then treated as an alphanumeric item.

TABLE 5-2. TYPE OF MOVE OPERATIONS

Sending Operand	Receiving Data-Item							
	Group data-item	Alphabetic	Alpha-numeric	Alpha-numeric edited	Numeric edited	Numeric integer/ nonint.	Index data-item	Boolean
Group data-item	P	P	P	P	P	P	X	P
Elementary data-item								
Alphabetic	P	P	P	P	X	X	X	X
Alphanumeric	P	P	P	P	F	F	X	B
Alphanumeric edited	P	P	P	P	X	X	X	X
Numeric edited	P	X	P	P	X	X	X	X
Numeric integer	P	X	S	S	A	A	X	X
noninteger	P	X	X	X	A	A	X	X
Boolean	P	X	P	P	X	X	X	B
Other operand								
Literal numeric integer	P	X	S	S	A	A	X	X
numeric noninteger	P	X	X	X	A	A	X	X
nonnumeric	P	P	P	P	F	F	X	B
boolean	P	X	P	P	X	X	X	B
Figurative constant ZEROS	P	X	P	P	A	A	X	B
SPACES	P	P	P	P	X	X	X	X
Other	P	P	P	P	F	F	X	X
Index data-item	X	X	X	X	X	X	X	X

A	Algebraic move
B	Boolean move
F	Alphanumeric item treated as unsigned numeric display item with algebraic move
P	Physical move
S	Sign removed; physical move to display item
X	Not allowed

DISPLAY Item to Alphanumeric or Alphanumeric Edited Item - Treated as alphanumeric except for operational sign described above.

USAGE Other than DISPLAY to Alphanumeric Edited Item - Sending item is converted before the move to an intermediate alphanumeric item with a size equal to the number of noninsertion characters.

Numeric Literal to Alphanumeric or Alphanumeric Edited Item - Treated as nonnumeric literal excluding sign.

ZERO to Alphanumeric or Alphanumeric Edited Item - Treated as if ALL were specified.

Table 5-2 also indicates that some moves are algebraic, boolean, or physical character moves. An algebraic move is from a category numeric or numeric edited item to a category numeric or numeric edited item as follows:

> In a move to a numeric item, the data is aligned by the decimal point and moved to the receiving digit positions with zero fill or truncation on either end, as required. When an assumed decimal point is not explicitly specified, the data item is treated as if it has an assumed decimal point immediately following its rightmost digit. For COMP-1 or COMP-4 items, decimal point alignment is accomplished by multiplying or dividing the data by an appropriate power of ten. For COMP-2 items, alignment is never needed since the item occupies a full memory word and the point location is implied by the exponent.
>
> A move to a numeric edited item is the same as to a numeric item, except where editing requirements cause replacement of the leading zeros.

A physical move involving a category alphabetic, alphanumeric, or alphanumeric edited receiving item is as follows:

> If the JUSTIFIED clause is not specified for the receiving item, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item, with space fill or truncation to the right, as required.
>
> If the JUSTIFIED clause is specified for the receiving item, sending data is aligned at the rightmost character position in the receiving data item with space fill or truncation to the left, as required.

A boolean move is an elementary move in which the receiving item is boolean. If the JUSTIFIED clause is not specified for the receiving item, the sending data is moved to the receiving character positions and aligned at the leftmost character position, with zero fill or truncation to the right as required. If the JUSTIFIED clause is specified for the receiving item, sending data is aligned at the rightmost character position of the receiving item, with zero fill or truncation to the left as required.

Format 2 MOVE

Format 2 MOVE moves selected subordinate identifier-1 items to selected data items subordinate to identifier-2. Either the keyword CORRESPONDING or its equivalent abbreviation CORR can be specified in the format. If identifier-3 is specified, each identifier-3 is treated the same as identifier-2. All identifiers in format 2 must be group items.

Items in identifier-2 are considered to correspond and are selected for the move when all of these conditions are true:

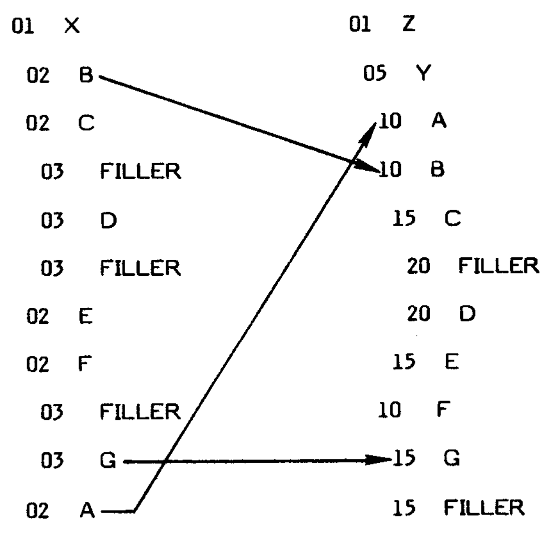
> The item subordinate to identifier-1 has the same data-name as an item subordinate to identifier-2.
>
> Either the sending item or the receiving item is an elementary data item.
>
> All possible qualifiers for the sending data items, up to but not including identifier-1, are identical to all possible qualifiers of the receiving data item up to but not including identifier-2. Correspondence of the qualified data-name includes FILLER, even though the word FILLER is not otherwise a qualifier.

Subordinate items not considered for the move include: a data item with a level-number 66 or 88; a data item with a Data Description entry containing a REDEFINES, OCCURS, or USAGE IS INDEX clause; a data item subordinate to one of these items; or a FILLER data item.

Identifier-1 and identifier-2 can include data items described by a REDEFINES or OCCURS clause in the Data Description entry and data items subordinate to such items. They cannot include data items with a level-number 66 or with a USAGE IS INDEX clause in the Data Description entry. Identifiers cannot be reference modified.

The class of any corresponding pair of data items need not be identical for a MOVE statement.

MOVE CORR X TO Y moves A, B, and G but not C, D, E, or F in the following example:



MOVE Examples

Examples of MOVE operations follow. ↑ is the assumed decimal point.

Sending Item	Picture of Receiving Item	Receiving Item
1 2$_{\uparrow}$3	999V99	$\boxed{0}\boxed{1}\boxed{2}_{\uparrow}\boxed{3}\boxed{0}$
	9999	$\boxed{0}\boxed{0}\boxed{1}\boxed{2}_{\uparrow}$
	SIGN LEADING SEPARATE, 99V9	$\boxed{+}\boxed{1}\boxed{2}_{\uparrow}\boxed{3}$
	9(10) COMP-1	See section 11
	9(10) COMP-2	See section 11
	9(4)COMP-4	See section 11
ALL "37"	X(5)	$\boxed{3}\boxed{7}\boxed{3}\boxed{7}\boxed{3}$
"ABC"	-9999.99	0ABC.00
-1.999 COMP-2	S99 SEPARATE	02-

MULTIPLY STATEMENT

The MULTIPLY statement (figure 5-16) multiplies one or more pairs of numeric items and sets the result or results in specified data items. The data item that receives the result depends on the format used:

Format 1 stores the result in the operand identifier following the word BY. If more than one receiving operand is specified, the respective product is stored in each receiving operand.

Format 2 stores the result in an identifier other than one of the operands. If more than one receiving operand is specified, the product is stored in each receiving operand.

The composite size of the data items in which the result is stored is limited to 18 digits. The composite size is determined by superimposing all result identifiers aligned on their respective decimal points in a hypothetical data item.

The ROUNDED phrase and the SIZE ERROR phrase are discussed with the ADD statement. Additional information about arithmetic operations is in section 1.

Format 1 MULTIPLY

Format 1 MULTIPLY occurs as follows: The value of identifier-1 or literal-1 is multiplied by the current value of identifier-2. The result is stored as the new value of identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is multiplied by, and the result stored in, each identifier-3.

Identifier-1, identifier-2, identifier-3, and all literals must be elementary numeric data items.

Format 2 MULTIPLY

Format 2 MULTIPLY occurs as follows: The value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2. The result is stored as the new value of identifier-3. If identifier-4 is specified, the result is also stored as the new value of each identifier-4 specified.

Identifier-1, identifier-2, and all literals must be elementary numeric items. Identifier-3 and identifier-4 can be elementary numeric or elementary numeric edited items.

OPEN STATEMENT

The format 1 OPEN statement (figure 5-17) initiates file processing. OPEN must be successfully executed prior to execution of any other statement that references the file, with the following exceptions:

> Files accessed by ACCEPT and DISPLAY statements cannot be referenced in an OPEN statement, except for a file with the implementor-name "OUTPUT".
>
> Sort or merge files must not be opened by the program, as discussed in section 7, Sort/Merge Facility.

Section 6, Report Writer Facility, discusses OPEN in relation to report files.

The format 2 OPEN statement is used to open files associated with a relation specified in a sub-schema. Refer to section 14, Sub-Schema Facility.

During OPEN statement processing, tape file labels are checked or written unless the NO REWIND phrase is specified. In this case, no label processing is performed. For indexed, direct, and actual-key files, the key of

```
Format 1

    MULTIPLY {identifier-1}  BY identifier-2 [ROUNDED]  [, identifier-3 [ROUNDED]] . . .
             {literal-1   }

        [; ON SIZE ERROR imperative-statement]


Format 2

    MULTIPLY {identifier-1}  BY  {identifier-2}  GIVING identifer-3 [ROUNDED]
             {literal-1   }      {literal-2   }

        [, identifier-4 [ROUNDED]] . . . [; ON SIZE ERROR imperative-statement]
```

Figure 5-16. MULTIPLY Statement Format

```
Format 1

           ( INPUT file-name-1 [REVERSED          ]  [, file-name-2 [REVERSED          ]] . . . )
           (                   [WITH NO REWIND    ]  [              [WITH NO REWIND    ]]       )
           (                                                                                    )
    OPEN  <  OUTPUT file-name-3 [WITH NO REWIND]  [, file-name-4 [WITH NO REWIND]] . . .        > . . .
           (                                                                                    )
           ( I-O file-name-5 [, file-name-6] . . .                                              )
           (                                                                                    )
           ( EXTEND file-name-7 [, file-name-8] . . .                                           )
```

Figure 5-17. OPEN Statement Format 1

reference is set to the primary key data item. The FILE STATUS data item, if any, is updated. Successful execution of OPEN makes the associated record area available to the program; it does not obtain or release a data record.

A file can be opened more than once during a program, as long as an intervening CLOSE statement without the REEL/UNIT phrase or LOCK phrase precedes a subsequent OPEN.

If the file to be opened has been described with the OPTIONAL keyword of the SELECT clause and the file is not present, no program action occurs during open processing. See the READ statement.

In the statement, each file-name must specify the name of a file established in a SELECT clause of the Environment Division. The FD entry for an existing file must describe the same file structure as described when the file was created. The file can be a file described in a sub-schema.

The OPEN statement establishes the open mode and positions the file. The specific statements allowed subsequently for reading, writing, positioning, or updating a file depend, in addition to the open mode, on the ACCESS MODE clause in the Environment Division and on the file organization. See table 3-2 in section 3. The four open modes are established by a keyword of the same name:

INPUT

Valid for all existing files that are to be read, but are not to be extended or updated. For tape files, causes any beginning reel or file labels to be checked. Positions the file to its beginning, unless the file resides in the file INPUT or unless the REVERSED phrase or the NO REWIND phrase is also specified for a tape file.

OUTPUT

Valid for all new files that are to be written, but are not to be extended or read. Required to create any new file. For tape files, causes any beginning reel or file labels to be written and the file to be positioned after the labels. For indexed, direct, and actual key files, the first OPEN OUTPUT specified for a file determines the collating sequence for the keys of the file. (The collating sequence affects the following keys: both record and alternate keys of indexed files and only alternate keys of direct and actual key files.) If the ANSI=AUDIT parameter is not specified in the COBOL5 control statement, the collating sequence for the file is the sequence established for the program at the time the OPEN OUTPUT statement is executed. If the ANSI=AUDIT parameter is specified, the collating sequence for the file is the native sequence. The collating sequence for a file remains the same for the life of the file.

I-O

Valid for all existing mass storage files that are to be read or updated. Positions the file to its beginning.

EXTEND

Valid for existing sequential mass storage files that are to have records added at the end of existing records. When a file opened with EXTEND is subsequently read, no boundary exists to distinguish original file records from appended records. A file created through a COPY control statement is positioned after an end-of-file.

For files on mass storage, OPEN with INPUT or I-O positions the file to the beginning of the first record currently existing within the file. If no file records exist, the next executed format 1 READ statement for the file results in an at end condition.

NO REWIND Phrase

The NO REWIND phrase is valid only for sequential files on tape or mass storage. When the phrase is omitted, the file is rewound during OPEN processing.

NO REWIND does not reposition a file: the program is responsible for correct file positioning prior to OPEN, such as prior execution of a CLOSE statement with the NO REWIND phrase.

If LABEL RECORDS ARE STANDARD is specified in the FD for the file, no label processing is performed. If this is the first OPEN of the file, the LABEL control statement should be used to properly label the file.

REVERSED Phrase

The REVERSED phrase is valid only for sequential files. The files must be on mass storage or on a single reel of tape. The phrase can be used only if the records on the file are record type F, W, or Z. If the record type is F, the record length must be a multiple of 10 characters. Refer to table 4-2 to relate record types to Record Description Entry Clauses. The block type must not be K or E. Refer to the BLOCK CONTAINS clause to determine block type.

REVERSED positions the file at its end, so that subsequent READ statements make file records available in reverse order: the first record returned to the record area is the last record in the file. REVERSED phrase processing is inefficient; its use should be avoided when possible.

NOTE

Because of anticipated changes in this product, use of the REVERSED phrase is not recommended. For guidelines, see appendix F.

Multifile Set Tape OPEN

Multifile set files are those declared by the MULTIPLE FILE TAPE clause of the I-O-CONTROL paragraph; more than one file exists on the reel or set of reels. (One file with several partition boundaries does not define a multifile set.) Treatment of a file contained in a multifile environment is logically equivalent to the treatment of a sequential file contained in a single file tape environment. All files in a multifile set must have standard labels.

Only one of the files in a multifile tape can be open at any given time. The files can be opened in any order when they are to be read. When a file is opened with OUTPUT or EXTEND, however, a specific tape structure is presumed: the file with the position number being written must be preceded by existing files with immediately lower position numbers. Further, no existing file can have a position number higher than that of the file being opened. Writing after an open with EXTEND or OUTPUT effectively destroys all files positioned after the file that was written.

PERFORM STATEMENT

The PERFORM statement (figure 5-18) explicitly transfers control to a specified set of statements, with control returning implicitly when execution of the specified statements is complete. If procedure-name-1 is specified, the specified set of statements is a procedure. If procedure-name-1 and procedure-name-2 are specified, the specified set of statements is a range of procedures. If procedure-name-1 is not specified, the specified set of statements is an imperative statement, delimited by END-PERFORM.

The number of times the specified set of statements executes depends on the format:

Format 1 executes the statements one time.

Format 2 executes the statements the number of times indicated in the PERFORM statement.

Format 3 executes the statements until the condition in the PERFORM statement is satisfied.

Format 4 executes the statements until all conditions in the PERFORM statement are satisfied.

Unless END-PERFORM is specified, the procedure to be executed by a PERFORM statement should not immediately follow the PERFORM statement. The procedure might execute one more time than intended.

A procedure associated with one PERFORM statement can overlap or intersect the procedures associated with another PERFORM statement, as long as neither PERFORM statement includes a PERFORM statement associated with the other procedure.

If procedure-name-1 is omitted, an imperative statement and the END-PERFORM phrase must be specified. If procedure-name-1 is specified, imperative-statement and END-PERFORM must not be specified.

When procedure-name-1 is specified, the PERFORM statement is referred to as an out-of-line PERFORM statement. When procedure-name-1 is not specified, the PERFORM statement is referred to as an in-line PERFORM statement.

See section 8, Segmentation Facility, for information about PERFORM statements in segmented programs.

```
Format 1

PERFORM [procedure-name-1 [{THRU   } procedure-name-2]] [; imperative-statement ; END-PERFORM]
                           {THROUGH}

Format 2

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

     [; imperative-statement ; END-PERFORM]

Format 3

PERFORM [procedure-name-1 [{THRU   } procedure-name-2]] [; WITH TEST {BEFORE}]
                           {THROUGH}                                 {AFTER }

     UNTIL condition-1  [; imperative-statement ; END-PERFORM]

Format 4

PERFORM [procedure-name-1 [{THRU   } procedure-name-2]] [; WITH TEST {BEFORE}]
                           {THROUGH}                                 {AFTER }

     VARYING {identifier-1  } FROM {identifier-2  } BY {identifier-3} UNTIL condition-1
             {index-name-1  }      {index-name-2  }    {literal-2   }
                                   {literal-1     }

     [AFTER {identifier-4 } FROM {identifier-5 } BY {identifier-6} UNTIL condition-2] . . .
            {index-name-3 }      {index-name-4 }    {literal-4   }
                                 {literal-3    }

      [; imperative-statement ; END-PERFORM]
```

Figure 5-18. PERFORM Statement Format

Nested PERFORM Statements

When a sequence of statements within a PERFORM procedure includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must either be totally included in, or totally excluded from, the sequence referred to by the first PERFORM. Thus, an active PERFORM whose execution begins within the range of another active PERFORM must not allow control to pass to or beyond the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements cannot have a common exit.

All PERFORM Formats

All literals must be numeric literals. All identifiers must be elementary numeric data items described in the Data Division. Procedure-name-1 and procedure-name-2 can be paragraph-names or section-names.

The THROUGH phrase establishes a range of procedures to be executed under PERFORM statement control. A consecutive sequence of operations is executed, beginning at procedure-name-1 and ending with the last statement in any procedure-name-2. Any number of procedures can appear contiguously between procedure-name-1 and procedure-name-2. At the end of the last statement in the range of procedures, control implicitly returns to the next executable statement following PERFORM.

Within the procedures, GO TO and PERFORM statements can occur, but control should always be returned to within the procedures. When two or more logical paths exist to the return point, procedure-name-2 can be the name of a paragraph consisting of the EXIT statement to which all paths lead.

Format 1 PERFORM

Format 1 PERFORM executes the specified set of statements one time.

Format 2 PERFORM

Format 2 PERFORM executes the specified set of statements the number of times indicated by the value of integer-1 or the data item referenced by identifier-1. If the value is zero or negative, the statements are not executed. Changes to identifier-1 within the statements have no effect on the number of times statements are performed.

Format 3 PERFORM

Format 3 PERFORM executes the specified set of statements until the condition specified after keyword UNTIL is true. Condition-1 can be any conditional expression. The evaluation of the condition takes place at the beginning of the PERFORM statement execution, if the WITH TEST phrase is not specified. If the condition is true at the start of PERFORM execution, the statements are not executed.

If the WITH TEST AFTER phrase is specified, the evaluation of the condition takes place after the execution of the last statement of the specified set of statements. The statements are executed at least once, regardless of the initial conditions.

If the WITH TEST BEFORE phrase is specified, the evaluation of the condition takes place at the beginning of the PERFORM statement execution. If the condition is true at the start of PERFORM execution, the statements are not executed.

Format 4 PERFORM

Format 4 PERFORM executes the specified set of statements while varying a data item or index-name until condition-1 is true. During execution of the specified set of statements, any changes to the variables specified in the format are taken into consideration and affect the operation of the PERFORM statement. The evaluation of the conditions takes place at the beginning of the PERFORM statement execution, if the WITH TEST phrase is not specified. If condition-1 is true at the start of execution, the statements will not be executed.

VARYING Phrase

In Format 4, the operands immediately following these keywords have the function:

VARYING	Item to be varied
FROM	Initial value of VARYING item
BY	Increment or decrement to VARYING item; must not be zero

AFTER Phrase

AFTER allows varying more than one variable. The variable is used in the executions associated with condition-1. For two or more identifiers to be varied, the last identifier goes through a complete cycle each time the preceding identifier is incremented or decremented. Details of the execution of PERFORM for one or two conditions are shown in the flowcharts of figure 5-19.

When an index-name is specified in the VARYING or AFTER phrase, the FROM phrase must specify a positive integer, and the BY phrase must specify a nonzero integer. The index-name is initialized as described with the SET statement and subsequently incremented or decremented as shown in figure 5-19.

WITH TEST Phrase

If the WITH TEST AFTER phrase is specified, the evaluation of the condition takes place after the execution of the last statement of the specified set of statements. The statements are executed at least once, regardless of the initial conditions. Incrementing of the controlling variables occurs after the test.

If the WITH TEST BEFORE phrase is specified, the evaluation of the condition takes place at the beginning of the PERFORM statement execution. If the condition is true at the start of PERFORM execution, the statements are not executed.

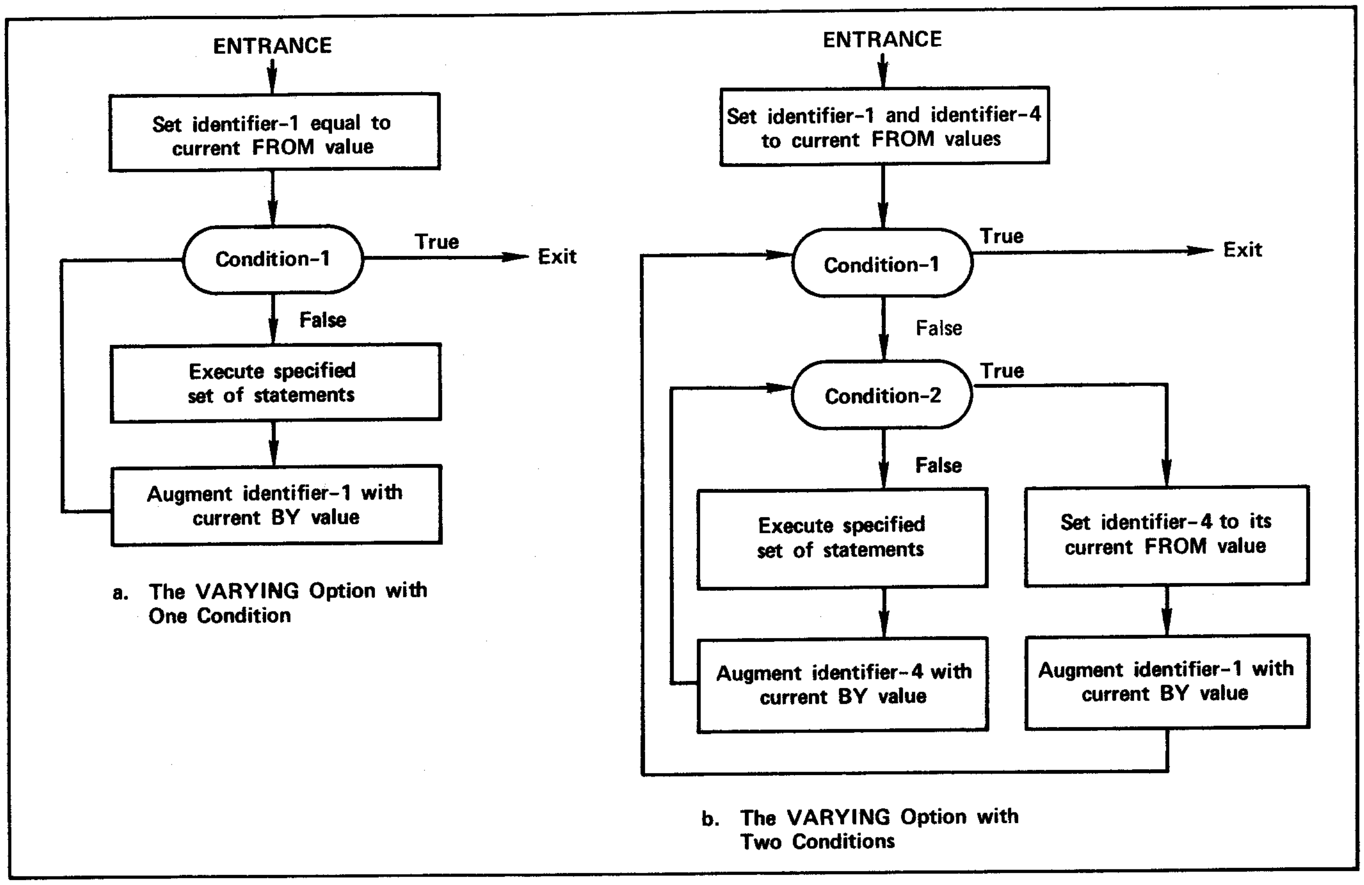

ENTRANCE
Set identifier-1 equal to current FROM value
Condition-1
True
Exit
False
Execute specified set of statements
Augment identifier-1 with current BY value
a. The VARYING Option with One Condition
ENTRANCE
Set identifier-1 and identifier-4 to current FROM values
Condition-1
True
Exit
False
Condition-2
True
False
Execute specified set of statements
Set identifier-4 to its current FROM value
Augment identifier-4 with current BY value
Augment identifier-1 with current BY value
b. The VARYING Option with Two Conditions

Figure 5-19. Format 4 PERFORM Flowchart

The values of identifiers at the completion of the PERFORM operation depend on the specification of the WITH TEST phrase:

If the WITH TEST AFTER phrase is specified, each identifier contains the same value it contained at the end of the most recent execution of the specified set of statements.

If the WITH TEST BEFORE is specified or if the WITH TEST phrase is not specified, each data item varied by an AFTER phrase contains the current value of the data item referenced by the identifier in the associated FROM phrase. The data item referenced by identifier-1 has a value that exceeds its last used setting by one increment or decrement value. If condition-1 is true when the PERFORM statement is entered, however, the value is that of the data item referenced by identifier-2.

PURGE Statement

The PURGE statement (figure 5-19.1) causes MCS to delete partial messages that are to be transmitted to specified destinations. This statement is only allowed under the NOS operating system.

When a PURGE statement is executed, the communication description (CD) output area referenced by cd-name must contain the symbolic names and the number of destinations receiving messages that were partially released by one or more SEND statements. The PURGE statement eliminates all messages awaiting transmission that do not have an EMI or EGI associated with them. Execution of the PURGE statement updates the STATUS KEY field of the CD output area, and possibly the ERROR KEY field, that corresponds with the specified destinations.

```
PURGE cd-name
```

Figure 5-19.1. PURGE Statement Format

READ STATEMENT

The READ statement makes a record available in the record area associated with the file specified. Both the file organization and the access mode determine which format of the READ statement is appropriate and which record is returned.

Format 1 must be used for all files in sequential access mode, and for any file in dynamic access mode when a record is to be accessed sequentially by position.

Format 2 must be used for all files in random access mode, and for any file in dynamic access mode when a record is to be accessed randomly by key.

Format 3 and format 4 are used to access data base records that are joined together in a relation. See section 14, Sub-Schema Facility.

Format 1 and format 2 of the READ statement are shown in figure 5-20.

```
Format 1

    READ file-name [NEXT] RECORD [INTO identifier]

        [; AT END imperative-statement]


Format 2

    READ file-name RECORD  [INTO identifier]

        [; KEY IS data-name]

        [; INVALID KEY imperative-statement]
```

Figure 5-20. READ Statement Format 1 and Format 2

The file must be open for INPUT or I-O. Any access mode is allowed. Execution of READ updates any FILE STATUS data item defined for the file.

File-name must be the name of a file described with an FD entry in the File Section. It can be a file described in a sub-schema. READ must not be used with files described by SD entries or files described by an FD entry with a REPORT clause.

When the records of a file are described with more than one Record Description entry, these records automatically share the same storage area. The contents of any data items that lie beyond the range of the current data record are undefined.

Format 1 READ

Format 1 retrieves the next record in the file identified by file-name. The NEXT phrase is required when the file is open in the dynamic access mode. The next record is affected by the file organization:

For relative files, the next existing record is retrieved. If the RELATIVE KEY clause is specified in the File-Control entry for the file, the key of the record read is returned to the key item.

For indexed files, the next record with the same key of reference is retrieved. The key of reference is established by an OPEN statement or by the value of the data-name item specified with either a START statement or the preceding format 2 READ statement. The final record occurrence for a particular value of the key of reference can be detected by checking for a value of 02 in the FILE STATUS data item. When the primary key is the key of reference and it is not embedded in the record, the data item referenced in the RECORD KEY clause is updated to contain the key value. When an alternate record key is the key of reference and duplicate alternate key values exist, the order in which records are retrieved is determined by the DUPLICATES phrase of the ALTERNATE RECORD KEY clause:

If the keyword ASCENDING appears, records are retrieved in ascending primary key order. The ASCENDING option must be specified if the alternate key is within a repeating group (T type records) and the file organization is the extended AAM format.

If the keyword ASCENDING is omitted, records are retrieved in the order in which they were written or rewritten.

For direct files, considerations are the same as for indexed files.

For actual-key files, considerations are the same as for indexed files.

For word-address files, groups of words of the length of the Record Description entry, including evaluation of any DEPENDING ON phrase, are retrieved. Valid data may or may not exist; no indication of invalid data is given. The program is responsible for determining the validity of any data read, since the READ statement simply transfers words from the current file position.

For sequential files, the next record is read. If the end of a tape reel is encountered during an operation, standard operating system procedures occur for swapping reels. The program does not process the reel swap.

NEXT Phrase

The NEXT phrase must be used for files open in dynamic access mode when records are to be retrieved sequentially. Processing is as described above. Otherwise, NEXT is documentary only.

INTO Phrase

The INTO phrase moves the record being read from the record area associated with file-name to the area specified by identifier. The record is available in both areas after READ execution. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before the record is moved to identifier.

The phrase can be used when the FD entry for the file is followed by one or more Record Description entries that specify a fixed length or variable-length record. When more than one Record Description entry exists for the file, however, the phrase cannot be used if any elementary level 01 record defines a numeric or numeric edited item.

The storage area associated with identifier and the record area associated with file-name must not be the same storage area. Identifier is filled with blanks if the size of the record is less than the size of identifier, unless the move is defined as an elementary numeric operation.

AT END Phrase

The AT END phrase specifies the imperative-statement to be executed when an at-end condition occurs. The condition occurs when:

A file referenced in an OPEN statement by an OPTIONAL phrase is not present when the first READ statement is executed.

No next record exists.

An attempt is made to read past the last word written to a word-address file.

A partition boundary (such as that written when a CLOSE statement is executed) is encountered, even though additional records might exist in the file.

The program is responsible for subsequent actions if file access is to continue.

Sequential files must be closed and successfully reopened.

Files with other organizations must be repositioned according to the subsequent processing desired, by using one of these statements:

CLOSE followed by a successful OPEN

START

Format 2 READ.

When the phrase is omitted, any applicable format 1 USE statement executes, as described with the FILE STATUS clause of the FILE-CONTROL paragraph in section 3.

Format 2 READ

Format 2 retrieves from file-name the record indicated by the value of a key. It is valid for all organizations except sequential when the file is open in the random or dynamic access mode. Format 2 also establishes the key of reference for indexed, direct, and actual-key files that are subsequently to be read through a format 1 READ statement.

The key item that governs the read is defined in the File-Control entry of the Environment Division. The key item depends on the file organization:

For relative files, the data-name of the RELATIVE KEY clause defines the key item. The value in the key item determines the record to be retrieved.

For word-address files, the data-name of the WORD-ADDRESS KEY clause defines the key item. The value in the key item determines the record to be retrieved.

For indexed, direct, and actual-key files, the data-name of the RECORD KEY clause defines the primary key. The data-names of one or more ALTERNATE RECORD KEY clauses define alternate record keys. Records are read by primary key unless the KEY IS phrase of the READ statement specifies read by an alternate record key value. The key item identified by the presence or absence of the KEY IS phrase specifies the key of reference. The value of the key of reference determines the record to be retrieved. If no record in the file has the same value as the key of reference, an invalid key condition exists and the key of reference is undefined.

INVALID KEY Phrase

The INVALID KEY phrase specifies the imperative-statement to be executed when an invalid key condition occurs. The condition exists when:

An attempt is made to read past the last word of a word-address file.

A key value of a relative file does not correspond to the value of a record in the file.

An existing record does not have the value specified by the key of reference for an indexed, direct, or actual-key file.

When the phrase is omitted, strict ANSI usage requires an appropriate format 1 USE statement for the file. See the FILE STATUS clause of the FILE-CONTROL paragraph in section 3.

INTO Phrase

The INTO phrase is the same as for format 1 READ.

KEY IS Phrase

The KEY IS phrase establishes the key of reference for the READ. It is valid only for indexed, direct, and actual-key files. When the phrase is omitted, the key of reference is the primary key.

The phrase can specify either of the following as the key of reference:

Data-name of the RECORD KEY clause.

Data-name of any ALTERNATE RECORD KEY clause.

An alternate record key reference cannot be subscripted or indexed; however, the alternate record key item can be described with the OCCURS clause. In this case, the alternate record key is within a repeating group. The first key item in the repeating group in the record area is used for comparison purposes. The alternate key value in the record retrieved can be any occurrence of the repeating group.

RECEIVE Statement

The RECEIVE statement (figure 5-20.1) causes a message, a message segment, or a portion of a message or message segment to be returned from a specified MCS queue. If no message text is available in the queue, an imperative statement specified in the RECEIVE statement is executed; if no imperative statement is specified in the RECEIVE statement, the execution of the program containing the RECEIVE statement is suspended. This statement is allowed only under the NOS operating system.

```
RECEIVE cd-name { MESSAGE }  INTO identifier-1
                { SEGMENT }

       [ ; NO DATA imperative-statement]
```

Figure 5-20.1. RECEIVE Statement Format

When a RECEIVE statement is executed, the communication description (CD) input area referenced by cd-name must contain one or more symbolic input queue names. These names identify the simple or compound queue that contains the next returned message or message segment. When a simple queue is accessed, message text is returned only if the queue already contains a complete message or the queue contains an incomplete message because part of it was returned to the same program during the execution of a previous RECEIVE statement. A message is not qualified to be returned during execution of the RECEIVE statement if the message has not been completely received by MCS. When a compound queue is referenced, the simple queues within the compound queue are searched in the order specified in the Application Definition Language. Message text is returned from the first simple queue that contains message text available to the program in the order specified in the Application Definition Language.

If the execution of a RECEIVE statement causes message text to be returned to the program, the MESSAGE DATE, MESSAGE TIME, SYMBOLIC SOURCE, TEXT LENGTH, END KEY, and STATUS KEY fields of the CD input area are updated during execution of the statement. If a compound queue is accessed, the symbolic queue name(s) of the simple queue with the stored message being returned is entered in the CD input area. Next, control is transferred to the executable statement that follows the RECEIVE statement, regardless of the presence of the NO DATA phrase. A message or message segment can be a null message or null message segment. A null message or null message segment does not contain any message characters. When a null message or null message segment is returned during execution of a RECEIVE statement, message text is considered to have been returned and all the fields previously mentioned are updated as normal; the TEXT LENGTH field is set to zero.

One of the following two actions can occur if a RECEIVE statement executes successfully (the STATUS KEY field contains the value 00 following execution), but no message text is available in the simple queue or queues accessed during execution of the statement:

> If the NO DATA phrase is not specified, the execution of the program is suspended until a message is available in the simple or compound queue accessed by the RECEIVE statement, and until this message or part of the message is returned to the program. The one exception to this rule occurs when a program is still in this suspended state during shutdown of MCS or the MCS application. In this case, the STATUS KEY field of the CD input area is updated (indicating the shutdown condition) and control is transferred to the next executable statement.
>
> When the NO DATA phrase is specified, the TEXT LENGTH and STATUS KEY fields of the CD input area are updated, and the imperative statement in the NO DATA phrase is executed.

If a RECEIVE statement does not execute successfully, only the STATUS KEY field is updated and control is transferred to the next executable statement (whether or not the NO DATA phrase is specified).

A single execution of a RECEIVE statement causes return of either a message (when the MESSAGE phrase is used), or a message segment (when the SEGMENT phrase is used). During execution of a RECEIVE statement with the MESSAGE phrase, an EMI or EGI indicates a message boundary, and any ESIs are ignored. During execution of a RECEIVE statement with the SEGMENT phrase, an ESI, EMI, or EGI indicates a segment boundary. The message or segment returned is entered in the receiving area referenced by identifier-1, starting at the leftmost character position of the receiving area. End indicators are not entered in the receiving area as part of the message text, but are stored in the END KEY field of the CD input area. All characters transferred to the receiving area are in display code; no editing takes place while the characters are being entered.

If the area has more character positions than the number of characters contained in the message or segment returned, the unused character positions are not space filled. In case the number of characters in the message or segment is greater than the number of character positions in the receiving area, the receiving area is completely filled and the END KEY is set. The remaining portion of the message or segment can be returned to the program during one or more subsequent executions of RECEIVE statements. If part of a message has been returned to a program, the remainder of the message can only be accessed by that same program and only if the next RECEIVE statement uses the same queue names. After the execution of a STOP RUN statement by a program, remaining portions of messages partially returned to that program are purged by MCS.

RELEASE STATEMENT

The RELEASE statement transfers records to the initial phase of a SORT operation. See section 7, Sort/Merge Facility.

RETURN STATEMENT

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records from the final phase of a MERGE operation. See section 7, Sort/Merge Facility.

REWRITE STATEMENT

The REWRITE statement (figure 5-21) replaces an existing record in a file on mass storage. The file can have any organization except word-address. It can be a file described in a sub-schema.

```
REWRITE record-name [FROM identifier]

      [; INVALID KEY imperative-statement]
```

Figure 5-21. REWRITE Statement Format

REWRITE releases the record specified by record-name. The record is no longer available in the record area unless the file is named in a SAME RECORD AREA clause. If so, the record is available to the program as a record of other files specified in that clause.

The file must be open for I-O; any access mode is allowed. Both the access mode and the file organization affect the phrase that can be specified and the results obtained. Execution of REWRITE updates any FILE STATUS data item defined for the file.

Record-name, which can be qualified, must be a record named in a level 01 entry in the File Section of the Data Division. Record-name and identifier must not refer to the same storage area.

For sequential files, the record type for a record being rewritten must be either F (fixed length) or W (control word). The replacing record must have the same number of character positions as the record being replaced. For other file organizations, the replacing record need not be the same size.

For a file open in the random or dynamic mode, the replacing record is identified by the value of the data item specified by the RELATIVE KEY or the RECORD KEY clause for the file associated with record-name. The value must be the same as a record existing in the file. The contents of any data items defined by ALTERNATE RECORD KEY clauses can differ from those in the record being replaced; however, the values of alternate record keys must not duplicate existing alternate record key values in the file unless the DUPLICATES phrase was specified when the alternate record key was defined.

For a file open in the sequential access mode, the last input-output statement preceding REWRITE must be a format 1 READ. REWRITE then replaces the record previously read. For an indexed, direct, or actual-key file, the primary key is used to identify the record to be replaced; it must be the same as the primary key value of the record just read.

FROM Phrase

Execution of a REWRITE statement with the FROM phrase is equivalent to execution of:

MOVE identifier TO record-name

REWRITE record-name

Record-name and identifier must not reference the same storage area. The contents of the record area prior to the execution of the implicit MOVE statement has no effect on REWRITE execution.

INVALID KEY Phrase

The INVALID KEY phrase can be used only for indexed, direct, and actual-key files open in any access mode and for relative files open in random or dynamic access mode. The imperative-statement is executed when any of the following invalid key conditions exist:

The file does not contain the record specified by the record key.

The primary key has been changed between the READ and REWRITE for files open in the sequential access mode.

The DUPLICATES condition for alternate record keys has been violated.

When the invalid key condition exists, the updating operation does not take place and the data in the record area is not affected.

When the phrase is omitted, strict ANSI usage requires an appropriate format 1 USE statement. See the FILE STATUS clause of the FILE-CONTROL paragraph in section 3.

SEARCH STATEMENT

The SEARCH statement (figure 5-22) searches a table for an element that satisfies the specified conditions and adjusts an index-name to point to that element. The type of search and the conditions on which the search ends depend on the format used.

Format 1 is a sequential search that stops when any specified condition is satisfied.

Format 2 is a binary search that stops only when all specified conditions are satisfied.

```
Format 1

SEARCH identifier-1 [VARYING {index-name-1}]  [; AT END imperative-statement-1]
                             {identifier-2}

   ; WHEN condition-1 {imperative-statement-2}  [; WHEN condition-2 {imperative-statement-3}] ...
                      {NEXT SENTENCE         }                     {NEXT SENTENCE         }

   [; END-SEARCH]


Format 2

SEARCH ALL identifier-1 [; AT END imperative-statement-1]

              {                {EQUALS     }    {identifier-3           }}
              { data-name-1    {IS EQUAL TO}    {literal-2              }}
   ; WHEN     {                {IS =       }    {arithmetic-expression-1}}
              {                                                          }
              { condition-name-1                                         }

   [          {                {EQUALS     }    {identifier-4           }}]
   [          { data-name-2    {IS EQUAL TO}    {literal-3              }}]
   [ AND      {                {IS =       }    {arithmetic-expression-2}}] ...
   [          {                                                          }]
   [          { condition-name-2                                         }]

   {imperative-statement-2}  [; END-SEARCH]
   {NEXT SENTENCE         }
```

Figure 5-22. SEARCH Statement Format

The table to be searched is specified by identifier-1. The table must be described by an OCCURS clause containing an INDEXED BY phrase that defines index-names. Identifier-1 must not be indexed, reference modified, or subscripted in the SEARCH statement.

Only one dimension of a multidimensioned table is searched during SEARCH statement execution. (If identifier-1 is a data item subordinate to a data item containing an OCCURS clause, the table has more than one dimension.) A SEARCH statement must be repeated to search a second dimension. SET statement execution can adjust index-names to values appropriate for different dimensions.

The inclusion of the END-SEARCH explicit scope terminator makes the SEARCH statement into an imperative (rather than conditional) statement and facilitates the nesting of SEARCH statements.

Format 1 SEARCH

Format 1 is a sequential search that begins at the current value of the search index-name and that ends when one of the specified conditions is satisfied or the table ends. A SET statement often precedes the SEARCH statement to initialize the table index-name, even when another index-name is used for the search. The VARYING phrase establishes the index-name to be used. The search takes place only if the value of the search index-name is within the limits of the table elements.

The search operates by evaluating the conditions in the order written, making use of the index settings to determine the occurrence of those items to be tested. The process repeats, as necessary, using an incremented index-name, until a condition is satisfied; the imperative-statement associated with the condition is then executed.

At the conclusion of the search, the index-name used in the search is always set to the occurrence of the element in the table that satisfies the condition. If no condition is satisfied, index-name contains one more than the highest table element occurrence.

If the entire table is searched and no condition is satisfied, control passes to any AT END imperative-statement or the next executable sentence.

VARYING Phrase

The VARYING phrase specifies which index-name of identifier-1 is to be used in the search. It can also specify an item to be incremented each time the index-name associated with identifier-1 is incremented. When the phrase is omitted, the first index-name specified in the INDEXED BY phrase of the OCCURS clause for identifier-1 is used in the search.

index-name-1

When index-name-1 appears in the INDEXED BY phrase of the OCCURS clause for identifier-1, index-name-1 is the item used for the search.

When index-name-1 appears as an index-name for a table other than identifier-1, the occurrence number represented by index-name-1 is incremented by the same amount as the occurrence number represented by the index-name associated with identifier-1. Incrementing for both items takes place at the same time.

identifier-2

When identifier-2 is an index data item, it is incremented by the same amount as, and at the same time as, the index associated with identifier-1 is incremented.

When identifier-2 is not an index data item, it is incremented by 1 each time the index-name associated with identifier-1 is incremented.

Identifier-2 must be described as USAGE IS INDEX or as an integer data item; it must not be described with COMP-2.

AT END Phrase

The AT END phrase specifies the imperative-statement to be executed when:

The index-name value is outside the table limits at the start of the search.

No condition is satisfied during the search.

When the phrase is omitted, control passes to the next executable sentence if either of these instances occurs.

Format 2 SEARCH

Format 2 SEARCH is a binary search for a table element that satisfies one or more conditions. A successful search ends when all specified conditions are satisfied by a table element. The search is unsuccessful if a table element that satisfies the conditions cannot be found. Format 2 should be used in preference to format 1 when the table to be searched is large.

Identifier-1 must be described with the KEY IS phrase of the OCCURS clause as well as by the INDEXED BY phrase. The data-names in the KEY IS phrase are used in the WHEN and AND phrases of SEARCH to establish the conditions on which the search is successful.

When a condition-name is referenced in a WHEN or AND phrase, the condition-name must have only a single value. The data-name associated with condition-name must appear in the KEY IS phrase that describes identifier-1. Data-names can be qualified.

When a data-name is referenced in a WHEN or AND phrase, each data-name referenced must be indexed by the first index-name associated with identifier-1 along with other indexes or literals as required, and must be referenced in the KEY IS phrase that describes identifier-1.

When either a data-name or a condition associated with data-name is referenced, all preceding data-names in the KEY IS phrase of identifier-1, or their associated condition-names, must also be referenced.

Identifiers referenced in the WHEN phrase and identifiers specified in an arithmetic expression must not be referenced in the KEY IS phrase of identifier-1. Neither can they be indexed by the first index-name of identifier-1.

The results of a format 2 SEARCH operation are predictable only when the data in the table is ordered in the same manner as described by the ASCENDING/DESCENDING KEY phrase of identifier-1 and the contents of the key referenced in the WHEN phrase are sufficient to identify a unique table element. At the end of a successful search, the index-name indicates an occurrence that allows all conditions to be satisfied. Control then passes to the imperative-statement associated with the specified conditions.

If the binary search of the table cannot satisfy all conditions, control passes to any AT END imperative-statement or to the next executable sentence. The final setting of the index is not predictable for an unsuccessful search.

Figure 5-23 shows the flow of control when three conditions are specified.

SEND Statement

The SEND statement (figure 5-23.1) causes a message, message segment, or a portion of a message or message segment to be released to one or more specified destinations maintained by MCS. MCS enqueues the message text in one or more output queues and determines, after the message is complete, when to initiate transmission of the message. The SEND statement is only allowed under the NOS operating system.

When the SEND statement is executed, the communication description (CD) output area referenced by cd-name must contain the symbolic names and the number of destinations receiving the message being released. Symbolic destinations are defined in the application definition and can include terminals, journals, interprogram queues, application name, or broadcast lists. If message text is released, the TEXT LENGTH field of the CD output area must indicate the number of

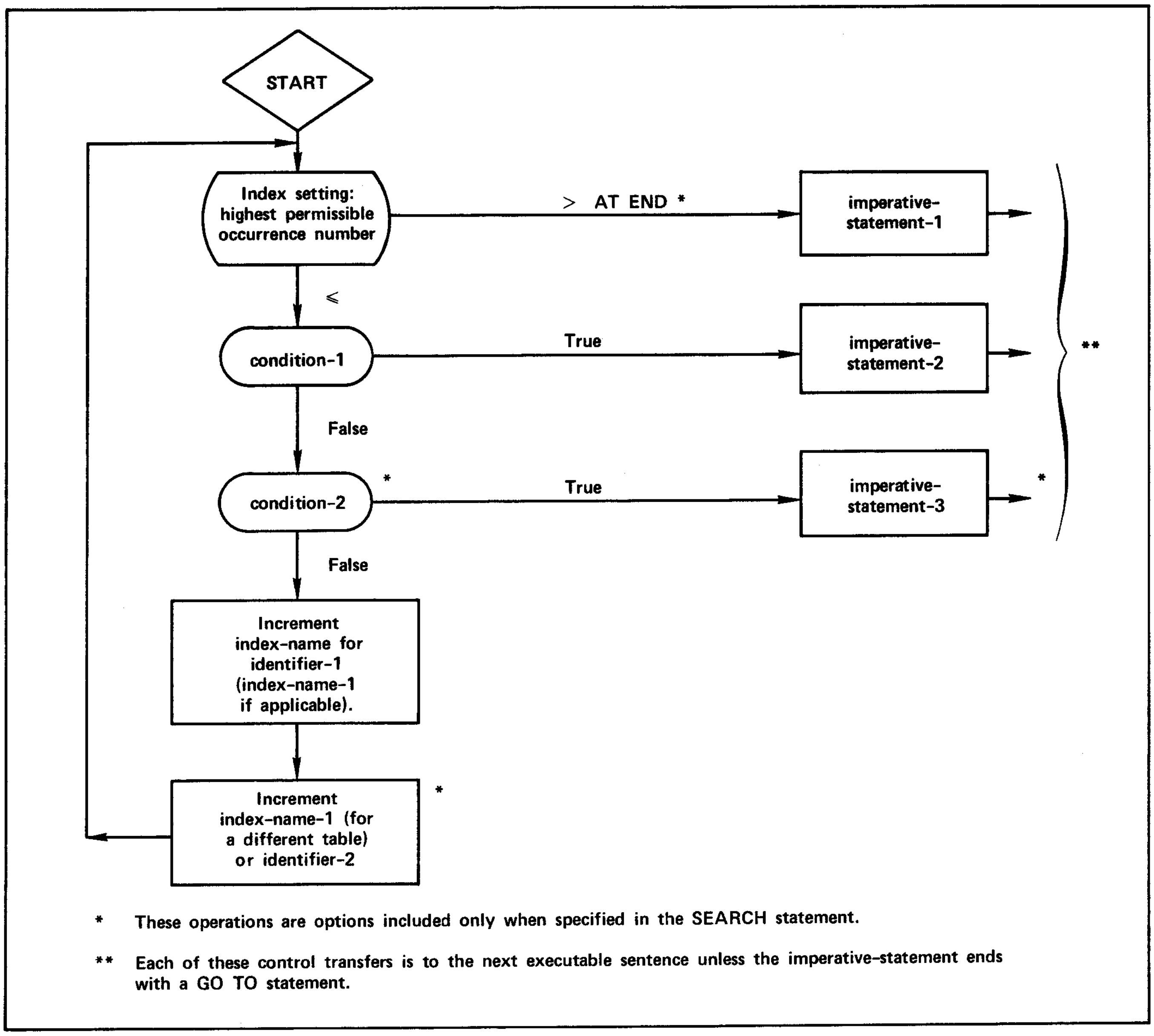

START
Index setting: highest permissible occurrence number
> AT END *
imperative-statement-1
≤
condition-1
True
imperative-statement-2
False
condition-2
*
True
imperative-statement-3
*
**
False
Increment index-name for identifier-1 (index-name-1 if applicable).
Increment index-name-1 (for a different table) or identifier-2
*
* These operations are options included only when specified in the SEARCH statement.
** Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 5-23. Flowchart of SEARCH Operations

```
Format 1

     SEND cd-name FROM identifier-1

Format 2

                                          ( WITH identifier-2 )
     SEND cd-name [FROM identifier-1]     ) WITH ESI          (
                                          ) WITH EMI          (
                                          ( WITH EGI          )

     [                         ( ( [identifier-3] )   [LINE ] )  ]
     [ { BEFORE }  ADVANCING   ) ( [integer]      )   [LINES] )  ]
     [ { AFTER  }              ) { mnemonic-name }               ]
     [                         ( { PAGE          }            )  ]
```

Figure 5-23.1. SEND Statement Format

character positions of the message area that is referenced by identifier-1 and that is to be transferred. These message characters are display code characters transferred from the leftmost character position of the message area. Execution of the SEND statement updates the STATUS KEY field of the CD output area, and possibly the ERROR KEY field that corresponds with the specified destinations.

An end indicator referenced in the WITH phrase of a SEND statement indicates the status of any message text released by that statement, and/or of any message text with no associated end indicator released by previous executions of SEND statements in the same program. The end-of-segment indicator (ESI) indicates that the message segment is complete; the end-of-message indicator (EMI) indicates that the message is complete and implies an ESI; the end-of-group indicator (EGI) indicates the group of messages is complete and implies an ESI and EMI. If the WITH phrase references identifier-2, then the contents of identifier-2 indicates the type of end indicator to be associated with the message text. A value of 1 is an ESI; 2 is an EMI; 3 is an EGI. A value of 0, or any value other than 1, 2, or 3, means no indicator is associated with the message text, that causes the SEND statement to release a portion of a message or message segment. If a value of other than 1, 2, or 3 is used, identifier-1 must be specified and the second format of the SEND statement with identifier-2 has the same effect as the first format.

A partial message released to MCS can only be completed by subsequent executions of SEND statements in the same program. MCS does not transmit any part of a message to its destination(s) until the entire message has been released to MCS. Partial messages released by a program, but not terminated by an EMI or EGI at the time the program executes a STOP RUN statement, are purged by MCS. It is possible, using the second format of the SEND statement, to release null messages or message segments to MCS. A null message or segment is created whenever a program releases an EGI, EMI, or ESI with no associated message text (the text length is zero or has no identifier-1).

The ADVANCING phrase controls the vertical positioning of a message or message segment at a destination where positioning is applicable. The ADVANCING phrase is ignored when specified in a SEND statement that releases a portion of a message or message segment (does not release an end indicator). At destinations where positioning applies, both a message and message segment begin at the leftmost character position of the line that includes the message or message segment being printed. Additional lines can be used to print a message or message segment that is longer than the physical line size of the output device.

If integer or identifier-3 is specified in the ADVANCING phrase, the message or message segment is printed at the destination(s) before or after (depending on the phrase used) downward repositioning occurs and for as many lines as indicated by the value of integer or the content of identifier-3. If the content of identifier-3 or integer is zero, no repositioning occurs before or after the message or segment is printed. If PAGE is specified, printing of the message or segment occurs before or after repositioning to the next physical page. If PAGE is specified on a CRT device, displaying occurs before or after clearing the screen and returning the cursor.

A specified mnemonic-name must be an implementor-name defined as a 1-character alphanumeric literal in the SPECIAL-NAMES paragraph of the Environment Division. Table 5-2.1 defines the characters allowed in this literal and the effect of the characters on vertical positioning. The phrases BEFORE and AFTER have no effect if a mnemonic-name is used; the positioning information is entirely derived from the character used in the implementor-name literal. If a character that does not appear in the table 5-2.1 is used in this literal, the effect is the same as if a space had been used in the literal.

If specified, the ADVANCING phrase is effective for null messages and message segments. If a message or message segment is released by a SEND statement that does not specify the ADVANCING phrase, and the message or segment is to be transmitted to a destination where positioning is applicable, positioning is provided as if the statement specified AFTER ADVANCING 1 LINE.

When a SEND is used with a special destination (the MCS application name), the text is submitted to MCS where it is processed as a command. The message must consist of a single segment and must be a valid MCS command as defined in User Commands, section 7 of the Message Control System reference manual. The MCS response to a message sent by a program is stored in a response queue. (See section 3 of the Message Control System reference manual.)

TABLE 5-2.1. VERTICAL POSITIONING
(SEND Statement)

Character	Effect
space	Space 1 line before printing.
0	Space 2 lines before printing.
-	Space 3 lines before printing.
+[†]	Position to start of current line before printing.
*[†]	Position to new page or home cursor before printing.
1	Position to new page, or clear screen and home cursor, before printing.
.[†]	Space 1 line after printing.
/[†]	Position to start of current line after printing.
,[†]	Perform no action, print at next available position. The message or segment may not begin at the left-most character position of the line when this character is used.
Any other ASCII Character	Space 1 line before printing.

[†]Terminal dependent; if not allowed at the terminal, the character has the same effect as a space or a blank.

Segments longer than 4100 characters are truncated when sent to terminals; however, if the text is divided into multiple segments, no truncation occurs.

Using a SEND for a destination that is a broadcast list holds the message in the associated output queue until at least one list member is connected and eligible to receive the message. At that time the message is transmitted to all connected terminals. The output for all other list members is discarded.

SET STATEMENT

The SET statement (figure 5-24) performs several different functions, depending on the format used:

Format 1 sets one or more index-names (defined by INDEXED BY phrase of OCCURS clause), index data items, or elementary integer data items to the number of a table element.

Format 2 increases or decreases the value of one or more index-names.

Format 3 establishes a collating sequence or character code set.

Format 4 alters switch settings.

Format 5 alters the value of conditional variables.

In format 1 and format 2 of the SET statement, sending and receiving operands that overlap cause undefined results.

Format 1 SET

Format 1 sets each operand specified after the keyword SET, in turn, to the value of the operand specified after the keyword TO. Operands that can be specified as the sending and receiving items are shown in table 5-3.

TABLE 5-3. ALLOWED FORMAT 1 AND FORMAT 2 SET STATEMENT OPERATIONS

Sending Item	Receiving Item		
	Integer Data Item	Index-Name	Index Data Item
Integer literal	No	Yes	No
Integer data item	No	Yes	No
Index-name	Yes	Yes	Yes
Index data item	No	Yes	Yes

Before format 1 execution, the value of any index-name-3 must correspond to an occurrence number of an element in the table in which index-name-3 is defined by the INDEXED BY phrase of the OCCURS clause. After execution, the value of any index-name-1 must show the same correspondence to an element in its associated table.

Integer-1, which can be signed, must have a value greater than 0.

The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement might be undefined. If a SEARCH with the ALL phrase results in execution of the AT END imperative-statement, the associated index is undefined. At the end of a PERFORM sequence with the VARYING and UNTIL phrases, the index-name might be undefined if the condition specified with UNTIL does not involve the index or its table.

Each identifier following SET is evaluated for subscripting or indexing immediately before its value is changed.

Format 2 SET

Format 2 increments (UP BY) or decrements (DOWN BY) the value of each index-name specified. Integer-2 can be signed; identifier-4 must be an elementary numeric integer item. The value of the index-names must correspond to an occurrence number of an element in their associated tables before and after SET execution. Otherwise, format 2 processing is as described for format 1, with table 5-3 above specifying the valid operand items.

Format 1

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

Format 2

```
SET index-name-4  [, index-name-5] . . . { UP BY   } { identifier-4 }
                                         { DOWN BY } { integer-2    }
```

Format 3

```
    { { SORT       }                                            }
    { { MERGE      } COLLATING SEQUENCE                         }
    { { SORT-MERGE }                                            }
SET { { PROGRAM    }                                            } TO alphabet-name
    {                                                           }
    { CODE-SET FOR { file-name-1  [, file-name-2] . . . }       }
    {              { ALL FILES                          }       }
```

Format 4

```
SET { mnemonic-name-1  [, mnemonic-name-2] . . . TO { ON  } } . . .
                                                    { OFF }
```

Format 5

```
SET condition-name TO TRUE
```

Figure 5-24. SET Statement Format

Format 3 SET

Format 3 phrases establish a collating sequence or code set. Alphabet-name must be specified in an ALPHABET clause of the SPECIAL-NAMES paragraph in the Environment Division. Phrases result in the following:

SORT or MERGE COLLATING SEQUENCE

SORT-MERGE COLLATING SEQUENCE

Alphabet-name is the collating sequence used for subsequent sort or merge operations.

PROGRAM COLLATING SEQUENCE

Alphabet-name is the program collating sequence used for subsequent nonnumeric comparisons. It is also used for subsequent sort or merge operations unless a following SET statement specifies a different collating sequence for sort and/or merge.

CODE-SET FOR

Alphabet-name is the code set for files on tapes in UNIVAC 1100 series FIELDATA format. The phrase is not valid for other data formats. The alphabet specified is used during execution of READ, REWRITE, and WRITE statements; it is not used during OPEN or CLOSE since the phrase does not take effect until after the file is opened. The file-names specified should be FIELDATA format tape files. In the SPECIAL-NAMES paragraph, alphabet-name must be associated with UNI.

This phrase is an alternative to the CODE-SET clause of the FD entry.

Format 4 SET

Format 4 sets mnemonic-name to ON or OFF. Switches remain set until they are changed by another SET statement, by operator action at any time, by the terminal user during a program pause (a STOP literal statement), or by a SWITCH control statement between job steps. The switch mnemonic-names must be defined in the SPECIAL-NAMES paragraph with a SWITCH-n IS mnemonic-name clause.

Switches 1 through 6 can be set initially by control statement or operator action prior to the current job step. Any of the switches 1 through 6 set within the program affects subsequent job steps. Switches 7 through 126 do not affect any other job steps.

Format 5 SET

Format 5 sets a condition variable to a value that causes condition-name to be true. Condition-name must be associated with a conditional variable. The literal in the VALUE clause associated with condition-name is moved to the conditional variable in accordance with the rules for elementary moves. If more than one literal is specified in the VALUE clause, the first literal in the VALUE clause is moved.

SORT STATEMENT

The SORT statement sorts records. See section 7, Sort/Merge Facility.

START STATEMENT

The format 1 START statement (figure 5-25) positions a relative, indexed, direct, or actual-key file for subsequent sequential retrieval of records. The file must be open in the INPUT or I-O mode; access mode must be sequential or dynamic.

The format 2 START statement is used for files described in a sub-schema. Refer to section 14.

Execution of the START statement establishes the key of reference by positioning the index file to the first alternate key value that meets the specified condition. The search begins either from the current key of reference or from the beginning of the index file if the key of reference has not yet been established. The key that satisfies the condition becomes the new key of reference. If the comparison is not satisfied by any alternate key value, the INVALID KEY clause is executed.

Successful execution of READ INVALID KEY or START establishes a key of reference for purposes of future access to the file. The key of reference is the primary or alternate key of the record read or located. Once the key of reference has been established, it can only be changed by execution of another START, READ, or OPEN statement, or by execution of the INVALID KEY phrase. Refer to the Multiple-Index Processor (MIP) user's guide for further detail.

Execution of START updates any FILE STATUS data item defined for the file. It establishes the current position of the file associated with file-name as the beginning of the first record with a key satisfying the comparison.

For indexed, direct, and actual-key files, the key of reference for subsequent format 1 READ statements is established by the START statement. The key of reference is determined by the KEY phrase:

> If the KEY phrase is omitted, the key of reference is the primary key. (However, the phrase cannot be omitted for direct and actual-key file organizations.)
>
> If the KEY phrase specifies a primary key, the key of reference is that primary key. (However, the primary key can be specified for indexed file organizations only.)
>
> If the KEY phrase specifies an alternate record key, the key of reference is that alternate record key.
>
> If the KEY phrase specifies a leading portion of a primary key or alternate record key, the key of reference is the primary key or alternate record key corresponding to the leading portion.

KEY Phrase

The KEY phrase specifies the data item to be compared with keys of records existing in the file. Data-name can be qualified. When the phrase is omitted, the relational operator defaults to EQUAL and data-name defaults to the data item described by the RELATIVE KEY or RECORD KEY clause appropriate for the file organization. However, the phrase cannot be omitted for direct and actual-key file organizations. If the KEY phrase specifies an alternate key that is within a repeating group, indexing or subscripting must not be used. The value to be used for comparison is in the first occurrence of the repeating group in the record area. The matching value can be in any occurrence of the repeating group in the record on the file.

The type of comparison is specified by the relational operator in the KEY phrase. When the key is alphanumeric and the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter operand. All other numeric or nonnumeric comparison rules apply. Nonnumeric comparisons are made with respect to the program collating sequence that was in effect when the file was first opened for output.

The KEY phrase can specify one of three conditions, each of which can be stated in alternative words:

> IS EQUAL TO or EQUALS or IS =
>
> IS GREATER THAN or EXCEEDS or IS
>
> IS NOT LESS THAN or IS NOT

The particular data-name that can be used depends on the file organization declared in the SELECT clause of the FILE-CONTROL paragraph in the Environment Division.

Format 1

```
                          ┌      ⎧ IS EQUAL TO        ⎫           ┐
                          │      ⎪ EQUALS             ⎪           │
                          │      ⎪ IS =               ⎪           │
START file-name           │ KEY  ⎨ EXCEEDS            ⎬ data-name │
                          │      ⎪ IS GREATER THAN    ⎪           │
                          │      ⎪ IS >               ⎪           │
                          │      ⎪ IS NOT LESS THAN   ⎪           │
                          └      ⎩ IS NOT <           ⎭           ┘

[; INVALID KEY imperative-statement]
```

Figure 5-25. START Statement Format 1

For relative files, data-name must specify the data item defined by the RELATIVE KEY clause.

For direct and actual-key files, data-name cannot specify the RECORD KEY data item (that is, the primary key). Data-name must be an alternate record key or the leading portion of an alternate record key.

For indexed, direct, and actual-key files, data-name can specify the data item defined by:

> The ALTERNATE RECORD KEY clause. If data-name appears in an entry that contains an OCCURS clause or is subordinate to an OCCURS clause, data-name cannot be subscripted or indexed; the first occurence is always implied.
>
> A category alphanumeric data item subordinate to the ALTERNATE RECORD KEY data item whose leftmost character position corresponds to the leftmost character position of the ALTERNATE RECORD KEY data item. If two ALTERNATE RECORD KEY data items are defined as starting in the same character position, no item subordinate to them can be specified.

For indexed files, data-name can also specify a data item defined by:

> The RECORD KEY clause (that is, the primary key).
>
> A category alphanumeric data item subordinate to the RECORD KEY data item whose leftmost character position corresponds to the leftmost character position of the RECORD KEY data item.

INVALID KEY Phrase

The INVALID KEY phrase specifies the imperative-statement that is executed if the comparison is not satisfied by any record in the file. In this instance, the key of reference is undefined.

STOP STATEMENT

The STOP statement (figure 5-26) temporarily or permanently suspends program execution.

> STOP RUN permanently stops the program and returns control to the operating system. In MCS under NOS, execution of a STOP RUN statement purges any portion of a message transferred from the COBOL program through a SEND statement, but not terminated by EGI or EMI (no portion of the message is sent). The STOP RUN statement also purges any portion of a message partially obtained through a RECEIVE statement.
>
> STOP literal puts the program in a PAUSE state, with execution resuming at operator command.

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \underline{\text{RUN}} \\ \text{literal} \end{array} \right\}$$

Figure 5-26. STOP Statement Format

STOP RUN

STOP RUN is the normal way to terminate a program. End-of-program processing, including the close of any open files, occurs when the statement is executed. If any files are left open, except those opened by ACCEPT or DISPLAY, an attempt is made to close the files. If a file can be closed, the message FILE lfn NOT CLOSED AT STOP RUN - CLOSED BY COBOL is given and the file is closed as if CLOSE file-name had been issued. If the file cannot be closed for any of the following reasons, then the message FILE lfn NOT CLOSED AT STOP RUN AND CANNOT BE CLOSED is given and the file is left open (buffer might not be cleared):

> Valid FIT cannot be located, because the file was opened by an overlay that is no longer loaded or by a subprogram that was cancelled.
>
> Close routine not available because of one of the above reasons for not locating a valid FIT, or because the file was opened in an overlay or dynamic subprogram that does not contain the STOP RUN statement.

STOP literal

STOP literal displays the literal for operator action. If the program is executing as part of a batch job, the literal is displayed at the central site operator console. If the program is executing as a result of an interactive terminal command, the literal is displayed at the terminal. In both instances, the program does not resume execution until the operator takes an appropriate action.

The literal can be an unsigned integer, any other numeric literal, or a nonnumeric literal. As many as 60 characters can be displayed; excess characters are truncated on the right. A figurative constant appears as a single character. The following characters display as blanks at the central site: $ % > ; ≠ < ∨] .

The message received by the operator is:

> PAUSE Δ Δ Δ Δ Δ Δ Δ Δ literal

For the central site operator, the first 27 characters of the literal appear on the PAUSE line; any remaining characters appear on a second line.

The central site operator can respond with any of the commands available to the operator. Program execution does not resume, however, until a GO command is entered.

The terminal operator can enter any message. The message is ignored and program execution continues.

If the RERUN ON condition-name phrase was specified in the I-O-CONTROL paragraph, the switch status is tested after returning from the pause.

STRING STATEMENT

The STRING statement (figure 5-27) juxtaposes the partial or complete contents of two or more data items into a single data item. Transfer of characters to the receiving item occurs as described with the MOVE statement for elementary alphanumeric-to-alphanumeric moves without space fill. When a sending item and a receiving item share a part of their storage areas, the results are unpredictable.

```
       { identifier-1 } [ , identifier-2 ]                    { identifier-3 }
STRING { literal-1    } [ , literal-2    ] . . . DELIMITED BY { literal-3    }
                                                              { SIZE         }

   [   { identifier-4 } [ , identifier-5 ]                    { identifier-6 } ]
   [ , { literal-4    } [ , literal-5    ] . . . DELIMITED BY { literal-6    } ] . . .
   [                                                          { SIZE         } ]

   INTO identifier-7  [WITH POINTER identifier-8]  [; ON OVERFLOW imperative-statement]
```

Figure 5-27. STRING Statement Format

NOTE

Refer to appendix F for recommendations on the use of the STRING statement.

The sending items immediately precede the keywords DELIMITED BY. Literal sending items must be nonnumeric literals or figurative constants without ALL; identifier sending items must contain data with a USAGE of DISPLAY. Figurative constants are implicit one-character items.

The receiving item is identifier-7. It must be described as USAGE IS DISPLAY. Identifier-7 can be a group item or an elementary data item without editing symbols. The STRING statement does not initialize or fill identifier-7; any character position that does not receive a sending character is not affected by statement execution.

None of the identifiers can reference boolean data items. Identifier-7 cannot be reference modified.

The number of characters moved from the sending items depends on the DELIMITED BY phrase. SIZE specifies each preceding sending item is to be transferred in its entirety. Literal-3 or the contents of identifier-3 within a sending item stops the transfer. The delimiting literal must be nonnumeric and can be a figurative constant without ALL. Identifier-3 must be described with USAGE IS DISPLAY. A figurative constant is an implicit one-character item.

Transfer of characters to the receiving item begins at the leftmost character of the first sending item and stops when the character string specified by DELIMITED BY occurs or the item ends. The delimiting character string is not transferred.

Transfer resumes from the leftmost character of the second and subsequent sending items, with transfer stopping when the delimiter is encountered or the item ends. Transfer from all sending items stops when the receiving item is filled or all sending items are exhausted.

POINTER Phrase

The POINTER phrase specifies the character position within the receiving item at which the transfer is to begin. If the phrase is omitted, transfer is to the leftmost position in the item, character one. Each time a character is transferred to the receiving item, the item associated with POINTER is incremented by one. Consequently, identifier-8 must be an elementary integer item of sufficient size to contain a value equal to the size of the receiving item, plus one. The initial value of identifier-8, which must be set by the program, must not be less than one.

At the end of STRING execution, identifier-8 will have been incremented by the number of characters transferred to the receiving item. Only the portion of the receiving item that was referenced during transfer is changed. All other portions of the receiving item remain as they were before STRING was executed.

ON OVERFLOW Phrase

The ON OVERFLOW phrase specifies the imperative-statement that is executed if the receiving item is filled before the sending items are exhausted. The imperative-statement is also executed if the POINTER item value is less than one or greater than the number of characters in the receiving item. In the absence of the phrase, control passes to the next executable statement when an overflow condition exists.

SUBTRACT STATEMENT

The SUBTRACT statement (figure 5-28) subtracts elementary numeric data items and sets the result in a specified data item. The data item that receives the result depends on the format used:

Format 1 stores the result in one of the operand identifiers.

Format 2 stores the result in an identifier other than one of the operands.

Format 3 subtracts corresponding items.

All literals and the contents of all identifiers appearing before the keyword FROM are operands. All operands must be numeric category items.

The composite size of identifiers and/or literals is limited to 18 digits. The composite size is determined by superimposing data items on their respective decimal points in a hypothetical data item. The items used in composite size determination depend on the format of SUBTRACT:

Format 1 uses all identifiers and/or literals specified.

Format 2 uses all operands, but not the identifiers into which the result is stored.

Format 3 uses separately each pair of corresponding data items.

The ROUNDED phrase, the SIZE ERROR phrase, and the CORRESPONDING phrase are the same as discussed with the ADD statement. Additional information about arithmetic operations is in section 1.

```
Format 1

    SUBTRACT { literal-1    } [ , literal-2    ] . . . FROM identifier-m  [ROUNDED]
             { identifier-1 } [ , identifier-2 ]

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


Format 2

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

         GIVING identifier-n  [ROUNDED] [ , identifier-o  [ROUNDED] ] . . .

         [; ON SIZE ERROR imperative-statement]


Format 3

    SUBTRACT { CORRESPONDING }  identifier-1 FROM identifier-2 [ROUNDED]
             { CORR          }

              [, identifier-3 [ROUNDED]] . . .  [; ON SIZE ERROR imperative-statement]
```

Figure 5-28. SUBTRACT Statement Format

Format 1 SUBTRACT

Format 1 SUBTRACT occurs as follows: The values of the operands are first added; their sum is then subtracted from the current value of identifier-m. The result is stored in identifier-m. If identifier-n is specified, the sum of the operands is subtracted from, and the result stored in, each identifier-n specified. All format 1 identifiers must be elementary numeric items.

Format 2 SUBTRACT

Format 2 SUBTRACT occurs as follows: The values of the operands are first added; their sum is then subtracted from the current value of identifier-m or literal-m. The result is stored in each identifier-n. If identifier-o is specified, the result is stored in each identifier-o specified. Result identifiers specified after the keyword GIVING can be elementary numeric or elementary numeric edited items.

Format 3 SUBTRACT

Format 3 SUBTRACT subtracts the contents of items in identifier-1 from, and stores the results in, corresponding items in identifier-2. If identifier-3 is specified, corresponding data items in identifier-1 and in each identifier-3 specified are treated accordingly. All identifiers in a format-3 SUBTRACT must be group items. Identifiers must not be reference modified.

SUPPRESS STATEMENT

The SUPPRESS statement inhibits presentation of a report group. See section 6, Report Writer Facility.

TERMINATE STATEMENT

The TERMINATE statement completes processing of specified reports. See section 6, Report Writer Facility.

UNSTRING STATEMENT

The UNSTRING statement (figure 5-29) separates contiguous data in a sending field into one or more receiving items. Transfer of characters to the receiving fields occurs as described for the MOVE statement. Depending on the phrases used in the statement, the entire sending item or parts of the sending item are transferred. During transfer, the system can tally the number of characters examined in the sending item. When a sending item and a receiving item share a part of their storage areas, the results are unpredictable.

NOTE

Refer to appendix F for recommendations on the use of the UNSTRING statement.

The sending item, identifier-1, must be an elementary category alphanumeric item or a group item and must not be reference modified. The receiving items, identifier-4 and identifier-7, must be USAGE IS DISPLAY elementary or group items and can be described in a PICTURE clause as: alphabetic without the symbol B, alphanumeric category, or numeric without the symbol P. If the symbol V is used in a numeric receiving item, it is ignored. If the symbol S is used, the transfer of characters occurs as for an alphanumeric move.

Any subscripting or indexing associated with any identifier is evaluated only once, immediately before any data is transferred from or to the item. A level 88 item must not be specified in an UNSTRING statement.

```
UNSTRING identifier-1

[ DELIMITED BY [ALL] {identifier-2}  [, OR [ALL]  {identifier-3}] ...]
                     {literal-1   }               {literal-2   }

INTO identifier-4 [, DELIMITER IN identifier-5]  [, COUNT IN identifier-6]

     [, identifier-7 [, DELIMITER IN identifier-8]  [, COUNT IN identifier-9]] ...

   [WITH POINTER identifier-10] [TALLYING IN identifier-11]

[; ON OVERFLOW imperative-statement]
```

Figure 5-29. UNSTRING Statement Format

POINTER Phrase

The POINTER phrase specifies the character position within the sending item at which transfer is to begin. The first character position is position 1. If the phrase is omitted, transfer begins with the leftmost character. Each time a character is transferred, the POINTER identifier-10 item is incremented by one. The item, which must be an elementary numeric integer item, must have its initial value established by the program.

TALLYING Phrase

The TALLYING phrase specifies an item to be incremented by one for each receiving item acted upon. Identifier-11 must be an elementary numeric integer item with an initial value established by the program.

ON OVERFLOW Phrase

The ON OVERFLOW phrase specifies an imperative-statement that executes when the receiving items are filled, taking account of any DELIMITED BY phrase before the end of the sending item is encountered or when the POINTER value is less than one or is greater than the size of the sending item. Control passes to the next executable sentence in the absence of the ON OVERFLOW phrase or at the successful completion of UNSTRING.

DELIMITED BY Phrase

The DELIMITED BY phrase separates the sending item into fields that can be used for unstringing. Examining of the sending item begins at the leftmost character or at the character position indicated by the POINTER phrase. The examining continues until the characters specified by DELIMITED BY are encountered, the sending item ends, or the receiving item is full. Examining for the second or subsequent receiving item begins at either the character after any delimiter that stopped the previous examining or the character after the last character previously examined. Examining continues until the delimiter is encountered, the sending item ends, or the receiving item is full. Examining of the characters of the sending item follows the rules of a MOVE statement. A numeric receiving item might not contain the expected characters if the last group of characters of the sending item does not end with a character indicated in the DELIMITED BY phrase.

The delimiter is specified by literal-1 or the contents of identifier-2. It can be a nonnumeric literal, figurative constant without the keyword ALL, or an elementary or group category alphanumeric item containing any characters from the computer character set. Eacn identifier or literal in a DELIMITED BY phrase acts as a single delimiter. A figurative constant stands for a single character.

In the absence of the keyword ALL, two or more contiguous delimiters in the sending item cause the current receiving item to be space or zero filled, according to the description of the receiving item. If ALL appears, however, two or more contiguous delimiters are treated as a single delimiter and the following characters in the sending item are transferred to the receiving items.

DELIMITER IN Phrase

The DELIMITER IN phrase is valid only when DELIMITED BY is used. It specifies the alphanumeric elementary or group item to which the delimiting characters for this receiving item are transferred during UNSTRING. If the delimiting condition is the end of the sending item rather than a specified delimiter, identifier-5 is set to spaces.

COUNT IN Phrase

The COUNT IN phrase is valid only when DELIMITED BY is used. It specifies the elementary numeric integer item in which the system sets the number of characters examined in the sending item, excluding delimiting characters, for the receiving item associated with identifier-4.

USE STATEMENT

The USE statement defines the conditions under which the following procedures are executed. It can appear only in the Declaratives portion of the Procedure Division and must immediately follow a section header. The USE statement itself is not executed; rather, the associated procedures execute when the system determines that one of the specified conditions has occurred.

USE statements are concerned with several different types of conditions:

Format 1 specifies procedures for input-output error or exception handling.

Format 2 specifies procedures for execution immediately before a report group is written to the report file. See section 6, Report Writer Facility.

Format 3 identifies user items that are to be monitored by the associated debugging section. See section 10, Debugging Aids.

Format 4 specifies the algorithm used to compute the home block number for each record in a file with direct organization.

Format 5 specifies a procedure for generating an access control key for data base files. See section 14, Sub-Schema Facility.

Format 6 specifies procedures to be executed when a deadlock situation occurs during data base file processing. See section 14, Sub-Schema Facility.

Format 1 and format 4 of the USE statement are shown in figure 5-30. The combined total of format 1 and format 4 USE statements must not exceed 64.

Files named in formats 1 and 4 must not be sort or merge files. Appearance of a file-name must not cause the simultaneous request for more than one USE procedure.

Procedures must not reference any nondeclarative procedures. Execution of any statement within the USE procedure must not cause execution of a USE procedure that has previously been invoked and has not yet returned control to the invoking routine.

Format 1 USE

Format 1 USE procedures execute for all file organizations under any of these conditions:

After standard input-output error routines are executed.

When an invalid key condition occurs, but an INVALID KEY phrase is not in the input-output statement that caused the condition.

When an at end condition occurs, but the AT END phrase is omitted from the input-output statement.

The words ERROR and EXCEPTION are synonymous and can be used interchangeably. If a file-name appears in the statement, the procedures are valid only for those files named. Otherwise, procedures execute for all files in the open mode specified in the USE statement.

Control returns to the system error routine that called the procedure. If the error is a fatal CYBER Record Manager error or a COBOL detected error, the program might terminate after the declarative procedure executes. To override this normal abort, execute ENTER "C.IOENA" before exiting from the declarative. This command causes the program to continue; however, continued execution of some errors can cause serious consequences such as infinite looping. If "C.IOENA" is called, the error limit (ERL) should be set via the USE clause in the SELECT statement or with a FILE control statement. To determine what error occurred and the appropriate action to take, C.IOST should be called.

Format 4 USE

Format 4 USE procedures execute each time a record in a direct file is accessed. The same USE procedures must be specified in every program that accesses the file.

The procedure must use the contents of the primary key to determine a value corresponding to a home block number. The result, which the program must place in the special register HASHED-VALUE, must be a positive integer value less than the value specified in the BLOCK COUNT clause in the File-Control entry for the file. HASHED-VALUE is defined as an 8-digit COMP-1 integer item.

WRITE STATEMENT

The WRITE statement (figure 5-31) releases a record to an output file. The file must be opened for OUTPUT, I-O, or EXTEND. Both the access mode and the file organization affect the format and use of WRITE.

Format 1 releases a record to a file with sequential organization.

Format 2 releases a record to a file with organization other than sequential.

Execution of WRITE updates any FILE STATUS item defined for the file.

In the statement, record-name specifies the record to be written. Record-name must be a record named in a level 01 entry in the File Section of the Data Division. The record can be contained in a file described in a sub-schema. Record-name must not be the name of a record in an SD entry.

Format 1

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

Format 4

```
USE FOR HASHING ON file-name-1 [, file-name-2]
```

Figure 5-30. USE Statement Format 1 and Format 4

```
Format 1

   WRITE record-name [FROM identifier-1]

        [                     ( {identifier-2}  [LINE ] ) ]
        [ {BEFORE}            ( {integer     }  [LINES] ) ]
        [ {AFTER }  ADVANCING {                         } ]
        [                     ( {mnemonic-name}         ) ]
        [                     ( {PAGE         }         ) ]

        [ ; AT {END-OF-PAGE}   imperative-statement ]
        [      {EOP        }                        ]


Format 2

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

Figure 5-31. WRITE Statement Format

Once WRITE executes, the record is no longer available in the record area, unless the WRITE statement execution is unsuccessful due to an invalid key condition or a boundary violation. If the associated file is named in a SAME RECORD AREA clause, the record is available to the program as a record of other files specified in the same clause.

The number of characters written to the file is the number of characters indicated by the Record Description entry. The program is responsible for the content of the record area; that is, any desired space fill must be provided within the program.

Format 1 WRITE

Format 1 is valid only for sequential files. Only the FROM optional phrase has meaning for files that are not to be printed. The other optional phrases can be used on files that are not to be printed, but they might insert a carriage control character as the first character of every record and might generate extra dummy records. If the phrases are used and the record type is not Z, the run is aborted.

FROM Phrase

Execution of a WRITE with the FROM phrase is equivalent to execution of:

    MOVE identifier-1 TO record-name.

    WRITE record-name.

Record-name and identifier-1 must not reference the same storage area. The contents of the record area prior to the implicit MOVE have no effect on WRITE execution. Information in identifier-1 remains available to the program.

ADVANCING Phrase

The ADVANCING phrase specifies positioning before or after the record is written, depending on whether the keyword BEFORE or the keyword AFTER appears in the phrase. The remainder of the phrase determines the type of positioning:

Identifier-2 or integer

Skip number of lines specified by integer or the value of identifier-2. Identifier-2 must be an unsigned integer data item. A value of zero can be specified.

PAGE

Skip to the first print line on the next logical page defined by the LINAGE clause in the Data Division FD entry. Alternatively, a top-of-form carriage control character can be used for this purpose. See the LINAGE clause in section 4. If no LINAGE clause exists for the file containing record-name, a top-of-form carriage control character becomes the first character of the record. If the device on which the file is being output has no top-of-form concept (for example, a file being printed at a Teletypewriter), the phrase acts as if ADVANCING 1 LINE had been specified.

Mnemonic-name

Carriage control character defined by mnemonic-name becomes the first character in the current or next record. The mnemonic-name must be specified in an implementor-name IS mnemonic-name clause in the SPECIAL-NAMES paragraph of the Environment Division. Mnemonic-name cannot be used to position the record when the FD entry for the file contains a LINAGE clause.

If a WRITE statement with the ADVANCING phrase has been executed on a file since the file was last opened, a subsequent WRITE statement that does not specify the ADVANCING phrase causes a record to be written as if AFTER ADVANCING 1 LINE had been specified if the first character of the line is a space. If the first character is not a space, the record is written as if AFTER ADVANCING mnemonic-name had been specified.

Mixing BEFORE and AFTER can cause overprinting. For example, AFTER 2 followed by BEFORE integer causes overprinting. Advancing integer lines moves the carriage forward integer lines; it does not leave integer blank lines between printed lines. For example, AFTER integer followed by AFTER 3 leaves two blank lines between the two printed lines.

Blank lines are generated for positioning in the following instances:

A WRITE statement with the ADVANCING phrase

A WRITE statement without ADVANCING on a file with LINAGE specified

A WRITE statement without ADVANCING on a file with a WRITE ADVANCING statement as the previous operation

The COBOL compiler might insert standard carriage control characters (blank + - 0) in these lines as well as in the line to be printed. The hyphen character is not legal on some devices under the NOS operating system. It can be eliminated by specifying NOTRIP on the execution control statement. If the character is legal, NOTRIP might cause a larger print file and slower printing.

END-OF-PAGE Phrase

The END-OF-PAGE phrase and its equivalent EOP are valid only for print files in which the FD entry includes a LINAGE clause. LINAGE defines a page body that includes an optional footing area.

If a WRITE statement execution causes printing or spacing in the footing area of the page body, an end-of-page condition exists. The WRITE statement is executed, and then the imperative-statement in the EOP phrase is executed.

If the WRITE statement execution would exceed the page body including the footing area, a page overflow condition exists. For a page overflow condition, the WRITE statement executes, the file is positioned to the beginning of the next logical page, the special register LINAGE-COUNTER is reset to 1, and the imperative-statement of the EOP phrase executes.

If no footing area is defined, an overflow condition exists.

A combined end-of-page and overflow condition acts as an overflow condition.

When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists, and the contents of the record area are unaffected. Exception conditions are discussed with the FILE STATUS data item in section 3.

Format 2 WRITE

Format 2 must be used for any file that does not have sequential organization. Its use and results are affected by file organization, access mode, and open mode.

Sequential Access Mode

Files open with OUTPUT or I-O in the sequential access mode have records written to the file under system control.

Relative and word-address files need not have a key defined within the program. The system generates a key for each record referenced in a format 2 WRITE statement, writes the record to the file, and returns the key value to the program when the RELATIVE KEY or WORD-ADDRESS KEY data item is specified.

Extended actual-key files must have a RECORD KEY data item defined within each record or within the Working-Storage Section, although the program does not establish a value in the item. Initial actual-key files must have an embedded key. The system generates a key for each record referenced in a format 2 WRITE statement, writes the record to the file, and returns the key value to the program in the RECORD KEY data item.

Extended direct files must have a RECORD KEY data item defined within each record or within the Working-Storage Section. Initial direct files must have an embedded key. The program is responsible for setting a unique key value in the item for each file record. The key value determines the location of the record in the file.

Extended indexed files must have a RECORD KEY data item defined within each record or within the Working-Storage Section. Initial indexed files must have an embedded key. The program is responsible for setting a unique key value in the item for each file record and for writing the records in order by ascending key value. The key value determines the location of the record in the file. An invalid key condition exists for any attempt to write a record out of order for initial indexed files. Sequential access mode is not required for extended indexed files.

When an indexed, direct, or actual-key file has alternate keys, all alternate record key values must be unique within all records of the file if the DUPLICATES phrase is omitted from the ALTERNATE RECORD KEY clause for the file. Any duplication of an alternate record key value under these circumstances creates an invalid key condition.

For most efficient processing, an indexed file should be created only in the sequential access mode.

Random or Dynamic Access Mode

Files open with OUTPUT or I-O in the random access mode or dynamic access mode have records written according to the key values supplied by the program.

In files with organizations other than word-address, WRITE cannot be used to rewrite an existing record. An invalid key condition occurs on an attempted write of a record that duplicates an existing record key value or that violates one of the ALTERNATE RECORD KEY phrases (such as no duplicate values).

Word-address files have the record written to the address specified by the value of the WORD-ADDRESS KEY data item, with no regard for other information that might be at that location.

Extended indexed and extended direct files must have a RECORD KEY data item defined within each record or within the Working-Storage Section. Initial indexed and initial direct files must have the key embedded in the record. The value of the item must be unique among all file records.

Extended actual-key files must have a RECORD KEY data item defined within each record or within the Working-Storage Section. Initial actual-key files must have an embedded key. The value of the item must be unique among all records of the file for initial actual-key organization. The key must be formatted to correspond to a block number and a record number within the block. The block number must not be more than 1 greater than the highest number of an existing block. System key generation is recommended for all records written to an actual-key file. A zero value in the key data item requests system generation. The system generates a key value, writes the record to that location, and returns the key value to the program in the RECORD KEY data item.

Relative file records are written to the location specified by the value of the RELATIVE KEY data item.

The values for any alternate record keys defined for indexed, direct, and actual-key files must meet the same criteria as for files open in sequential access mode.

FROM Phrase

The FROM phrase is the same as for format 1.

INVALID KEY Phrase

The INVALID KEY phrase specifies the imperative-statement that executes when an invalid key condition occurs. Situations that cause the condition include the following:

Duplicated primary key values for records existing in indexed, direct, or actual-key files or relative key values in relative files.

Violations of the ALTERNATE RECORD KEY phrases for indexed, direct, or actual-key files.

Improperly formatted actual-key file primary key, including a block number that exceeds the highest existing block number plus one. (That is, records not written in ascending block order.)

When the phrase is omitted, strict ANSI usage requires an appropriate format 1 USE statement for the file. See the FILE STATUS discussion in section 3.

REPORT WRITER FACILITY

The Report Writer facility allows a report to be produced from program specification of the report's physical appearance, rather than through detailed specification of the procedure needed to write the report.

A report is named in a REPORT clause of an FD entry in the File Section of the Data Division. The description of the report file appears in the Report Section of the Data Division.

Each report is defined in the Report Section using a Report Description (RD) entry followed by Report Group Description entries.

The RD entry specifies the overall page format and defines break control items that allow, for example, subtotals to be generated at the bottom of each page.

Each Report Group Description entry specifies one of the seven different types of report groups. Table 6-1 summarizes report group types and the circumstances under which they are generated. A report group generally is one line of a page, but it can extend over several lines. The TYPE statement of the entry defines the report group type.

TABLE 6-1. REPORT GROUP TYPE SUMMARY

Report Group Type	When Written to Report File
Report heading	Once per report during execution of the first GENERATE statement.
Page heading	Once at the beginning of each page.
Body group:	
Control heading	Either once per report or each time a control break occurs, depending on the CONTROL clause.
Detail	Each time a GENERATE data-name statement executes.
Control footing	Either once per report or each time a control break occurs, depending on the CONTROL clause.
Page footing	Once at the end of each page.
Report footing	Once per report during execution of the TERMINATE statement.

Each elementary item in a report group is described by the position it is to occupy on the page and by the origin of the data. Relative or absolute vertical position is specified by the LINE NUMBER clause; horizontal position is specified by a COLUMN NUMBER clause. Each elementary item description must include one of the following clauses to define a printable item and to identify the origin of data for the item:

SOURCE specifies a sending item.

SUM specifies items that are to be added to obtain the item; it is valid only for a CONTROL FOOTING report group.

VALUE specifies a literal.

Many clauses that define elementary items in a report group are the same as those used for Record Description entries in the File Section of the Data Division. Only those unique to the Report Section are explained here; see section 4 for details of other clauses.

Report groups are written to the report file through execution of Procedure Division statements. The report file must be opened by the program before any Report Writer statement is executed, and should be closed after the last Report Writer statement. No other input-output statements should reference the report file.

Report Writer statements are:

INITIATE	Initializes counters and special registers.
GENERATE	Writes a DETAIL report group or produces a summary report, depending on statement elements.
TERMINATE	Performs end-of-report processing.
SUPPRESS	Suppresses report group in a USE BEFORE REPORTING declarative procedure.

Each time the program executes a GENERATE data-name statement, a type DETAIL report group is written to the report file. Other report groups might also be written, depending on the current circumstances analyzed by Report Writer. The user program does not directly initiate processing of any report group type except DETAIL. A summary report is produced by a GENERATE report-name statement.

REPORT FILE

The report file is named by an FD entry in the File Section. Information for all reports named in the FD entry is accumulated on the report file. When more than one report is written to the file, the CODE clause should be specified in the Report Section so that entries for one report can be distinguished from those of other reports.

The job in which the program appears is responsible for the ultimate disposition of the report file. If the SELECT clause assigns the file to the implementor-name "OUTPUT", the report is printed at job termination. Otherwise, the job should preserve the report file for later processing or output.

Each report group written to the file contains carriage control characters and data properly formatted by Report Writer to produce the report described in the program. When only one report is specified, the report can be printed by a control statement that copies the report file to a file with print disposition. When more than one report is specified, however, the report file should be processed by a user program that uses CODE clause identifiers to separate lines of one report from lines of another.

REPORT FILE STRUCTURE

The report file described in the FD entry must have sequential organization. Records in the file are 137 or 139 characters in length, depending on whether the CODE clause is used. When the CODE clause is specified, the line is 139 characters:

> The first two characters in the line are the code identifier.
>
> The third character is the carriage control character.
>
> Characters 4 through 139 are data defined by the program as column numbers 1 through 136.

When the CODE clause is omitted, line length is 137 characters, with the system-supplied carriage control character in the first position.

The file has block type C and record type Z structure. Report records are generated by the WRITE statement with ADVANCING phrase. Refer to the WRITE statement for further information about these records.

REPORT CLAUSE

The REPORT clause (figure 6-1) specifies the names of reports to be written to the file. The clause must appear in an FD entry defining the report file.

$$\left\{\begin{array}{l}\underline{\text{REPORT}}\ \text{IS}\\ \underline{\text{REPORTS}}\ \text{ARE}\end{array}\right\}\ \text{report-name-1}\ [,\ \text{report-name-2}]\ \ldots$$

Figure 6-1. REPORT Clause Format

A Record Description entry cannot follow an FD entry that contains a REPORT clause.

In the clause, each report-name specifies a report that must be the subject of an RD entry in the Report Section. The presence of more than one report-name in the REPORT clause indicates that the file contains more than one report. The order of the appearance of the report-names is insignificant. A report-name must appear in only one REPORT clause.

SPECIAL REGISTERS

Two special registers, referenced by the reserved words PAGE-COUNTER and LINE-COUNTER, are automatically created for each report specified in the Report Section. They implicitly describe 6-digit COMP-1 unsigned integers, whose values are maintained by Report Writer.

PAGE-COUNTER and LINE-COUNTER are referenced in the Report Section by the SOURCE clause. Within the Report Section, references need not be qualified. Outside of the Report Section, these registers can be used in any context in which a data-name with an integer value can appear; references to PAGE-COUNTER and LINE-COUNTER must be qualified by the report-name when more than one report is specified for the program.

PAGE-COUNTER

The value of PAGE-COUNTER is used by Report Writer to number the pages of a report. The register is set to 1 when the report is initialized by the INITIATE statement. It is incremented by 1 each time a page advance occurs.

The value of PAGE-COUNTER can be altered by Procedure Division statements.

LINE-COUNTER

The value of LINE-COUNTER represents the line number on which the print line is to be written. The register is set to zero when the report is initialized by the INITIATE statement. It is incremented during processing of each report group and reset to zero at the beginning of each page. The value of LINE-COUNTER cannot be altered by Procedure Division statements.

REPORT SECTION

The Report Section of the Data Division describes reports that are generated through the Report Writer. Each report must be named in the REPORT clause of an FD entry in the File Section and must be specified in detail, beginning with an RD entry, in the Report Section.

The Report Section is composed of a section header followed by Report Description entries. The section header is:

> <u>REPORT SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

Each RD entry is followed by any number of Report Group Description entries for report groups and elementary items that further describe report characteristics. At least one Report Group Description entry is required; at least one body group is required.

REPORT DESCRIPTION ENTRY

The Report Description entry names a report, specifies any identifying characters that are to be prefixed to each print line to distinguish lines for this report from those of other reports, and describes the physical structure and organization of the report. The entry consists of the level indicator RD followed by the report name and descriptive clauses defining the report and its structure. Clauses specify the vertical boundaries of the region within which each type of report group is printed, the break control items, and the definition of each page of the report. The general format of the Report Description entry is shown in figure 6-2.

```
RD   report-name
 [; CODE clause]   [; CONTROL clause]
 [; PAGE clause]   .
```

Figure 6-2. Report Description Entry Skeleton

Report-name, which specifies the name of the report, must also appear in one and only one REPORT clause of an FD entry. Report-name is the highest level qualifier that can be specified for LINE-COUNTER, PAGE-COUNTER, and all data-names defined within the Report Section. Report-name must follow the RD indicator.

CODE Clause

The CODE clause (figure 6-3) specifies a two-character identifier that Report Writer adds to each line to distinguish entries for this particular report. It should be used when the FD entry for the report file specifies more than one report.

```
CODE literal
```

Figure 6-3. CODE Clause Format

When the CODE clause is specified for one report named in an FD entry, it must be specified for all reports named in that entry.

In the clause, literal must be a two-character nonnumeric literal. The characters become the first two character positions of each line generated through Report Writer. The positions occupied by the literal are not included in the description of the print line, but they are included in the record size.

Separation of report entries on the basis of CODE clause characters is the responsibility of a user program.

CONTROL Clause

The CONTROL clause (figure 6-4) establishes a hierarchy of break control items to cause processing of control footings and control headings. The CONTROL clause must be specified when:

TYPE CONTROL HEADING or TYPE CONTROL FOOTING clauses are specified.

RESET clause is part of a SUM clause.

If the clause is omitted, control breaks are not processed.

Break control items referenced in this clause are the same items declared in the TYPE clause of a Report Group Description entry for a CONTROL FOOTING or CONTROL HEADING report group. Control footings and control headings are processed when a control break occurs. A control break is defined as a change in the value of a break control item detected by Report Writer during execution of a GENERATE statement. Each time a GENERATE statement executes, Report Writer compares the current values of all break control items with the values existing at the time the previous GENERATE statement executed. A change in the value of a break control item defines a control break for that item.

The number of CONTROL FOOTING or CONTROL HEADING report groups processed for a given control break depends on the position of the break control item in the CONTROL clause list. Processing occurs for the footing or heading associated with the rightmost break control item in the list, proceeds to the second-from-the-right item, and continues processing for the footing or heading item to the left. The last footing or heading processed for a given control break is the footing or heading associated with the break control item that caused the control break. See the GENERATE statement discussion for additional details of this process.

The break control item that causes a control break is reset after processing, unless the RESET phrase of the SUM clause that defined the item specifies other conditions for reset.

The values of break control items that are used to detect a control break are called prior values. The prior values are made available to break control items referenced in either a USE procedure or SOURCE clause associated with CONTROL FOOTING report group during control break processing. Any other references to the break control items access the current value. Execution of a TERMINATE statement causes the CONTROL FOOTING, REPORT FOOTING, and USE procedures to reference the prior values of the break control item as if a control break occurred.

In the clause, each data-name or FINAL must correspond to a data-name or FINAL in a Report Group Description entry with a TYPE CONTROL HEADING clause or a TYPE CONTROL FOOTING clause. Each data-name must define a different fixed-length item in the File Section, Working-Storage Section, or Common-Storage Section of the Data Division. Data-names can be qualified, but must not be subscripted or indexed. The clause defines break control items.

The leftmost data-name is the major break control item; the rightmost data-name is the minor break control item.

PAGE Clause

The PAGE clause (figure 6-5) defines the length of a logical page and the vertical subdivisions within which report groups are presented. When the PAGE clause is

```
{CONTROL IS   }  {data-name-1  [, data-name-2] . . .                }
{CONTROLS ARE }  {FINAL  [, data-name-1  [, data-name-2] . . .]     }
```

Figure 6-4. CONTROL Clause Format

```
      ┌LIMIT IS  ┐           ┌LINE ┐
PAGE  └LIMITS ARE┘ integer-1 └LINES┘ [, HEADING integer-2]

      [, FIRST DETAIL integer-3]  [, LAST DETAIL integer-4]

      [, FOOTING integer-5]
```

Figure 6-5. PAGE Clause Format

omitted or integer-1 is incorrectly specified, the report consists of a single page of indefinite length and only relative NEXT GROUP can be specified in any Report Group Description entry for the report. Phrases of the PAGE clause can appear in any order.

If line number specifications for REPORT HEADING and PAGE HEADING report groups overlap on the same page, the SUPPRESS statement should be used to suppress printing of one of the report groups at execution time. The same is true for overlapping PAGE FOOTING and REPORT FOOTING report groups. If neither is suppressed, an error exists and neither of the overlapping groups appears.

In the clause, integer-1 specifies the number of lines available on each page. Integer-1 must be an unsigned integer with a value greater than or equal to any value of integer-5. The maximum value of integer-1 is 999.

HEADING Phrase

The HEADING phrase designates the first line number on which a REPORT HEADING or PAGE HEADING report group can be presented. Integer-2 must be an unsigned integer with a value greater than or equal to 1. When the phrase is omitted or integer-2 is incorrectly specified, the value of integer-2 defaults to 1.

FIRST DETAIL Phrase

The FIRST DETAIL phrase specifies the first line number on which a body group can be presented. Integer-3 must specify an unsigned integer with a value greater than or equal to the value of integer-2. When the phrase is omitted or integer-3 is incorrectly specified, the value of integer-3 defaults to the value of integer-2. A REPORT HEADING or PAGE HEADING report group cannot be presented on or beyond the line number specified by integer-3.

LAST DETAIL Phrase

The LAST DETAIL phrase specifies the last line number on which a CONTROL HEADING or DETAIL report group can be presented. Integer-4 must specify an unsigned integer with a value greater than or equal to integer-3. When the phrase is omitted or integer-4 is incorrectly specified, the default value depends on the FOOTING phrase: if FOOTING is specified, the default value is integer-5; if FOOTING is omitted, the default is integer-1.

FOOTING Phrase

The FOOTING phrase specifies the last line number on which a CONTROL FOOTING report group can be presented. Integer-5 must be an unsigned integer with a value greater than or equal to the value of integer-4. When the phrase is omitted or integer-5 is incorrectly specified, the default value depends on the LAST DETAIL phrase: if LAST DETAIL is specified, the default value is integer-4; if LAST DETAIL is omitted, the default is integer-1. PAGE FOOTING and REPORT FOOTING report groups must follow the line number specified by integer-5.

Table 6-2 summarizes the regions established by the PAGE clause.

REPORT GROUP DESCRIPTION ENTRY

The Report Group Description entry (figure 6-6) specifies the characteristics of a report group and the individual items within a report group. The entry follows the RD entry in the Report Section.

A Report Group Description entry must begin with a level-number 01 entry that contains a TYPE clause. The level 01 entry can be followed by any number of group and elementary entries.

The data-name clause, when present, must follow the level number. The other clauses can appear in any sequence. An entry that contains a LINE NUMBER clause cannot have a subordinate entry that also contains a LINE NUMBER clause.

Format 1 Entry

The first entry in a Report Group Description entry must be format 1 with its required TYPE clause. The data-name is required only if one of the following occurs:

A GENERATE statement references a DETAIL report group;

The UPON phrase of a SUM clause references a DETAIL report group;

A CONTROL FOOTING report group qualifies a sum counter reference;

A USE BEFORE REPORTING procedure statement references a report group.

When the USAGE clause is specified, at least one subordinate entry must contain a SOURCE, SUM, or VALUE clause to define a printable item.

TABLE 6-2. PAGE REGIONS SUMMARY

Report Groups that can be Presented in the Region	First Line Number of the Region	Last Line Number of the Region
REPORT HEADING with NEXT GROUP NEXT PAGE phrase	integer-2	integer-1
REPORT FOOTING with LINE integer-1 NEXT PAGE phrase		
REPORT HEADING without NEXT GROUP NEXT PAGE phrase	integer-2	integer-3 minus 1
PAGE HEADING		
CONTROL HEADING	integer-3	integer-4
DETAIL		
CONTROL FOOTING	integer-3	integer-5
PAGE FOOTING	integer-5 plus 1	integer-1
REPORT FOOTING without LINE integer-1 NEXT PAGE phrase		

```
Format 1
  01  [data-name]
        [; LINE NUMBER clause]   [; NEXT GROUP clause]   ; TYPE clause   [; USAGE clause].

Format 2
  level-number [data-name]
        [; LINE NUMBER clause]   [; USAGE clause].

Format 3
  level-number [data-name]
        [; BLANK WHEN ZERO clause]   [; COLUMN NUMBER clause]   [; GROUP INDICATE clause]
        [; JUSTIFIED RIGHT clause]   [; LINE NUMBER clause]   ; PICTURE clause
        { ; SOURCE clause }
        { ; VALUE clause  }      [; USAGE clause].
        { ; SUM clause    }
```

Figure 6-6. Report Group Description Entry Skeletons

Format 2 Entry

A format 2 entry is a group entry, that, if present, must be subordinate to a format 1 entry and be followed by at least one format 3 entry. A format 2 entry can have a level-number of 02 through 48 and must contain at least one optional clause. Any data-name-1 specified can be used only to qualify a sum counter reference.

When the USAGE clause is specified, at least one subordinate entry must contain a SOURCE, SUM, or VALUE clause to define a printable item.

Format 3 Entry

A format 3 entry must define an elementary data item. The entry can follow either a format 1 entry or a format 2 entry; the level-number can be any integer from 02 through 49. In a format 3 entry:

An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.

An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.

A LINE NUMBER clause cannot be the only clause specified.

A GROUP INDICATE clause can only appear in a TYPE DETAIL report group.

A SUM clause can only appear in a TYPE CONTROL FOOTING report group.

Any data-name clause can be used only to qualify a sum counter reference, although it can always be specified.

The combined clauses that can be specified for a format 3 entry are shown in table 6-3.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause causes substitution of spaces for a zero value of an item. See section 4, Data Description Entry.

COLUMN NUMBER Clause

The COLUMN NUMBER clause (figure 6-7) specifies the column number position of a printable item on a print line. A printable item is established by the presence of a VALUE, SOURCE, or SUM clause; a print line by the presence of a LINE NUMBER clause.

```
COLUMN NUMBER IS integer
```

Figure 6-7. COLUMN NUMBER Clause Format

The COLUMN NUMBER clause must be specified at the elementary level within a report group. The clause must appear in an entry that either contains a LINE NUMBER clause or is subordinate to an entry that contains a LINE NUMBER clause.

In the clause, integer specifies the column number of the leftmost character position of the item. Integer must be an unsigned integer with a value in the range 1 through the number of printable columns in the print line. (The printable columns of a print line are numbered 1 through 136, left to right, with the first position of the print line being column 1.)

When more than one printable item exists within a print line, entries for each line must be ordered such that COLUMN NUMBER clauses define columns in ascending order.

GROUP INDICATE Clause

The GROUP INDICATE clause (figure 6-8) specifies that the associated printable item is presented only under special conditions. It can appear only at the elementary level of a DETAIL report group item.

```
GROUP INDICATE
```

Figure 6-8. GROUP INDICATE Clause Format

TABLE 6-3. FORMAT 3 CLAUSE COMBINATIONS

Format 3 Item to be Established	Clauses									
	SUM	VALUE	SOURCE	COLUMN	PIC	JUST	BLANK WHEN ZERO	GROUP INDICATE	USAGE	LINE
Nonprintable sum counter	M	-	-	-	M	-	-	-	-	P
Printable sum counter	M	-	-	M	M	-	P	-	P	P
Printable constant	-	M	-	M	M	P†	-	P	P	P
Nonprintable item moved from any Data Division Section	-	-	M	-	M	P†	P††	-	-	P
Printable item moved from any Data Division Section	-	-	M	M	M	P†	P††	P	P	P

† Permitted for an item which is not numeric or numeric edited.

†† Permitted for a numeric or numeric edited item

M Mandatory
P Allowed
- Not allowed

When either the PAGE clause or the CONTROL clause is specified for the associated RD entry, the GROUP INDICATE item is printed according to SOURCE or VALUE specifications at the first presentation of:

Associated report group in the report.

Report group after every page advance.

Report group after each control break.

When neither the PAGE clause nor the CONTROL clause is specified for the associated RD entry, the GROUP INDICATE item is presented the first time its report group is processed after the INITIATE statement is executed. The GROUP INDICATE clause is ignored thereafter. Spaces are supplied for the printable item whenever the GROUP INDICATE clause causes an item to be ignored.

JUSTIFIED Clause

The JUSTIFIED clause specifies nonstandard positioning of data in a receiving item. See section 4, Data Description Entry.

LINE NUMBER Clause

The LINE NUMBER clause (figure 6-9) defines a print line and specifies vertical positioning for that line within a report group. The clause must be specified for every entry that contains a COLUMN NUMBER clause, except that a LINE NUMBER clause must not be subordinate to an entry that also contains a LINE NUMBER clause. Positioning indicated by the clause occurs before the print line is written to the report file.

```
                  { integer-1 [ON NEXT PAGE] }
LINE NUMBER IS    { PLUS integer-2           }
```

Figure 6-9. LINE NUMBER Clause Format

In the clause, integer-1 and integer-2 specify the line on which information is to appear. Both integers must be unsigned and have a value 1 through 999. Either absolute or relative line numbers can be specified.

Integer-1 specifies an absolute line number. It can be used only when a PAGE clause has been specified. All LINE NUMBER clauses with absolute line numbers must precede clauses specifying relative line numbers. Successive clauses must specify integers in ascending, but not necessarily consecutive, order.

Integer-2 specifies a relative line number. Report Writer adds integer-2 to the line counter of the previous print line to determine the position of the line containing this clause. When a relative line number is the first LINE NUMBER clause of the report group, the line number on which the print line is presented is calculated as shown in table 6-4.

Values specified for both integer-1 and integer-2 must be within any vertical subdivisions defined with a PAGE clause.

The first LINE NUMBER clause specified for a PAGE FOOTING report group must be an absolute line number clause.

The NEXT PAGE phrase specifies that the report group is to be presented on a new page, beginning on the line indicated. The phrase can appear in either a REPORT FOOTING report group or the description of a body group. The phrase can appear only once in a report group and, if present, must appear in the first LINE NUMBER clause of the report group.

NEXT GROUP Clause

The NEXT GROUP clause (figure 6-10) specifies vertical positioning following the presentation of the last line of a report group. It must not be specified in a REPORT FOOTING or PAGE HEADING report group or in any report group entry that does not contain at least one LINE NUMBER clause.

```
                 { integer-1      }
NEXT GROUP IS    { PLUS integer-2 }
                 { NEXT PAGE      }
```

Figure 6-10. NEXT GROUP Clause Format

The NEXT GROUP clause refers to the next report group to be presented and, therefore, can affect the location at which the next report group is presented. If the clause appears in a REPORT HEADING report group, it can affect the location at which the PAGE HEADING is presented; if the clause appears in a PAGE FOOTING report group, it can affect the location at which the REPORT FOOTING report group is presented. The NEXT GROUP clause is ignored if it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.

Page positioning specified by the NEXT GROUP clause takes place after the presentation of the report group in which the clause appears, even if one or more of the lines in the group is not a print line. The information supplied by the NEXT GROUP clause is combined with information from the TYPE and PAGE clauses and the LINE-COUNTER value to determine a new value for LINE-COUNTER.

In the clause, integer-1 and integer-2 specify a line number on which the next report group is to be presented. Both integers must be unsigned with a value in the range 1 through 999. Integer-1 specifies an absolute number; integer-2 specifies a relative number as discussed with the LINE NUMBER clause.

When the PAGE clause is omitted from the RD entry, only a relative NEXT GROUP clause can be specified in any Report Group Description entry within that report.

The NEXT PAGE phrase can be used for any report group except PAGE FOOTING. It causes the next report group to appear on the following page of the report. When the phrase is specified for a REPORT HEADING report group, the heading appears by itself on the first page of the report.

TABLE 6-4. LINE NUMBER PRESENTATION RULES

Report Group	Line Number On Which First Line is Presented
REPORT HEADING:	
with PAGE clause	First LINE NUMBER + ((integer-2) - 1).
without PAGE clause	LINE-COUNTER + first LINE NUMBER clause.
PAGE HEADING:	
with PAGE clause	If the REPORT HEADING is presented on the same page, LINE-COUNTER + first LINE-NUMBER clause; otherwise, first LINE NUMBER clause + ((integer-2) - 1).
Body groups (CONTROL HEADING, DETAIL, and CONTROL FOOTING report groups):	
with PAGE clause	If LINE-COUNTER ≥ integer-3 and it is the first body group on the page, LINE-COUNTER + 1. If LINE-COUNTER ≥ integer-3 and it is not the first body group on the page, LINE-COUNTER + first LINE NUMBER clause. If LINE-COUNTER < integer-3, integer-3.
without PAGE clause	LINE-COUNTER + first LINE NUMBER clause.
PAGE FOOTING:	
with PAGE clause	Print line is presented on the line number specified by the first line number clause, which must be an absolute LINE NUMBER clause.
REPORT FOOTING:	
with PAGE clause	If presented on the same page as the PAGE FOOTING report group, LINE-COUNTER + first LINE NUMBER clause. Otherwise, first LINE NUMBER + integer-5.
without PAGE clause	LINE COUNTER + first LINE NUMBER.

PICTURE Clause

The PICTURE clause specifies the general characteristics and editing requirements of an elementary item. See section 4, Data Description Entry.

SOURCE Clause

The SOURCE clause (figure 6-11) identifies the sending data item that is to be moved to a printable item defined within a Report Group Description Entry. The clause is one of three (SOURCE, SUM, or VALUE) that must be specified for each elementary item in a report group to define the purpose of the item within the group.

```
SOURCE IS identifier
```

Figure 6-11. SOURCE Clause Format

The SOURCE clause specifies an implicit MOVE statement that causes the named data item to be transferred to the item within the report group prior to the presentation of the report group.

In the clause, identifier specifies the data item that is to be moved to the report group item. It must be defined so that the operation is allowed by MOVE statement rules.

Identifier can be defined in any section of the Data Division. If it is defined in the Report Section, it must either be the special register PAGE-COUNTER or LINE-COUNTER or be a sum counter of the report.

SUM Clause

The SUM clause (figure 6-12) establishes a sum counter and specifies the items to be summed. The SUM clause must only appear in the description of a CONTROL FOOTING report group. More than one SUM clause can appear in an elementary report group entry, but only one sum counter is established for the entry regardless of the number of SUM clauses specified.

```
{ SUM identifier-1 [, identifier-2] . . .

     [ UPON data-name-1

       [, data-name-2] . . . ] } . . .

[ RESET ON { data-name-3 } ]
           { FINAL       }
```

Figure 6-12. SUM Clause Format

The sum counter is set to zero during execution of the INITIATE statement. The types of summing that occur depend on the identifiers in the clause and the time of summing depends on the Procedure Division statements executed:

> Subtotaling is adding an item that is not a sum counter to a sum counter. It occurs whenever a GENERATE or TERMINATE statement is executed.
>
> Crossfooting is adding a sum counter to itself. It occurs whenever a TERMINATE statement is executed or a CONTROL FOOTING report group is presented as the result of a control break.
>
> Rolling forward is adding a sum counter of a lower level CONTROL FOOTING report group to a sum counter of a higher level CONTROL FOOTING report group. It occurs whenever a TERMINATE statement is executed or a CONTROL FOOTING report group is presented as the result of a control break.

The sum counter established is a numeric data item with an operation sign, whose size is equal to the number of receiving character positions specified by the accompanying PICTURE clause.

The highest qualifier allowed for a sum counter is report-name.

If the report group for a printable item contains a SUM clause, the sum counter serves as a source data item. Sum counter contents are moved to the printable item in accordance with the rules for a MOVE statement. If a report group contains both a data-name clause and a SUM clause, the data-name is the name of the sum counter, not the name of the printable item that the entry also defines.

In the clause, the identifiers must specify numeric data items that contain the values that are to be added into the sum counter. The identifiers themselves can specify sum counters that are defined either in the same control footing report group or in a lower level control footing report group. When two or more identifiers specify the same data item, the data item is added into the sum counter as many times as the data item is referenced in the SUM clause.

UPON Phrase

The UPON phrase allows for selective subtotaling of the identifiers. Subtotaling occurs only when the data-name of the UPON phrase is executed in a GENERATE data-name statement. When UPON is specified, the identifiers of the SUM clause must not name sum counters.

Data-name-1 and data-name-2 must be the names of DETAIL report groups described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. The data-names can be qualified by a report-name. When two or more data-names specify the same DETAIL report group, the incrementing that occurs when a GENERATE data-name statement is executed is repeated as many times as the data-name appears in the UPON phrase.

RESET Phrase

The RESET phrase specifies the conditions under which the sum counter is set to zero. Data-name-3, which designates the break control level at which the sum counter is reset, must be one of the data-names specified in the CONTROL clause for the report. Data-name-3 must not be a lower level break control item than the associated break control item for the report group in which the RESET phrase appears.

If FINAL is specified instead of data-name-3, it must also appear in the CONTROL clause for the report.

When the RESET phrase is omitted, the sum counter is set to zero at the time the control footing is processed for the report group in which the sum counter is specified.

TYPE Clause

The TYPE clause (figure 6-13) specifies the particular type of report group described by the entry and indicates the time at which the report group is to be presented. Each report must include at least one body group (CONTROL HEADING, DETAIL, or CONTROL FOOTING report group).

```
          ( {REPORT HEADING}                                )
          ( {RH            }                                )
          ( {PAGE HEADING}                                  )
          ( {PH          }                                  )
          ( {CONTROL HEADING}     {data-name-1}             )
          ( {CH             }     {FINAL      }             )
TYPE IS  <  {DETAIL}                                         >
          ( {DE    }                                        )
          ( {CONTROL FOOTING}     {data-name-2}             )
          ( {CF             }     {FINAL      }             )
          ( {PAGE FOOTING}                                  )
          ( {PF          }                                  )
          ( {REPORT FOOTING}                                )
          ( {RF            }                                )
```

Figure 6-13. TYPE Clause Format

REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING must not appear more than once in the description of a report.

The PAGE HEADING and PAGE FOOTING report groups can only be specified in a report for which the PAGE clause is included in the corresponding Report Description entry.

After each report group is processed and before the report group is written to the report file, the following occurs:

> The USE BEFORE REPORTING declarative procedure, if any, for the report group is executed.
>
> If a SUPPRESS statement was executed or the report group is nonprintable (the report group has no LINE NUMBER clause), there is no further processing for the report group.
>
> The print line is formatted and written to the report file.

In the clause the two-character keywords are abbreviations for the immediately preceding type.

REPORT HEADING

The REPORT HEADING phrase specifies a report group that is presented once per report, as the first report group. The report group is processed during execution of the chronologically first GENERATE statement for the report.

PAGE HEADING

The PAGE HEADING phrase specifies a report group that is presented as the first report group on each page of the report. The phrase is valid only when a PAGE clause appears in the RD entry. The page heading does not appear on a page designated to contain only a REPORT HEADING or a REPORT FOOTING report group.

A SOURCE clause or a USE declarative statement for a PAGE HEADING report group cannot reference a break control item, an item containing or subordinate to it, or an item that redefines or renames any part of a break control item.

CONTROL HEADING

The CONTROL HEADING phrase specifies a report group that is presented when a control break occurs. Several CONTROL HEADING report groups can be specified, each with a different data-name-1 or FINAL.

FINAL specifies that the report group is to be presented once per report, during execution of the first GENERATE statement for the report. Only one TYPE CONTROL HEADING FINAL clause can appear for a report.

Data-name-1 specifies a break control item that is defined in a CONTROL clause. If a control break occurs during GENERATE statement execution, the CONTROL HEADING report groups are written, starting at the lowest level group and continuing through the group at which a control break occurred. The order of the data-names in the CONTROL clause establishes the relative high and low levels.

A page advance, which includes the processing of PAGE FOOTING and PAGE HEADING report groups, occurs as needed before the CONTROL HEADING report group is presented.

DETAIL

The DETAIL phrase specifies a report group that is presented when a corresponding GENERATE statement is executed. A DETAIL report group is required if the GENERATE data-name statement is used; for a GENERATE report-name statement, a DETAIL report group is not required and no more than one can appear in the Report Description entry.

When a DETAIL report group is processed, subtotaling is performed as designated by the DETAIL report group. A GENERATE data-name statement also causes a report group to be presented as output for the report file, but a GENERATE report-name does not.

A page advance, which includes the processing of PAGE FOOTING and PAGE HEADING report groups, occurs as needed before the DETAIL report group is presented.

CONTROL FOOTING

The CONTROL FOOTING phrase specifies a report group that is presented at the end of a control group for a designated data-name. If CONTROL FOOTING FINAL is specified, the report group is presented once per report as the last body group of the report.

Data-name-2 specifies a break control item that is defined in a CONTROL clause.

If a control break occurs during GENERATE statement execution, the CONTROL FOOTING report groups are written, starting at the lowest level group and continuing through the group at which the control break occurred. The order of the data-names in the CONTROL clause establishes the relative high and low levels.

A page advance, which includes the processing of PAGE FOOTING and PAGE HEADING report groups, occurs as needed before the CONTROL FOOTING report group is presented.

During processing of a CONTROL FOOTING report group, the sum counter is crossfooted and rolled forward. After the CONTROL FOOTING report group is presented, the sum counter is reset. The resetting of the sum counter takes place even if no control footing is defined for a given break control data-name.

A SOURCE clause or a USE declarative statement for a CONTROL FOOTING report group cannot reference an item containing or subordinate to a break control item, or an item that redefines or renames any part of a break control item.

PAGE FOOTING

The PAGE FOOTING phrase specifies a report group that is presented as the last report group on each page, except a page designated to contain only a REPORT HEADING report group or only a REPORT FOOTING report group. The phrase is valid only when a PAGE clause appears in the RD entry. When a PAGE FOOTING report group is presented on a page that contains a REPORT FOOTING report group, it is the second last report group on the page.

A SOURCE clause or a USE declarative statement for a PAGE FOOTING report group cannot reference a break control item, an item containing or subordinate to it, or an item that redefines or renames any part of a break control item.

REPORT FOOTING

The REPORT FOOTING phrase specifies a report group that is presented once for each report, as the last report group of the report. The REPORT FOOTING report group is presented during the execution of a corresponding TERMINATE statement if at least one GENERATE statement was executed for the report.

A SOURCE clause or a USE declarative statement for a REPORT FOOTING report group cannot reference an item containing or subordinate to a break control item or an item that redefines or renames any part of a break control item.

USAGE Clause

The USAGE clause (figure 6-14) is documentary only in a Report Group Description entry.

```
[USAGE IS] DISPLAY
```

Figure 6-14. USAGE Clause Format

VALUE Clause

The VALUE clause (figure 6-15) defines the value of a constant for a printable item. The clause is format 1 of the VALUE clause discussed in section 4, Data Division. Information in section 4 also applies to the VALUE clause used in a Report Group Description entry.

The value of literal is used each time its report group is printed, subject to the qualifications of a GROUP INDICATE clause also present for the entry.

```
VALUE IS literal
```

Figure 6-15. VALUE Clause Format

REPORT WRITER STATEMENTS

Report Writer statements in the Procedure Division generate the report file. Only Report Writer statements and OPEN and CLOSE can reference the report file.

Before any Report Writer statement is executed, the report file must be opened by an OPEN statement with an OUTPUT or EXTEND phrase. After all Report Writer statements, the report file should be closed by a CLOSE statement without either the REEL or UNIT phrase.

GENERATE STATEMENT

The GENERATE statement (figure 6-16) directs the Report Writer to produce a report to the specifications of the corresponding RD entry. Depending on the element in the statement, either a summary report or a single DETAIL report group is processed. The GENERATE statement must appear between an INITIATE statement and a TERMINATE statement.

$$\text{GENERATE} \begin{Bmatrix} \text{data-name} \\ \text{report-name} \end{Bmatrix}$$

Figure 6-16. GENERATE Statement Format

The first GENERATE statement executed for a report saves the report's break control item values; subsequent GENERATE statements cause a check for a control break. When a control break occurs, control processing occurs and the new break control item values are saved to check for subsequent control breaks.

The first GENERATE statement for a given report processes report groups in the following order, when applicable:

REPORT HEADING report group

PAGE HEADING report group

CONTROL HEADING report groups

DETAIL report group.

Subsequent GENERATE statements cause a check for control break, when applicable:

CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses access the break control item values.

CONTROL FOOTING report groups then are processed in the order of low level to high level, ending with the level at which the control break occurred.

CONTROL HEADING report groups are processed in the order of high level to low level, beginning with the level at which a control break occurred.

If GENERATE data-name is specified, the DETAIL report group is also presented as required.

The item specified in the GENERATE statement determines whether an entire report or a single report group is to be written to the report file:

Report-name

Names a report for which summary processing is to be performed. Report-name can be specified if the corresponding RD entry contains all of the following:

A CONTROL clause.

Not more than one DETAIL report group.

At least one body group.

Data-name

Names a DETAIL report group. The data-name can be qualified by a report-name.

During summary processing, no DETAIL report groups are written to the report file, but subtotaling is performed if designated by an associated DETAIL report group.

During DETAIL processing, Report Writer performs all processing of all other report groups, when applicable.

INITIATE STATEMENT

The INITIATE statement (figure 6-17) initializes each report named. These functions are performed:

All sum counters are set to zero.

```
INITIATE report-name-1  [, report-name-2] . . .
```

Figure 6-17. INITIATE Statement Format

The special register LINE-COUNTER is set to zero.

The special register PAGE-COUNTER is set to 1.

The INITIATE statement must be preceded by an OPEN statement, with either an OUTPUT or EXTEND phrase, that references the file associated with the report.

In the statement, each report-name specifies a report defined by a Report Description entry.

A subsequent INITIATE statement must not be executed for a given report-name unless an intervening TERMINATE statement executes.

SUPPRESS STATEMENT

The SUPPRESS statement (figure 6-18) inhibits the presentation of a report group. It can only appear in a USE BEFORE REPORTING procedure, and must appear in the USE procedure that names the report group for which suppression is to be performed.

```
SUPPRESS PRINTING
```

Figure 6-18. SUPPRESS Statement Format

The SUPPRESS statement inhibits the following report group functions:

Presentation of the print lines of the report group.

Processing of all LINE clauses in the report group.

Processing of the NEXT GROUP clause in the report group.

Adjustment of the special register LINE-COUNTER.

TERMINATE STATEMENT

The TERMINATE statement (figure 6-19) completes processing of the specified reports. During termination processing, report groups for each report are presented as specified with automatic page advancing and presentation of the PAGE HEADING and PAGE FOOTING report groups as required. The TERMINATE statement must be executed for each report that is initialized by the INITIATE statement.

The TERMINATE statement does not close the file. A CLOSE statement must be issued for the file after each report in an initiated condition is named in a TERMINATE statement.

```
TERMINATE report-name-1 [, report-name-2] . . .
```

Figure 6-19. TERMINATE Statement Format

In the statement, each report-name names a report for which termination processing is to be executed.

If a GENERATE statement was executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement, the following report groups are presented in the order shown:

CONTROL FOOTING report groups, in the order of low level to high level, using the prior set of control values as though a control break occurred.

PAGE FOOTING report group.

REPORT FOOTING report group, using prior control values as though a control break occurred.

USE BEFORE REPORTING DECLARATIVE

The USE BEFORE REPORTING declarative (figure 6-20) specifies Procedure Division statements that are to be executed just before a report group is written to the report file. The USE BEFORE REPORTING procedure is executed just before any spacing associated with the report group is completed.

```
USE BEFORE REPORTING identifier.
```

Figure 6-20. USE BEFORE REPORTING Declarative Format

In the declarative, identifier specifies the data-name of a report group for which the USE BEFORE REPORTING procedure is to be executed. The TYPE clause defines the report group. A report group cannot appear in more than one USE BEFORE REPORTING procedure.

The USE BEFORE REPORTING procedure cannot contain a GENERATE, INITIATE, or TERMINATE statement, or any statement that alters the value of any break control item associated with the report containing the corresponding report group. In the procedure, statements must not reference a group data item that contains or is subordinate to a break control item, and must not reference an item that redefines or renames any part of a break control item.

See the USE statement with the Procedure Division for additional information about declaratives.

SORT/MERGE FACILITY

The Sort/Merge facility sorts or merges records according to a set of keys contained within those records. The sort operation orders one or more files; the merge operation combines two or more identically ordered files.

Records to be sorted can originate in one of two ways:

> When the USING phrase of the SORT statement is specified, the records to be sorted must reside in the files named in the phrase. All records in the files are sorted. Each file must be described in an FD entry in the File Section. The SD entry in the File Section redescribes the records such that the sort keys are identified.
>
> When the INPUT PROCEDURE phrase of the SORT statement is specified, the program must contain an input procedure that identifies each individual record to be sorted. In this case, an input file, as such, does not exist. Only records referenced in a RELEASE statement within an input procedure are processed by the SORT statement. The input procedure in the program obtains records from external files or otherwise creates the records individually. Each record referenced in a RELEASE statement is accumulated by the sort routines until the procedure is completed; the sort then occurs. The SD entry associates a Record Description with a particular SORT statement and identifies the keys for the sort.

Records to be merged must reside in files.

Results of a sort or merge can be returned to the program in one of two ways:

> When the GIVING phrase of a SORT or MERGE statement is specified, the sorted or merged records exist as an output file. This file must be described in an FD entry in the File Section.
>
> When the OUTPUT PROCEDURE phrase of the SORT or MERGE statement is specified, the system transfers control to an output procedure within the program at the conclusion of the sort or merge operation. One ordered record is released to the procedure each time a RETURN statement executes. The program is then responsible for writing the record to a file or otherwise processing or preserving the ordered records.

The sort or merge operation uses the record description in the SD entry of the File Section to determine the size of records to be sorted and the location of keys within each record.

The following Procedure Division statements are involved with sort and merge operations:

SORT	Order records
MERGE	Combine records from ordered files
RELEASE	Transfer individual record to initial phase of sort
RETURN	Obtain individual record from final phase of sort or merge

During the execution of a SORT or MERGE statement, a block of central memory is allocated for the sort or merge operation. The default size of this block is 9216_{10} words. Generally, the default size is sufficient; however, more efficiency can be achieved by increasing the memory size for large sort/merge operations. To regulate the memory size, the following statement must be executed:

```
    ENTER "C.SORTP" USING fl, init-sq.
```

The memory size specified by fl is used for all subsequent sort/merge operations. Init-sq applies to subsequent sort operations; it indicates whether initial sequence is maintained for duplicate sort keys. If init-sq is omitted or is zero, initial sequence is not maintained; if a value other than zero is specified, initial sequence is maintained. If only the init-sq. is to be changed, the default size (fl) must be specified. When the ENTER "C.SORTP" statement is not used, duplicate values for all sort keys are returned in the initial sequence. Both fl and init-sq must be COMP-1 items.

SORT-MERGE DESCRIPTION ENTRY

The Sort-Merge Description entry in the File Section identifies the records to be sorted or merged. The following Record Description entry describes the size of the records and identifies the key or keys to be used during the sort or merge.

The Sort-Merge Description entry itself (figure 7-1) begins with the level indicator SD. File-name, which must immediately follow SD, associates the entry with a particular SORT or MERGE statement. Only Sort/Merge Procedure Division statements can reference file-name.

The file-name used in the SD entry must be named in the Environment Division in a FILE-CONTROL paragraph SELECT clause. The SELECT clause and the ASSIGN clause are the only ones allowed in the paragraph. If the memory area is to be shared by different files during sort or merge processing, the I-O-CONTROL paragraph must be included in the Environment Division.

The two clauses of the SD entry are optional and can appear in any order. RECORD CONTAINS is redundant, since the following Record Description entries establish actual record size. DATA RECORDS is documentary only. See the File Description entry in the Data Division for more clause information.

Record Description entries that follow the SD entry establish the maximum size of records to be processed. Record length should correspond to the size of the largest record to be processed; the system truncates any records found to exceed the size specified. Minimum record size for fixed length records (RT=F) to be merged should be 10 characters. Smaller records result in one extra record at the end of the output file for each input file merged. When more than one Record Description entry follows a given SD entry, all records occupy the same record area but the entry redefines the area. The record-name in the Record Description entry establishes the location of the record referenced in any RELEASE or RETURN statement in an input procedure or output procedure.

SORT/MERGE STATEMENTS

The Sort/Merge statements in the Procedure Division are as follows. See section 8, Segmentation Facility, for information about SORT and MERGE statements in segmented programs.

SORT STATEMENT

The SORT statement (figure 7-2) causes records from an input procedure or input files to be sorted according to record descriptions in the associated SD entry. The statement can appear anywhere in the Procedure Division other than in the declaratives, in an input procedure, or in an output procedure.

No more than one file of a multifile tape reel can be referenced in the statement.

File-name-1 must be the same as file-name in an SD entry in the File Section of the Data Division.

The order in which records are to be sequenced is established by the keywords ASCENDING and DESCENDING.

ASCENDING

Sort order is from the lowest value of the key data-name to the highest value.

DESCENDING

Sort order is from the highest value of the key data-name to the lowest value.

Values are ordered in accordance with the rules for comparison of operands in a relation condition, as discussed in section 1.

The keys on which the records are to be sequenced are specified by the data-names in the statement. The first data-name specified is the most significant and the last data-name specified is the least significant, with data-names ordered by significance between the first and last.

```
SD file-name

  [          ( CONTAINS [integer-1 TO] integer-2 CHARACTERS )  ]
  [ ; RECORD ) IS VARYING IN SIZE [[FROM integer-3]         (  ]
  [          )      [TO integer-4] CHARACTERS]              (  ]
  [          (      [DEPENDING ON data-name-1]              )  ]

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

Figure 7-1. SD Entry Format

```
SORT file-name-1 ON {DESCENDING} KEY data-name-1 [, data-name-2] . . .
                    {ASCENDING }

      [ON {DESCENDING} KEY data-name-3 [, data-name-4] . . .] . . .
      [   {ASCENDING }                                      ]

[WITH DUPLICATES IN ORDER]

[COLLATING SEQUENCE IS alphabet-name]

{INPUT PROCEDURE IS section-name-1 [{THRU   } section-name-2]}
{                                  [{THROUGH}               ]}
{USING file-name-2 [, file-name-3] . . .                     }

{OUTPUT PROCEDURE IS section-name-3 [{THRU   } section-name-4]}
{                                   [{THROUGH}               ]}
{GIVING file-name-4                                           }
```

Figure 7-2. SORT Statement Format

Key data-names can be qualified. Each data item identified must be described in at least one record of the associated SD entry. The data items must be fixed length; they cannot contain an OCCURS clause or be subordinate to an entry that contains an OCCURS clause. The data items cannot be described with a USAGE clause specifying COMP-4. The data items cannot be boolean items.

When records have the same values for all keys on which the records are being sequenced, the records are sequenced in the order that they were released to the sort operation. This initial sequence option requires 10 extra characters per record, which can be a significant amount of overhead. If initial sequence is not needed, the C.SORTP routine should be entered with the init-sq parameter omitted to remove this option and reduce the overhead.

NOTE

Refer to appendix F for recommendations on the use of the SORT statement.

DUPLICATES Phrase

The DUPLICATES phrase provides compatibility with future ANSI Standards, and is documentary only.

COLLATING SEQUENCE Phrase

The alphabet-name specifies the collating sequence that is used in the comparisons of nonnumeric key data items. However, execution of a SET statement with a SORT or SORT-MERGE COLLATING SEQUENCE phrase prior to execution of the SORT statement causes the collating sequence of the SET statement to override the collating sequence specified in the SORT statement.

Alphabet-name must be defined in an ALPHABET clause in the SPECIAL-NAMES paragraph of the Environment Division.

If the COLLATING SEQUENCE phrase is not specified and the collating sequence is not established by a SET statement prior to execution of the SORT statement, the collating sequence used for the sort is determined by the COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph in the Environment Division.

INPUT PROCEDURE Phrase

The INPUT PROCEDURE phrase specifies the procedure that is to be executed under system control at the time the SORT statement is executed. Control passes to the input procedure through the SORT statement; the input procedure must not be entered directly. The procedure can include statements needed to select, create, or modify records; it must include the execution of a RELEASE statement for each record to be passed to the initial phase of a sort operation.

The input procedure must include one or more sections. Section-name-1 is the first or only section name. If more than one section is included, the sections must appear contiguously in the program and section-name-2 must specify the last section of the input procedure. The sections of the input procedure must not form part of any output procedure.

The input procedure must not contain a SORT statement, MERGE statement, or any statement such as GO TO, ALTER, or PERFORM that transfers control outside the input procedure. Statements that implicitly transfer control to declaratives are allowed.

USING Phrase

The USING phrase specifies the names of the files that are to be sorted. These files are used as input and are sorted according to the descriptions in the SD entry describing file-name-1. If file-name-2 and file-name-3 are both specified, these files are, in effect, concatenated and treated as a single file; the result is one sorted file (file-name-4). File-name-2 and file-name-3 must be identified in FD entries in the File Section of the Data Division. The associated SELECT clause in the FILE-CONTROL paragraph of the Environment Division can specify a file with sequential organization or with any other organization.

The input files must not be in open mode at the time the SORT statement is executed. The USING phrase of the SORT statement causes the following actions to be taken for each named input file:

Initiation of the file is performed as if an OPEN statement with the INPUT phrase were executed.

Each record is obtained and released to the sort operation as if a READ statement with the NEXT and the AT END phrases were executed.

Termination is performed as if a CLOSE statement without optional phrases were executed.

Any USE procedures associated with these functions are executed in the normal manner.

Input files named in the USING phrase must not be referenced by statements in the output procedure or a USE AFTER ERROR declarative procedure.

GIVING Phrase

The GIVING phrase specifies the output file to hold sorted records. File-name-4 names the output file, which must be described in an FD entry in the File Section of the Data Division with a record size equal to the size specified for the records sorted.

The FILE-CONTROL paragraph entry in the Environment Division must specify an actual-key, indexed, relative, or sequential file organization. If file-name-4 references an indexed file, then data-name-1 must be associated with the ASCENDING phrase and must be the primary key for that file.

The output file must not be in open mode at the time the SORT statement is executed. The GIVING phrase of the SORT statement causes the following actions to be taken for the output file:

Initialization of the file is performed as if an OPEN statement with the OUTPUT phrase were executed.

Each sorted record is returned and written to the output file as if a WRITE statement without optional phrases were executed.

Termination of the file is performed as if a CLOSE statement without optional phrases were executed.

Any USE procedures associated with these functions are executed in the normal manner.

An attempt to write beyond the boundaries of the output file for which no USE AFTER ERROR declarative is specified or from which control was returned from a USE AFTER ERROR declarative procedure causes the file to be terminated as if a CLOSE statement without optional phrases were executed.

The output file must not be manipulated by statements in an input procedure or USE AFTER ERROR declarative procedure.

OUTPUT PROCEDURE Phrase

The OUTPUT PROCEDURE phrase specifies the procedure that is to be executed under system control after records have been sorted. Control passes to the output procedure through the SORT statement; the output procedure must not be entered directly. The procedure can include statements needed to select, modify, or copy records; it must include the execution of a RETURN statement for each record to be obtained from the final phase of the sort operation.

The output procedure must include one or more sections. Section-name-3 is the first or only section name. If more than one section is included, the sections must appear contiguously in the program and section-name-4 must specify the last section of the output procedure. The sections of the output procedure must not form part of any input procedure.

The output procedure must not contain a SORT statement, MERGE statement, or any statement such as GO TO, ALTER, or PERFORM that explicitly transfers control outside the output procedure. Statements that implicitly transfer control to declaratives are allowed.

MERGE STATEMENT

The MERGE statement (figure 7-3) combines two or more identically sequenced sequential files on a set of specified keys and makes the merged records available to either an output procedure or an output file. The statement can appear anywhere in the Procedure Division other than in the Declaratives section, an input procedure, or an output procedure.

File name-1 must be described in an SD entry in the File Section of the Data Division.

Only one file from a multifile reel can be referenced in a given MERGE statement, and no file can be named in a MERGE statement more than once.

The order in which records are merged is established by the keywords ASCENDING and DESCENDING. If the specification conflicts with the sequenced order of the input files, the result of the merge operation is undefined.

ASCENDING

Input files are ordered from the lowest key data-name values to the highest values.

DESCENDING

Files are ordered from high values to low values of the key data-name.

During the merge operation, the values are ordered in accordance with the rules for comparison of operands in a relation condition.

The keys on which the input files are sequenced are specified by data-name. The data-names specify the keys and their significance, with the first key specified being the most significant key and the last key the least significant key. If the specification conflicts with the sequences of the input files, the result of the merge operation is undefined.

Key data-names can be qualified. Each data item identified must be described in at least one record of the associated SD entry. The data items must be fixed length; they cannot contain an OCCURS clause or be subordinate to an entry that contains an OCCURS clause. The data items cannot be described with a USAGE clause specifying COMP-4.

```
MERGE file-name-1 ON {DESCENDING} KEY data-name-1 [, data-name-2] . . .
                     {ASCENDING }

   [ON {DESCENDING} KEY data-name-3  [, data-name-4] . . .] . . .
   [   {ASCENDING }                                        ]

[COLLATING SEQUENCE IS alphabet-name]

USING file-name-2, file-name-3 [, file-name-4] . . .

{OUTPUT PROCEDURE IS section-name-1 [{THRU   } section-name-2]}
{                                   [{THROUGH}               ]}
{GIVING file-name-5                                            }
```

Figure 7-3. MERGE Statement Format

COLLATING SEQUENCE Phrase

The alphabet-name specifies the collating sequence that is used in the comparison of nonnumeric key data items. However, execution of a SET statement with a MERGE or SORT-MERGE COLLATING SEQUENCE phrase prior to execution of the MERGE statement causes the collating sequence of the SET statement to override the collating sequence in the MERGE statement. The alphabet-name must be defined in an ALPHABET clause in the SPECIAL-NAMES paragraph of the Environment Division.

If the COLLATING SEQUENCE phrase is not specified and the collating sequence is not established by the SET statement prior to execution of the MERGE statement, the collating sequence used for the merge is determined by the COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph in the Environment Division.

USING Phrase

The USING phrase specifies the names of the files to be merged. File-name-2 and file-name-3 are used as input files for the merge operation. These files must be described in FD entries in the Data Division, must have sequential organization, and must have records that are equal to the size of the records specified in the SD entry describing file-name-1. The records of these files must be ordered according to the keys specified in the MERGE statement; otherwise, the order of the records in the resultant file is undefined and a diagnostic is written to the dayfile.

The input files must not be in open mode at the time the MERGE statement is executed. The USING phrase of the MERGE statement causes the following actions to be taken for each named input file:

Initiation of the file is performed as if an OPEN statement with the INPUT phrase were executed.

Each record is obtained and released to the merge operation as if a READ statement with the NEXT and the AT END phrases were executed.

Termination is performed as if a CLOSE statement without optional phrases were executed.

Any USE procedures associated with these functions are executed in the normal manner.

The input files named in the USING phrase must not be referenced by statements in either an output procedure or a USE AFTER ERROR declarative procedure.

OUTPUT PROCEDURE Phrase

The OUTPUT PROCEDURE phrase specifies the procedure that is to be executed under system control after records have been merged. Control passes to the output procedure through the MERGE statement; the output procedure must not be entered directly. The procedure can include statements that select, modify, or copy records as they are returned in their merged order; it must include the execution of a RETURN statement for each record to be obtained from the final phase of the merge operation.

The output procedure must include one or more sections. Section-name-3 is the first or only section name. If more than one section is included, the sections must appear contiguously in the program and section-name-2 must specify the last section within the output procedure. The sections of the output procedure must not form part of any other input or output procedure.

The output procedure must not contain a SORT statement, MERGE statement, or any statement such as ALTER, GO TO, or PERFORM that explicitly transfers control outside the output procedure. Statements that implicitly transfer control to declaratives are allowed.

Once the OUTPUT PROCEDURE has completed execution, control returns to the statement following the MERGE statement.

GIVING Phrase

The GIVING phrase specifies the file to which all the ordered records from the merge file are written. File-name-5 names the output file, which must be described in an FD entry in the File Section of the Data Division with a record size equal to the size specified for the records in the SD entry.

File-name-5 can specify a file with indexed, actual-key, sequential or relative file organization. When the output file has indexed organization, data-name-1 must specify the primary key and the MERGE statement must specify ASCENDING.

The output file must not be in open mode at the time the MERGE statement is executed. The GIVING phrase of the MERGE statement causes the following actions to be taken for the output file:

Initialization of the file is performed as if an OPEN statement with the OUTPUT phrase were executed.

Each merged record is returned and written onto the output file as if a WRITE statement without optional phrases were executed.

Termination of the file is performed as if a CLOSE statement without optional phrases were executed.

Any USE procedures associated with these functions are executed in the normal manner.

An attempt to write beyond the boundaries of the output file for which no USE AFTER ERROR declarative procedure is specified or from which control was returned causes the file to terminate as if a CLOSE statement without optional phrases were executed.

The output file must not be manipulated by statements in an output procedure or a USE AFTER ERROR declarative procedure.

RELEASE STATEMENT

The RELEASE statement (figure 7-4) transfers records to the initial phase of a sort operation. It must be specified within the range of an input procedure associated with a SORT statement.

```
RELEASE record-name [FROM identifier]
```

Figure 7-4. RELEASE Statement Format

The record that is transferred by the RELEASE statement is not available in the record area of the associated SD entry after execution of the RELEASE statement, unless the SD entry file-name is named in a SAME RECORD AREA clause in the I-O-CONTROL paragraph of the Environment Division. Specification of the SAME RECORD AREA clause also makes the record available to other files referenced in the SAME RECORD AREA clause.

In the statement, record-name must be the name of a record in the associated SD entry. The name can be qualified.

The FROM phrase specifies that the contents of the storage area named by identifier is moved in accordance with the rules of the MOVE statement to the storage area named by record-name before the contents of record-name is released to the sort.

The record released to the sort is not available in the record area of the SD entry after execution of the RELEASE statement unless the SAME RECORD AREA clause of the I-O-CONTROL paragraph is specified. In any case, the record is available in identifier after execution of the RELEASE statement.

RETURN STATEMENT

The RETURN statement (figure 7-5) obtains the next record in the order specified by the keys listed in either a SORT or a MERGE statement and makes the record available for processing in the record area associated with the SD entry. The statement must be specified within the range of an output procedure associated with either a SORT statement or a MERGE statement.

```
RETURN file-name RECORD [INTO identifier]

; AT END imperative-statement
```

Figure 7-5. RETURN Statement Format

Because the records of the SD entry can be of variable lengths, the contents of the current data record can be smaller than the record area allocated. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

The imperative-statement of AT END specifies the action to be taken after the last record is returned by either a sort or a merge operation.

No RETURN statement can be executed within the current output procedure after execution of imperative-statement.

The INTO phrase specifies that the current record is moved in accordance with the rules of the MOVE statement to the storage area named. Identifier names the storage area; it can be subscripted or indexed. The identifier and the file-name must name different storage areas.

The phrase can be used when the SD entry for the file is followed by one or more Record Description entries that specify fixed length or variable-length records. When more than one Record Description entry exists for the file, however, the phrase cannot be used if any elementary level 01 record defines a numeric or numeric edited item.

SEGMENTATION FACILITY

Segmentation allows the relocatable binary output of source program compilation to be split into overlays, so that not all of the object program need be in memory at one time during execution. Segmentation is a feature defined by COBOL; the words segment and overlay are used here in their COBOL context. In the context of the operating system loader, segments and overlays have related, but not identical, meanings. The following discussion deals only with COBOL programs.

Segmentation does not affect the logic of program execution. The logical sequence of statements in the Procedure Division is the same as the physical sequence, except for specific transfers of control. Control can be transferred to any paragraph in a section within any segment. Further, segmentation does not affect the need to uniquely qualify all program references.

When segmentation is selected, the entire Procedure Division of the source program must be written as a consecutive grouping of sections. Each section must be designated as part of a specific segment through the segment-number in the section header. The segment-number specifies the type of the segment in addition to identifying a particular segment.

The two types of segments are:

Fixed segments, composed of sections with segment-numbers 0 through 49.

Independent segments, composed of sections with segment-numbers 50 through 99.

NOTE

Refer to appendix F for recommendations on the use of segment numbers.

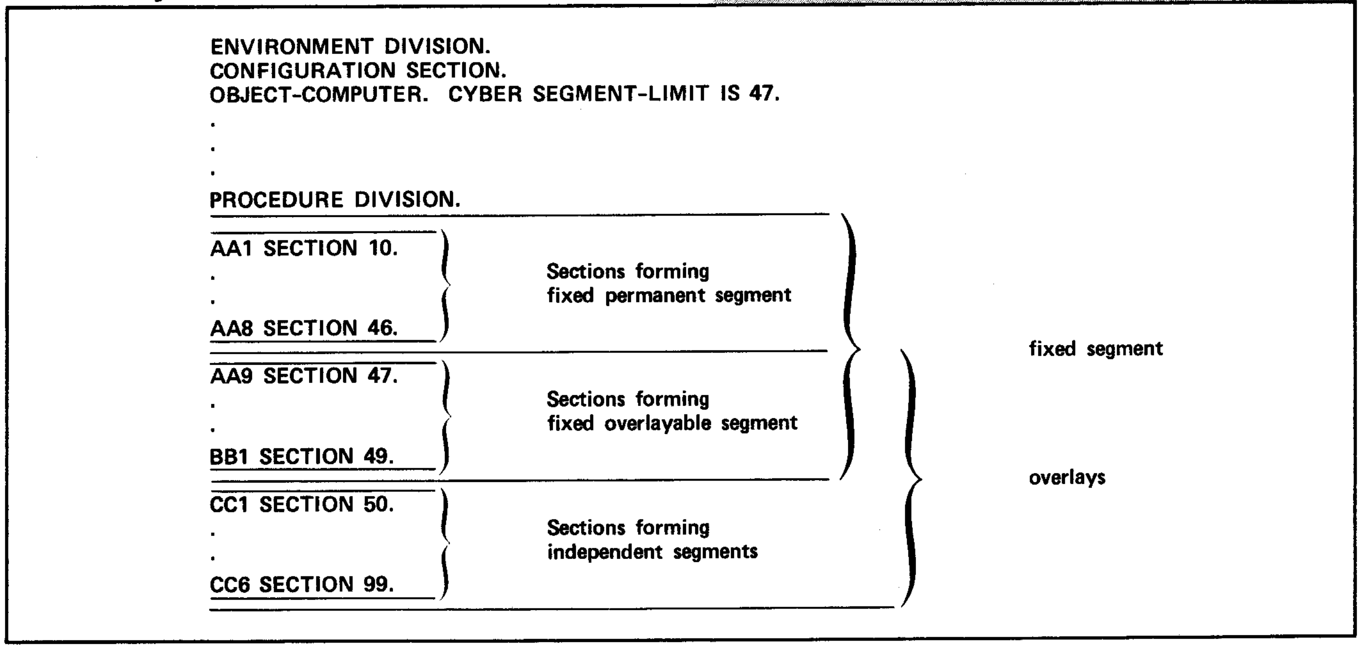
Fixed segments are the portion of the object program that is logically treated as if it were always in memory. Two types of fixed segments exist: permanent and overlayable. The SEGMENT-LIMIT clause of the OBJECT-COMPUTER paragraph defines the segment-numbers that apply to each type of fixed segment.

Fixed permanent segments cannot be overlaid by any other part of the program. Sections that must be available for reference at all times must be within fixed permanent segments. Ideally, all fixed segments are fixed permanent segments.

Fixed overlayable segments can overlay and can be overlaid by another fixed overlayable segment or an independent segment. Fixed overlayable segments are always made available in their last used state.

Independent segments are the portion of the object program that can overlay, and can be overlaid by, other independent segments or fixed overlayable segments. An independent segment is in its initial state whenever control is transferred to a section or paragraph within it as a result of a PERFORM, USE, SORT, MERGE, or GO TO statement in another segment and whenever control is transferred implicitly as a result of the execution of the last statement in the immediately preceding segment. Otherwise, an independent segment is in its last used state. Figure 8-1 shows an example of a program structured with segmentation.

When BBH=YES is specified in the USE clause of a File-Control entry or in a FILE control statement, the associated file must be opened and closed within one segment; no other overlayable segments can be called while the file is open.



```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.  CYBER SEGMENT-LIMIT IS 47.
.
.
.
PROCEDURE DIVISION.
AA1 SECTION 10.
.
.
AA8 SECTION 46.
AA9 SECTION 47.
.
.
BB1 SECTION 49.
CC1 SECTION 50.
.
.
CC6 SECTION 99.
```

Sections forming fixed permanent segment
Sections forming fixed overlayable segment
Sections forming independent segments
fixed segment
overlays

Figure 8-1. Segmented Progam Skeleton Example

If a subprogram is segmented, the program is not actually broken up into segments. The ALTER statements are caused to function properly by internal methods; therefore, no advantage in memory space is gained and some is lost. For this reason, segmentation should not be used in subprograms.

When a subprogram is called by a literal from a fixed overlayable or independent segment in the main program and the program-name is not in the FDL list of programs, the following steps must be taken:

The subprogram must reside on a user library.

The user library must be made available to the job before loading of the calling program.

If these steps are taken, the subprogram is initialized every time the overlay is loaded. If this is not desired, the subprogram must be loaded with the fixed permanent segments. This can be accomplished either with an LDSET,USE control statement or through an unexecuted ENTER or CALL statement in any segment with a segment number less than that specified in the SEGMENT-LIMIT clause.

SEGMENT-LIMIT CLAUSE

The SEGMENT-LIMIT clause (figure 8-2) in the OBJECT-COMPUTER paragraph establishes the range of numbers valid for the two types of fixed segments.

```
SEGMENT-LIMIT IS segment-number
```

Figure 8-2. SEGMENT-LIMIT Clause Format

The segment-number must have an unsigned integer value of 1 through 49. It specifies the lowest numbered segment that is to be considered as a fixed overlayable segment. Segments numbered with values less than the segment-number in the clause are considered as fixed permanent segments.

If the SEGMENT-LIMIT clause is omitted, segment numbers 0 through 49 are considered as fixed permanent segments.

PROCEDURE DIVISION CODING

The Procedure Division must be organized in sections when the division is segmented. Procedures referenced in particular statements within the division are restricted in location.

SECTION HEADER

Sections in the Procedure Division are assigned to a given segment by the section header (figure 8-3). All sections with the same segment-number become part of the same segment.

```
section-name SECTION [segment-number] .
```

Figure 8-3. Section Header Format

Each section-name must be a user-defined word. The segment-number must be an integer 0 through 99; if omitted, 0 is assumed. More than one section can have the same segment-number assigned.

The segment-number for a given section should be assigned according to section use in the program:

Sections that frequently communicate with one another should have the same segment-number.

Sections referred to more frequently should have lower segment numbers than sections referred to less frequently.

Sections that must be available for reference at all times should be within a fixed permanent segment.

Sections referred to very frequently should be within a fixed permanent segment.

Declarative sections must contain segment-numbers less than 50.

ALTER STATEMENTS

The following restriction applies to segmentation of programs that contain ALTER statements: if the paragraph of procedure-name-2 is in a section with a segment-number 50 through 99, the ALTER statement must be in a section with the same segment-number.

All other uses of ALTER are valid, even if the procedure to be altered is in a fixed overlayable segment.

Modified GO TO statements in independent segments might be returned to their initial states, as noted above for independent segments.

PERFORM STATEMENTS

The following restrictions apply to segmentation of programs that contain PERFORM statements:

If a PERFORM statement appears in a section that is not in an independent segment, it can have within its range, in addition to any Declarative sections whose execution is caused within that range, only one of the following:

Sections and/or paragraphs wholly contained in one or more nonindependent segments.

Sections and/or paragraphs wholly contained in a single independent segment.

If a PERFORM statement appears in an independent segment, it can have within its range, in addition to any Declarative sections whose execution is caused within that range, only one of the following:

Sections and/or paragraphs wholly contained in one or more nonindependent segments.

Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

SORT AND MERGE STATEMENTS

The following restrictions apply to segmentation of programs that contain SORT statements or MERGE statements:

If a SORT or MERGE statement appears in a section that is not in an independent segment, any input procedure or output procedure referenced by the statement must appear either wholly within nonindependent segments or wholly within a single independent segment.

If a SORT or MERGE statement appears in an independent segment, any input procedure or output procedure referenced by the statement must be contained either wholly within nonindependent segments or wholly within the same independent segment as the statement.

Input procedures and output procedures must not transfer control to another segment.

LIBRARY FACILITY 9

COBOL source libraries contain text that is available to the compiler for insertion into a COBOL source program at compile time. The inserted text is treated as part of the source program. COBOL source libraries must be created and maintained on a random program library created by the common product Update.

COPY STATEMENT

The COPY statement (figure 9-1) incorporates text from a COBOL source library into a COBOL source program. The copied text replaces the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period. The entire COPY statement is listed with two dash characters to the right of each line number. Any code introduced by the statement is listed after the statement. The Update identifier for each introduced line is listed starting in column 75 of the line. The only way to identify copied text is by this identifier.

A COPY statement can occur anywhere in the source program that a character-string or a separator can appear, except either within another COPY statement or within copied text. A COPY statement must be written as a sentence.

In the statement, text-name identifies the text that is to be copied; the first nine characters of text-name must specify the Update deck name that contains the required library text. The entire deck is copied into the source program.

OF PHRASE

The OF phrase specifies the logical file name of the file containing the deck text-name. The keywords OF and IN are synonomous and interchangeable. The file must be a random Update program library.

When the phrase is omitted, the deck text-name is assumed to reside on the file identified by the X parameter of the compiler call.

REPLACING PHRASE

The REPLACING phrase specifies changes to be made during the copy. If the phrase is omitted, the text is copied into the source program unchanged.

The operands immediately following the keyword REPLACING identify the library text to be changed. The operands immediately following the keyword BY specify the new text. A single word, literal, or identifier can be directly specified as an operand; more than a single word must be specified with the pseudo-text delimiters.

Pseudo-text specifies a sequence of character strings and/or separators. It must be enclosed in the pseudo-text delimiter, two contiguous equal signs. If a pseudo-text character string is continued, both characters of the delimiter must be on the same line.

Pseudo-text-1 must contain at least one character other than a space and at least one item in addition to any comment lines. It cannot contain debugging lines.

Pseudo-text-2 can be null and it can contain all spaces. If it is all spaces, the library text identified by pseudo-text-1 is replaced with spaces.

A comment line in both pseudo-text-1 and the library text is interpreted as a single space for purposes of matching. Comment lines appearing in pseudo-text-2 and the library text are copied into the source program unchanged.

Debugging lines can be specified in pseudo-text-2 and in the library text. The D in the indicator area is ignored in the library text during the comparison operation.

The identifier, literal, and word are treated as pseudo-text containing only identifier, word, and literal, respectively, for purposes of matching and replacing.

COMPARISON OPERATION

The REPLACING operands are compared to the library text from left to right, beginning with the leftmost library text word. Any separator comma, semicolon, and/or spaces preceding the leftmost library text-word is copied into the COBOL source program before comparison begins.

```
COPY text-name [{OF} library-name]
                {IN}

[REPLACING {   {==pseudo-text-1==}      {==pseudo-text-2==} }      ]
           { , {identifier-1     }  BY  {identifier-2     } } . . .].
           {   {literal-1        }      {literal-2        } }
           {   {word-1           }      {word-2           } }
```

Figure 9-1. COPY Statement Format

If pseudo-text-1 consists solely of either a separator comma or a separator semicolon, the separator character participates in the match as a text word; otherwise, any occurrence of a separator comma or a separator semicolon within either pseudo-text-1 or the library text is considered to be a single space during the comparison operation.

Each sequence of one or more space separators is considered to be a single space in both pseudo-text-1 and the library text.

Each successive REPLACING operand is compared to an equal number of contiguous library text characters until either a match is found or all the REPLACING operands have participated in the match. If no match is found, the leftmost library text word is copied into the COBOL source program unchanged; and the next comparison cycle starts, using the next successive library text word as the leftmost library text word. If a match occurs, the corresponding BY operand text is copied into the source program; and the next comparison cycle starts, using the library text word immediately following the last text word that participated in the match.

The comparison cycles continue until all library text is either copied into the source program or replaced with BY operands.

The COBOL source program text matches the library text if the ordered sequences match character by character.

UPDATE RANDOM PROGRAM LIBRARY USAGE

Update is a system utility used to create, maintain, and update source decks on libraries in compressed symbolic format. Either an entire program or a portion of a program can be maintained on the library.

Each text-name specified in the COBOL COPY statement must be the name of an Update deck or comdeck. The copied text consists of all lines following the DECK or COMDECK directive up to but not including the next DECK, COMDECK, WEOR, or CWEOR directive, or the end of the library. All other lines in the library, including CALL and IF, are not recognized as Update directives. Abbreviations for the directives can be specified.

The usual master control character used in defining a program library is the asterisk; however, a character other than the asterisk can be used. The Update facility is described fully in the Update reference manual.

A job deck used in an Update program library creation run under NOS/BE is shown in figure 9-2. Under NOS the REQUEST and CATALOG control statements must be replaced by appropriate permanent file control statements. The Update library is cataloged with the permanent file name JLIB. When the library is used, the library must be attached to the job before the compiler is called. The compiler call must have the X parameter if the OF phrase is omitted from any COPY statements and the default library is not desired.

Figure 9-3 shows the source listing obtained from a program with two COPY statements.

The first COPY statement causes RECDESC to be copied into the COBOL source program unchanged; the second COPY statement causes RECDESC to be copied into the COBOL source program with the identifier SORT-REC replacing the identifier GEN-REC and the IDENT-3 data item receiving a new name and description of PICTURE 999.

```
PLCREATE.
REQUEST,NEWPL,PF.
UPDATE,F,C=0,N=NEWPL,L=F.
CATALOG,NEWPL,JLIB,ID=MINE.
7/8/9
*DECK RECDESC
    01      GEN-REC.
            02      IDENT-1 PICTURE 9(8).
            02      FILL-1 PICTURE X(5).
            02      IDENT-2.
                    03      ID-2A PICTURE 99.
                    03      ID-2B PICTURE XXX.
            02      FILL-2 PICTURE X(10).
            02      IDENT-3 PICTURE XXX.
6/7/8/9
```

Figure 9-2. Example of Update Creation Run for COBOL Source Library

REPLACE STATEMENT

The REPLACE statement (figure 9-4) is used to replace source program text. Pseudo-text-1 identifies the source text to be changed. Pseudo-text-2 identifies the new text. The inserted text is treated as part of the source program; however, the source program listing is not affected.

A REPLACE statement can be used anywhere in the source program where a character-string occurs. It must be preceded by a separator period except when it is the first statement in a program. A REPLACE statement must be terminated by a separator period.

Except for COPY and REPLACE statements, the syntactic correctness of the source program text cannot be determined until all COPY and REPLACE statements have been processed.

All COPY statements contained in a source program are processed before any REPLACE statements. The text produced as a result of the processing of a REPLACE statement must not contain a REPLACE statement or a COPY statement.

FORMAT 1 REPLACE

Format 1 specifies the text of the source program to be replaced by the corresponding text. Each matched occurrence of pseudo-text-1 in the source program is replaced by the corresponding pseudo-text-2. Pseudo-text-1 must contain one or more text words. Pseudo-text-2 can contain zero, one, or more text words. If pseudo-text-2 contains all spaces, pseudo-text-1 is replaced with spaces. Character-strings within pseudo-text-1 and pseudo-text-2 can be continued on the following line. Both pseudo-text-1 and pseudo-text-2 must be enclosed in the pseudo-text delimiters, two contiguous equal signs.

Debugging lines are permitted in pseudo-text-2; they are not permitted in pseudo-text-1. The D in the indicator area is ignored during the comparison cycle. If a REPLACE statement is specified on a debugging line, the text that results from the processing of the REPLACE statement appears as though it were specified on debugging lines.

FORMAT 2 REPLACE

Format 2, REPLACE OFF, specifies that any text replacement currently in effect is discontinued. A REPLACE statement is in effect from the point at which it is specified until another occurrence of the format 1 REPLACE statement, a REPLACE OFF statement, or the end of the program.

COMPARISON OPERATION

Pseudo-text-1 is compared to an equivalent number of contiguous source program text words, beginning with the leftmost source program text word.

Pseudo-text-1 matches the source program text if, and only if, the text words that form pseudo-text-1 are equal, character for character, to the source program text words. If pseudo-text-1 consists solely of either a separator comma or semicolon, it participates in the match as a text word; otherwise, any occurrence of a separator comma or semicolon in pseudo-text-1 or in the source program text is considered to be a single space during the comparison operation. Each sequence of one or more space separators is considered to be a single space.

If no match occurs, the comparison is repeated with each successive occurrence of pseudo-text-1, until either a match is found or there is no successive pseudo-text-1.

a. **Program Submitted for Compilation**

```
.
.
.
FILE SECTION.
FD  GEN-FILE LABEL RECORD OMITTED
    DATA RECORD IS GEN-REC.
COPY RECDESC.
SD  SORT-FILE DATA RECORD IS SORT-REC.
    COPY RECDESC REPLACING GEN-REC BY SORT-REC
    == IDENT-3 PICTURE XXX == BY == NEW-IDENT PICTURE 999 ==.
PROCEDURE DIVISION.
.
.
.
```

b. **Compiler Call**

```
COBOL5,X=OLDPL.
```

c. **Source listing of Compiled Program**

```
            FILE SECTION.
            FD  GEN-FILE LABEL RECORD OMITTED
                DATA RECORD IS GEN-REC.
--          COPY RECDESC.
            01  GEN-REC.                                                    RECDESC.2
                02   IDENT-1 PICTURE 9(8).                                  RECDESC.3
                02   FILL-1 PICTURE X(5).                                   RECDESC.4
                02   IDENT-2.                                               RECDESC.5
                     03   ID-2A PICTURE 99.                                 RECDESC.6
                     03   ID-2B PICTURE XXX.                                RECDESC.7
                02   FILL-2 PICTURE X(10).                                  RECDESC.8
                02   IDENT-3 PICTURE XXX.                                   RECDESC.9
            SD  SORT-FILE
                DATA RECORD IS SORT-REC.
--              COPY RECDESC REPLACING
--                   GEN-REC BY SORT-REC
--                   == IDENT-3 PICTURE XXX == BY
--                        == NEW-IDENT PICTURE 999 ==.
 R          01  SORT-REC.                                                   RECDESC.2
                02   IDENT-1 PICTURE 9(8).                                  RECDESC.3
                02   FILL-1 PICTURE X(5).                                   RECDESC.4
                02   IDENT-2.                                               RECDESC.5
                     03   ID-2A PICTURE 99.                                 RECDESC.6
                     03   ID-2B PICTURE XXX.                                RECDESC.7
                02   FILL-2 PICTURE X(10).                                  RECDESC.8
 R              02   NEW-IDENT PICTURE 999.                                 RECDESC.9
            PROCEDURE DIVISION.
```

Figure 9-3. COPY Statement Example

When all occurrences of pseudo-text-1 have been compared and no match has occurred, the next source program text word is then considered as the leftmost source program text word, and the comparison cycle starts again with the first occurrence of pseudo-text-1.

Whenever a match occurs between pseudo-text-1 and the source program text, the corresponding pseudo-text-2 replaces the matched text in the source program. The source program text word immediately following the rightmost text word that participated in the match is then considered as the leftmost source program text word. The comparison cycle starts again with the first occurrence of pseudo-text-1.

The comparison operation continues until the rightmost text word in the source program text which is within the scope of the REPLACE statment has either participated in a match or been considered in the comparison cycle.

Comment lines or blank lines occurring in the source program text and in pseudo-text-1 are ignored for purposes of matching.

Format 1

<u>REPLACE</u> {, ==pseudo-text-1== <u>BY</u> ==pseudo-text-2==} . . .

Format 2

<u>REPLACE</u> <u>OFF</u>

Figure 9-4. REPLACE Statement Format

DEBUGGING AIDS

Three separate aids are available for use in debugging program execution. The aids are the Paragraph Trace facility, the Termination Dump facility, and the Debugging facility. They differ in that the Paragraph Trace facility and the Termination Dump facility accumulate information in a file for programmer analysis after execution completes, but the Debugging facility transfers control within the program for processing conditions as they occur.

The Paragraph Trace facility is selected by the DB=TR parameter on the compiler call. It is used to trace program flow: output is a file with messages indicating the name of a paragraph and the time the paragraph was executed. The trace itself is initiated and terminated by directives within the program. The job is responsible for the processing or preservation of the trace message file after the trace is concluded.

The Termination Dump facility is selected by the TDF parameter on the compiler call in conjunction with the C5TDMP control statement. It is used to obtain a formatted map of the contents of data items within COBOL programs. Upon normal or abnormal job termination, the contents of the execution time field length, which have been saved on a system file, are used along with tables saved at compilation time to provide the formatted dump.

The Debugging facility is selected by the DB=DL parameter on the compiler call or by the DEBUGGING MODE clause within the program. It can be used to monitor data items or procedures: control transfers to a debugging section in the Declaratives portion of the Procedure Division under conditions specified in a USE FOR DEBUGGING statement. A special register, DEBUG-ITEM, can be used in the debugging code. The Debugging facility also allows lines within the program to be declared debugging lines so that they execute only when the Debugging facility is selected.

PARAGRAPH TRACE FACILITY

The Paragraph Trace facility provides a history of the order in which paragraphs execute. Time information that is produced as part of the trace output can be used to determine the amount of central processor time required to execute any given paragraph.

The DB=TR parameter on the compiler call is required to compile the paragraph trace facility code. The source program must contain directives that turn the facility on or off during execution. Output is written to a file COBTRFL.

TRACE FILE

The trace file has the logical file name COBTRFL. It resides on mass storage unless the job requests that the file be assigned to another device. The file is unlabeled and has a structure of Z type records with C type blocking. Each record is 50 characters long. The file need not be referenced in a program unless it is processed in the program.

The job is responsible for preserving the file when the program ends.

Both status messages and paragraph trace messages are written to the file.

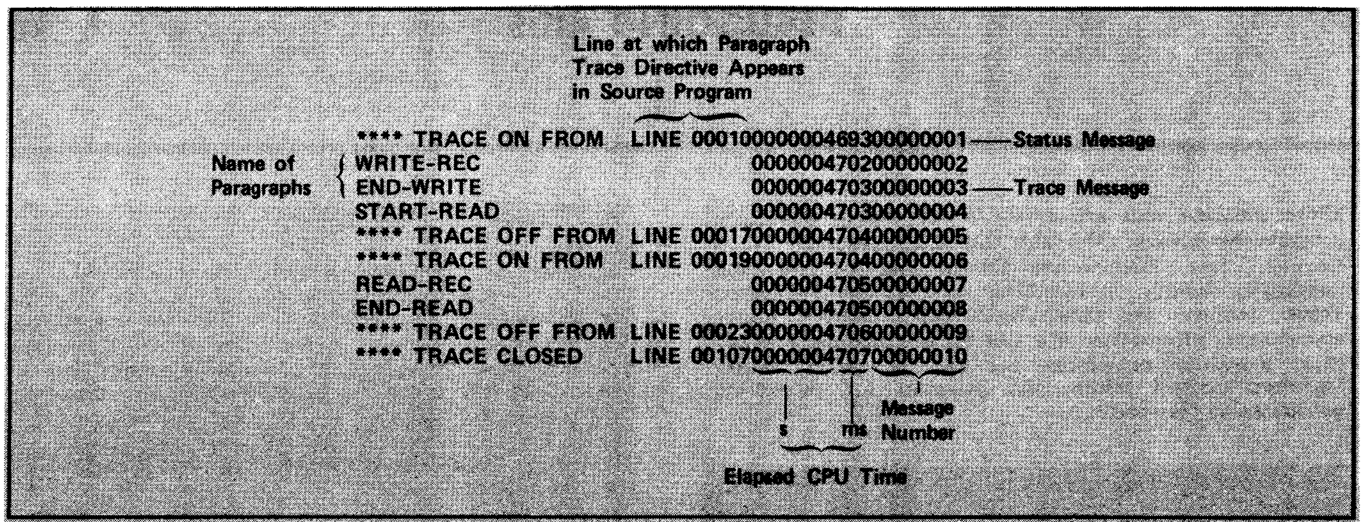
A status message is written to the file each time a Paragraph Trace directive statement executes. Message format is:

Character	Contents
1-4	* * * *
5-30	26 alphanumeric characters of message
31-40	10 numeric digits indicating central processor time used since start of job
41-48	8 numeric characters indicating consecutive number
49-50	Zero byte for Z type record

A paragraph trace message is written to the file each time a new paragraph begins execution. Its format is:

Character	Contents
1-30	30 alphanumeric characters giving paragraph name
31-40	10 numeric digits indicating central processor time used since start of job
41-48	8 numeric characters indicating consecutive number
49-50	Zero byte for Z type record

In both messages, the consecutive number indicates the record number of the message written, starting with 1. The central processor time used since start of job appears with seven digits for seconds followed by three digits for milliseconds. Figure 10-1 shows an example of output on COBTRFL.



```
**** TRACE ON FROM   LINE 00010000000469300000001
WRITE-REC                          000000470200000002
END-WRITE                          000000470300000003
START-READ                         000000470300000004
**** TRACE OFF FROM  LINE 00017000000470400000005
**** TRACE ON FROM   LINE 00019000000470400000006
READ-REC                           000000470500000007
END-READ                           000000470500000008
**** TRACE OFF FROM  LINE 00023000000470600000009
**** TRACE CLOSED    LINE 00107000000470700000010
```

Figure 10-1. Paragraph Trace Output Example

The file can be processed in a program after it has been closed by execution of an ENTER "C.STPTR" statement. The SELECT clause must specify an implementor-name of COBTRFL; the FD entry must specify LABEL RECORD OMITTED and define 50-character records.

TRACE DIRECTIVES

Three directives can be used in the program to control the Paragraph Trace facility. The directives are called by an ENTER statement that specifies a directive as a nonnumeric literal. For example, the facility is turned on by:

```
    ENTER "C.ONTR".
```

A status message is written to the trace file each time one of these directives is executed.

The directives can be used anywhere in the Procedure Division. Directives are:

C.ONTR

Turns on trace. Paragraphs subsequently executed are reported by messages on the trace file.

C.OFFTR

Turns off trace.

C.STPTR

Closes and rewinds the trace file. If omitted, the file is closed by the system when STOP RUN is executed.

C.ONTR and C.OFFTR can be used to accumulate messages from all of a program or only selected parts of a program. If C.ONTR is executed after execution of C.STPTR, the messages from previous trace activity are destroyed.

TERMINATION DUMP FACILITY

The Termination Dump facility provides a formatted map of the contents of all data items of one or more COBOL programs. A map of data items is produced for all COBOL programs that are both in memory at the time the job terminates and are also specified on a Fast Dynamic Loader (FDL) file. The main program, however, need not be specified on an FDL file to be included in the dump. If the Termination Dump facility is used, a dump is taken when a job terminates normally or abnormally.

A termination dump is taken when both of the following are true:

The TDF option appears on the compiler call.

The control statement C5TDMP is specified after the COBOL run (possibly after an EXIT control statement).

If the TDF option is specified, extra information is generated by the compiler which allows a formatted dump to occur at a later time. Tables needed for the dump are written to the file specified by the TDF parameter.

When the C5TDMP statement is executed, the termination dump analysis program uses a system file along with the file named in the TDF parameter to output the formatted map of data items. The system file contains the contents of the field length at job termination.

The format of the C5TDMP statement is:

```
    C5TDMP,L=lfn,T=lfn,I=lfn,NA.
```

The L parameter specifies the logical file name of the file to which the dump listing is written. If the parameter is omitted, the listing is written to the file OUTPUT.

The T parameter specifies the logical file name of the file from which compiler information is obtained. It is the same file name that appears in the TDF parameter on the compiler call. If the T parameter is omitted, the compiler information is obtained from the file TDFILE.

The I parameter specifies that a directive file is available. If I is not specified, it is assumed that there is no directive file, and all items are printed. If I is specified without a logical file name, it is assumed that the directives are on INPUT; otherwise, the directives are assumed to be on the file designated by lfn. The directives are free form, with no punctuation allowed. The only separators are spaces. A data-name cannot be the same word as a directive. The directives are:

<u>EXCEPT</u> {data-name}...

If EXCEPT is used, all items except those specified are printed.

<u>SELECT</u> {data-name}...

If SELECT is used, only the specified items are printed. EXCEPT and SELECT are mutually exclusive. That is, if one is specified, the other cannot be specified. If this is violated, the second one is ignored.

<u>OCCURRENCES</u> aa [<u>TO</u> bb] OF data-name.

If this is specified, only the indicated number of occurrences of data-name are printed. The range of numbers is specified by the integers aa and bb.

The NA parameter indicates that no array items are printed. Any directives can override this parameter for the specified items. Duplicate items within an array are indicated as duplicates by an appropriate message.

The termination dump indicates the last paragraph and line executed if TDF and DB=TR are both specified on the COBOL5 compiler call statement and the trace has been activated by the C.ONTR directive. If DB=TR is not specified, the termination dump indicates the line number of the last input-output statement executed. If DB=TR is not specified and no input-output statement is executed, the message NO I/O DONE appears.

A snap dump can be taken at any time by executing the statement:

ENTER "C.SNAP" USING parameter.

Parameter is either a literal or alphanumeric item of up to 30 characters in length which is printed on each snap dump. These snap dumps are written on the dump file in the order in which they were requested, and can be listed by one C5TDMP control statement, using the one set of controlling input directives.

The formatted map contains three columns. The first indicates the data-name of each elementary data item within the program, including each occurrence of FILLER. The second column indicates the type of data in the item. The types and their meanings are listed in table 10-1. The third column shows the contents of each item. Figure 10-2 shows an example of a termination dump.

TABLE 10-1. DATA TYPES IN TERMINATION DUMP

Type	Meaning
AL	Alphabetic
ALE	Alphabetic Edited
AN	Alphanumeric
ANE	Alphanumeric Edited
BL	Boolean
CP	COMPUTATIONAL
CP1	COMPUTATIONAL-1
CP2	COMPUTATIONAL-2
CP4	COMPUTATIONAL-4
ID	Index Data Item
IN	Index-Name
LC	LINE-COUNTER
NE	Numeric Edited

DEBUGGING FACILITY

The COBOL Debugging facility allows two types of debugging procedures to be included in a program.

Debugging lines, once compiled as executable code, are always executed.

Debugging sections, which are coded only in the Declaratives portion of the Procedure Division, execute only when an execution-time option is selected.

A special register, DEBUG-ITEM, is available for reference when the debugging section is compiled.

DEBUGGING LINES

Debugging lines are source program lines with a D in the indicator area. They can appear in any part of the program after the OBJECT-COMPUTER paragraph. Continuation lines are permitted, but each line must have a D in the indicator area and character-strings cannot be split between lines.

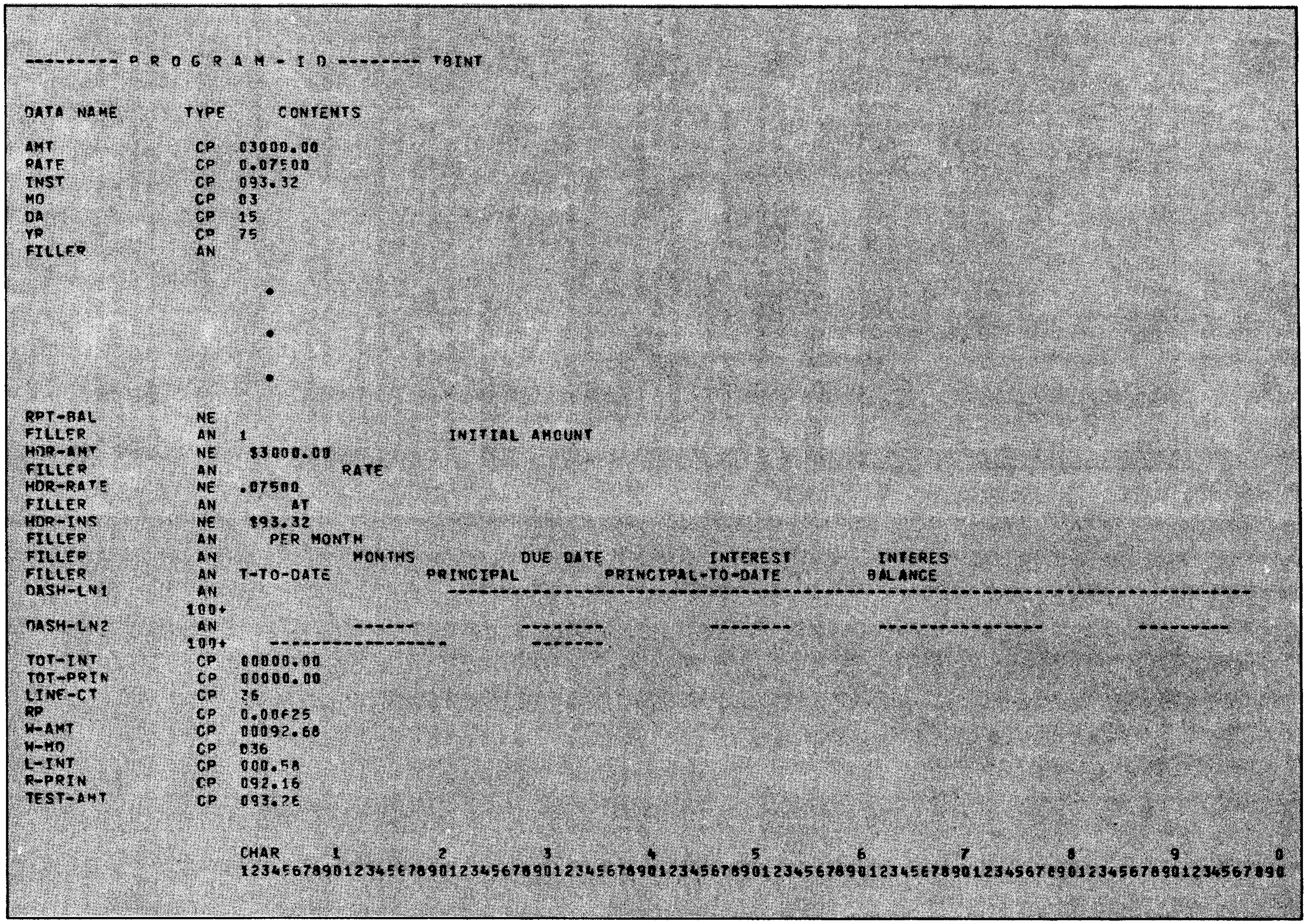
Debugging lines are compiled as executable code when either of the following is true:

The DEBUGGING MODE clause appears in the SOURCE-COMPUTER paragraph.

The DB=DL parameter appears on the compiler call.

In the absence of one of these compile-time options, all debugging lines are compiled as comments. Consequently, debugging lines should be coded so that they are syntactically correct as both comments and executable code.

Once the debugging lines are compiled, they execute each time the program executes; they cannot be suppressed at execution-time.



```
---------- P R O G R A M - I D ------- TBINT

DATA NAME      TYPE     CONTENTS

AMT             CP   03000.00
RATE            CP   0.07500
INST            CP   093.32
MO              CP   03
DA              CP   15
YR              CP   75
FILLER          AN

                        •

                        •

                        •

RPT-BAL         NE
FILLER          AN   1                   INITIAL AMOUNT
HDR-AMT         NE    $3000.00
FILLER          AN             RATE
HDR-RATE        NE   .07500
FILLER          AN        AT
HDR-INS         NE    $93.32
FILLER          AN      PER MONTH
FILLER          AN              MONTHS          DUE DATE          INTEREST        INTERES
FILLER          AN   T-TO-DATE         PRINCIPAL        PRINCIPAL-TO-DATE        BALANCE
DASH-LN1        AN                   ------------------------------------------------------------------------------
               100+
DASH-LN2        AN              ------          --------          --------        ----------------        ---------
               100+     -----------------        -------
TOT-INT         CP   00000.00
TOT-PRIN        CP   00000.00
LINE-CT         CP   36
RP              CP   0.00625
W-AMT           CP   00092.68
W-MO            CP   036
L-INT           CP   000.58
R-PRIN          CP   092.16
TEST-AMT        CP   093.26


                     CHAR     1         2         3         4         5         6         7         8         9         0
                     1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 10-2. Termination Dump Example

DEBUGGING SECTIONS

A debugging section is a section within the Declaratives portion of the Procedure Division that follows a USE FOR DEBUGGING declarative statement. The debugging section is compiled as executable code only when the DEBUGGING MODE clause (figure 10-3) appears in the SOURCE-COMPUTER paragraph. In the absence of the DEBUGGING MODE clause, debugging sections are compiled as comments.

```
WITH DEBUGGING MODE
```

Figure 10-3. DEBUGGING MODE Clause Format

Debugging Section Execution

Execution of the debugging section is controlled by switch 6, one of the six external software switches that are available to every job.

Switch-Status	Meaning for Debugging
ON	Debugging section code executes.
OFF	Debugging section code does not execute.

Switch 6 can be turned on internally or externally. Internally, it is turned on by specifying the SWITCH-6 IS mnemonic-name clause in the SPECIAL-NAMES paragraph and setting the mnemonic-name to ON in a format 4 SET statement.

Externally, switch 6 is turned on by a control statement executed prior to program execution. The control statement can be entered as part of an interactive or batch job. Its format is:

```
SWITCH,6.
```

The control statement acts as a toggle: the first occurrence of the statement turns the switch on; the second turns it off; the third turns it on again.

USE FOR DEBUGGING Declarative Statement

The USE FOR DEBUGGING statement (figure 10-4) identifies the user items that are to be monitored by the associated debugging section. It must be the first sentence after a section header within the Declaratives portion of the Procedure Division. Statements within the section form the debugging section that executes when the conditions established by the USE declarative occur.

The USE FOR DEBUGGING statement can reference one or several of the following items: identifier, procedure-name, file-name, or cd-name. Any particular identifier, procedure-name, file-name, or cd-name can be specified only once in a USE FOR DEBUGGING statement and cannot be duplicated in any other USE FOR DEBUGGING statement.

Reference modification is not allowed on the identifiers in a USE FOR DEBUGGING statement.

The debugging section is not executed as a result of execution of a statement in which file-name, identifier, or procedure-name is used as a qualifier.

```
USE FOR DEBUGGING ON

     ( [ALL REFERENCES OF] identifier-1 )
     { procedure-name-1                 }
     { file-name-1                      }
     { cd-name-1†                       }
     ( ALL PROCEDURES                   )

     [ [ALL REFERENCES OF] identifier-2 ]
     [ procedure-name-2                 ]
     [ file-name-2                      ]  . . . .
     [ cd-name-2†                       ]
     [ ALL PROCEDURES                   ]

†Can only be used under NOS.
```

Figure 10-4. USE FOR DEBUGGING Statement Format

ALL REFERENCES OF Phrase

The ALL REFERENCES OF phrase specifies that the debugging section is to execute each time a statement that explicitly references identifier-1 is executed.

Identifier-1 names the data item to be monitored. It must not name a data item defined in the Report Section other than sum counter.

If identifer-1 names an item that contains an OCCURS clause or is subordinate to an OCCURS clause, the subscripting or indexing normally required must be omitted.

When the ALL REFERENCES OF phrase is specified, the debugging section is executed immediately after the execution of a statement that explicitly references identifier-1, except under the following conditions:

In a RELEASE, REWRITE, or WRITE statement, the debugging section is executed prior to execution of the statement. If the FROM phrase is specified, the debugging section is also executed after any resulting move operation is completed.

In a GO TO statement with the DEPENDING ON phrase, the debugging section is executed before control is transferred by the GO TO statement and prior to the execution of any debugging sections associated with the procedure-name to which control is transferred.

In a PERFORM statement with either the VARYING, AFTER, or UNTIL phrase, the debugging section is executed immediately after each initialization, modification, or evaluation of the contents of identifier-1.

When identifier-1 is specified without the ALL REFERENCES OF phrase, the debugging section is executed for the RELEASE, REWRITE, WRITE, and PERFORM statements as indicated for the ALL REFERENCES OF phrase. For all other references to identifier-1, the debugging section is executed after execution of a statement that contains an explicit reference to identifier-1 and that updates the contents of identifier-1. (The contents of identifier-1 need not change.)

Procedure-name Phrase

If procedure-name-1 is specified, the debugging section is executed immediately before each execution of procedure-name-1 and immediately after the execution of any ALTER statement that references procedure-name-1.

Procedure-name-1 cannot be specified in a program for which an ALL PROCEDURES phrase is specified.

File-name Phrase

If file-name-1 is specified, the debugging section is executed as follows:

> After the execution of either an OPEN, CLOSE, DELETE, or START statement that references file-name-1.
>
> After the execution of a RETURN statement (after any other specified USE procedure) not resulting in the execution of an associated AT END imperative-statement.
>
> After the execution of a READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement.

Cd-name Phrase

If cd-name is specified, the debugging section is executed after the execution of any of the following statements, if these statements reference the specified cd-name: ACCEPT MESSAGE COUNT, ENABLE, DISABLE, RECEIVE (without the NO DATA imperative-statement), PURGE, or SEND statement.

ALL PROCEDURES Phrase

The ALL PROCEDURES phrase can appear only once in a program. When specified, the debugging section is executed immediately before execution of each procedure (except those appearing within a debugging section) and immediately after execution of each ALTER statement that references a procedure.

DEBUG-ITEM REGISTER

The special register DEBUG-ITEM is updated with each execution of a debugging section to provide information concerning the condition that caused execution of the debugging section. Each of the fields in the register can be accessed individually.

The implicit description of DEBUG-ITEM is shown in figure 10-5.

DEBUG-ITEM is space-filled and the following subordinate items updated as indicated prior to each execution of the debugging section.

DEBUG-LINE	Line number of the source statement.
DEBUG-NAME	First 30 characters of the name that caused execution of the debugging section.
DEBUG-SUB-1	The occurrence number of the first level of subscripting or indexing, if applicable.
DEBUG-SUB-2	The occurrence number of the second level of subscripting or indexing, if applicable.
DEBUG-SUB-3	The occurrence number of the third level of subscripting or indexing, if applicable.
DEBUG-CONTENTS	The contents of the data or procedure item being monitored. DEBUG-CONTENTS size is as large as necessary to hold the largest item monitored.

Updating of the subordinate entries in DEBUG-ITEM, with the exception of DEBUG-CONTENTS, is accomplished in accordance with the rules for the MOVE statement.

Table 10-2 shows the contents of DEBUG-ITEM when the debugging section is executed under various conditions. The user can write the DEBUG-ITEM to a print file in order to review the contents of DEBUG-ITEM.

```
01   DEBUG-ITEM.
     02   DEBUG-LINE PICTURE IS X(6).
     02   FILLER PICTURE IS X VALUE SPACE.
     02   DEBUG-NAME PICTURE IS X(30).
     02   FILLER PICTURE IS X VALUE SPACE.
     02   DEBUG-SUB-1 PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
     02   FILLER PICTURE IS X VALUE SPACE.
     02   DEBUG-SUB-2 PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
     02   FILLER PICTURE IS X VALUE SPACE.
     02   DEBUG-SUB-3 PICTURE IS S9999 SIGN IS LEADING SEPARATE CHARACTER.
     02   FILLER PICTURE IS X VALUE SPACE.
     02   DEBUG-CONTENTS PICTURE IS X(n).
     02   DEBUG-NUMERIC-CONTENTS REDEFINES DEBUG-CONTENTS PICTURE IS S9(n)V9(n)
             SIGN IS LEADING SEPARATE CHARACTER.
```

Figure 10-5. DEBUG-ITEM Description

TABLE 10-2. CONTENTS OF DEBUG-ITEM

Reference Causing Execution of the Debugging Section	Contents of DEBUG-ITEM		
	DEBUG-LINE	DEBUG-NAME	DEBUG-CONTENTS
First execution of the first nondeclarative procedure in the program	Line number of the first statement of the procedure	procedure-name	START PROGRAM
Reference to procedure-name-1 in an ALTER statement	Line number of ALTER statement	procedure-name-1	procedure-name specified in the TO phrase of an ALTER statement
Reference to procedure-name-1 in a GO TO statement	Line number of GO TO statement	procedure-name-1	
Reference to procedure-name-1 in a SORT or MERGE phrase	Line number of the SORT or MERGE statement	procedure-name-1	SORT INPUT, SORT OUTPUT, or MERGE OUTPUT, depending on whether the reference occurred in SORT input phrase, SORT output phrase or MERGE output phrase, respectively.
Reference to procedure-name-1 in a PERFORM statement	Line number of the PERFORM statement	procedure-name-1	PERFORM LOOP
Reference to procedure-name-1 in a USE statement	Line number of the USE statement	procedure-name-1	USE PROCEDURE
Control transferred to procedure-name-1 implicitly from the previous paragraph	Line number of the previous statement	procedure-name-1	FALL THROUGH
Reference to file-name-1	Line number of statement that references file-name-1	file-name-1	For a READ or RETURN statement, the entire record; otherwise, spaces.
Reference to identifier-1	Line number of the statement that references identifier-1	identifier-1	Contents of identifier-1

PRODUCT INTERFACES

Memory for COBOL 5 programs is managed by the Common Memory Manager (CMM). All COBOL 5 file input-output requests are implemented through CYBER Record Manager (CRM). COBOL 5 also interfaces with the Sort/Merge facility, FORTRAN programs or subroutines, the COMPASS assembly language, and the Message Control System (MCS). This section provides an overview of COBOL 5 interfaces with these products. For further detail, refer to the reference manuals for these products. The CYBER Database Control System (CDCS) interface with COBOL 5 is discussed separately in section 14, Sub-Schema Facility.

CMM INTERFACE

COBOL programs can explicitly interface with CMM in order to dynamically assign blocks of memory and move data to and from these assigned blocks. The COBOL-callable routine C.CMMMV can be used to move data to and from CMM blocks; this routine is documented in section 15.

The calls necessary to interface with CMM are listed in the CMM reference manual. In general, the only CMM routines that are useful to call from a COBOL program are the routines CMM.ALF (to allocate blocks) and CMM.FRF (to free the same blocks). A COBOL program illustrating the use of these routines is given in the CMM reference manual.

CRM INTERFACE

CYBER Record Manager is used during execution for all file input and output. The compiler generates information for the file information table (FIT) according to the statements of the source program. The FIT is created and initialized when a file is opened during execution.

The programmer can affect the information in the FIT in three ways:

By the clauses or phrases used in the source program.

By the USE literal clause of the FILE-CONTROL paragraph.

By FILE control statement execution prior to program execution.

FIT FIELDS FROM SOURCE CODE

The primary FIT fields established by source code statements are concerned with the file name and structure. The implementor-name of the SELECT clause establishes the logical file name in the FIT.

For all organizations the following fields are set:

Field and Symbolic Value	Clause or Statement Used
FO	ORGANIZATION Clause
SQ	SEQUENTIAL
WA	RELATIVE
IS	INDEXED
DA	DIRECT
AK	ACTUAL-KEY
WA	WORD-ADDRESS
RT, MNR, MRL	RECORD clause, Record Description entry, WRITE with ADVANCING phrase, FD entry LINAGE clause (see table 4-2, section 4).
RL	See table 4-2, section 4
Other record description fields	Record Description entry
PD	OPEN statement
INPUT	INPUT
OUTPUT	OUTPUT
I-O	I-O
LT	LABEL RECORDS clause
UL	OMITTED
S	STANDARD
ULP	Performed by system if VALUE OF phrase in LABEL RECORDS clause

User exits for DX, EX, LX are set such that control returns to the system execution-time routines. These routines then provide the capabilities described elsewhere in this manual.

For sequential files, the following FIT fields also are set:

FF	Always set
BT, MNB, MBL	BLOCK CONTAINS clause
BFS	RESERVE clause
MFN, PNO	MULTIPLE FILE TAPE clause

For indexed files, these FIT fields also are set. Fields set to 0 include DKI, BCK, and FWI.

IP	Installation parameter
DP	Installation parameter
KT, KA, KP, KL	RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
IBL	Installation parameter

MBL	BLOCK CONTAINS clause
XN	SELECT clause second implementor-name
CDT/DCT	Program collating sequence
ON	OPEN statement
NEW	OUTPUT
OLD	Other

For direct files, these FIT fields also are set. Fields set to 0 include BCK and FWI.

RKP, RKW, KA, KL	RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
HMB	BLOCK COUNT clause
MBL	BLOCK CONTAINS clause
HRL	USE FOR HASHING declarative
XN	SELECT clause second implementor-name
ON	OPEN statement
NEW	OUTPUT
OLD	Other

For actual-key files, the following FIT fields also are set. Fields set to 0 include BCK and FWI. HB is set to 1.

KA, KL	RECORD KEY clause, ALTERNATE RECORD KEY clause, and item description
MBL	BLOCK CONTAINS clause
RB	BLOCK CONTAINS clause and Record Description entry
ON	OPEN statement
NEW	OUTPUT
OLD	Other

For word-address files, the following FIT field also is set:

BFS	RESERVE clause

For relative files, the following FIT field also is set:

BFS	RESERVE clause

FIT FIELDS FROM USE CLAUSE

The USE clause of the FILE-CONTROL paragraph of the Input-Output Section of the Environment Division also affects the FIT. Values provided by this clause are compiled into the program, overriding values that would otherwise result from source program lines.

See the USE clause discussion for a list of the FIT fields that can be referenced in the clause.

FIT FIELDS FROM FILE CONTROL STATEMENT

The FILE control statement parameters become part of the FIT at the time the referenced file is opened during execution. Any parameter can be used; however, parameters should not be set at execution time without a full awareness of COBOL assumptions and CYBER Record Manager execution-time processing.

In general, parameters on a FILE control statement should be confined to those permitted by the USE clause. In particular, the following fields cannot be used or will have no effect:

EX	KL	LBL	MKL
FO	KP	LFN	ULP
HRL	KT	LT	WSA
KA	LA	LX	

A FILE control statement that specifies file equivalence cannot be used with COBOL. For example, the control statement FILE,LFN1=LFN2 is not allowed. If this facility is desired, equivalence can be specified on the execution call control statement, for example, LGO,LFN1=LFN2.

See the CYBER Record Manager reference manuals for a description of the FILE control statement.

SORT/MERGE INTERFACE

The Sort/Merge facility of the COBOL 5 language is similar to the capabilities provided by Sort/Merge 4, which operates under the NOS and NOS/BE operating systems.

Two of the collating sequences that can be specified directly in a COBOL program are the same as the collating sequences available through the Sort/Merge product.

COBOL Identification	Sort/Merge Identification
ASCII-64	ASCII6
CDC-64	COBOL6

Both the product and the language facility allow a user-specified collating sequence to be defined.

The Sort/Merge product uses Basic Access Methods (BAM) for execution-time file input and output. COBOL 5 also uses BAM, as well as Advanced Access Methods (AAM).

FORTRAN INTERFACE

FORTRAN Extended or FORTRAN 5 subroutines can be executed from a COBOL 5 program by specifying FORTRAN-X and the name of the routine in the ENTER statement. Parameters can be passed to the subroutines through the USING phrase of the ENTER statement. The ENTER statement without the FORTRAN-X should not be used to call a FORTRAN Extended or FORTRAN 5 subroutine because extra information contained in the parameter list could cause the FORTRAN compiler to abort or to process the parameters incorrectly. If FORTRAN Extended subroutines contain any PRINT statements, the call to the FORTRAN Extended compiler must contain the SYSEDIT parameter (FTN,SYSEDIT.). The COBOL 5 program must execute a DISPLAY statement or an OPEN statement specifying a file with the implementor-name "OUTPUT" before the subroutine is called.

For best results, the parameters specified in the USING phrase of the ENTER statement should be either COMPUTATIONAL-1 or COMPUTATIONAL-2 items, or level 01 items that are a multiple of 10 characters in length. COMPUTATIONAL-1 items correspond to FORTRAN integer items, and COMPUTATIONAL-2 items correspond to real items. Items of any other usage are treated as character strings. If character strings are being passed to FORTRAN 5 programs, the FORTRAN 5 character items should be specified as type CHARACTER. Returning characters other than type characters from FORTRAN 5 to COBOL could cause an error. A discussion of the internal storage of these items is contained in the paragraphs under the heading Numeric Data Format in this section.

COBOL 5 subprograms can be called by FORTRAN main programs. The first COBOL subprogram to be called must be compiled with the MSB parameter in the COBOL5 control statement (COBOL5,MSB.). All other subprograms can be compiled with the SB parameter or can be stacked with the main subprogram. If a FORTRAN main program calls a COBOL subprogram, the subprogram cannot contain any DISPLAY statements or any OPEN statements specifying a file with the implementor-name "OUTPUT".

Mixed COBOL and FORTRAN input-output operations function only in the file OUTPUT. For any other case, successful execution is not guaranteed.

NUMERIC DATA FORMAT

Numeric data is stored according to the program description of the data and options selected on the compiler call.

COMPUTATIONAL items are stored as display code characters. If the CC1 parameter appears on the compiler call, however, COMPUTATIONAL items are stored as if they were COMPUTATIONAL-1 items.

COMPUTATIONAL-1 items are stored as binary integers. The high order bits contain the sign of the number, and the remaining bits contain the value. The sign is represented as all zeros (positive value) or all ones (negative value). COMPUTATIONAL-1 values cannot exceed 48 bits, which is a value of 281 474 976 710 655. This format is the same as that for FORTRAN Extended integer variables with the exception that FORTRAN will allow bits 48 through 59 to contain other than all zeros or all ones. If they are not all zeros or all ones in COBOL, the results of COBOL operations on the items are unpredictable. COMPUTATIONAL-1 items for COBOL 4 and COBOL 5 are not the same. COBOL 4 COMP-1 items contain a floating point bias in bits 48 to 59.

COMPUTATIONAL-2 items are stored as floating point single precision numbers. The format is the same as the real format for FORTRAN Extended. The high order bits indicate the exponent and sign of the number; the low order bits are the normalized coefficient. See the computer system reference manual or the FORTRAN Extended reference manual for more information about this format.

COMPUTATIONAL-4 items are stored as signed binary integers in 6, 12, 18, 24, 30, 36, 42, or 48 bits. The high order bit contains the sign. A COMPUTATIONAL-4 item can appear anywhere in a word on a character boundary; no word alignment is assumed.

INDEX items are stored in a format that indicates the occurrence number and the location of the item from the beginning of the table.

Figure 11-1 shows the format of numeric items.

CALL/ENTER STATEMENT CODE

The CALL statement and the ENTER statement generate COMPASS assembly language code that passes the locations of data items to be used in the subprogram being called. The code passes an address to a list of pointers to the data items. The return jump instruction specifies the routine-name in the CALL or ENTER statement. The VFD instruction stores the line number of the CALL or ENTER statement within the source program in the lower half of the word containing the return jump instruction.

```
   SA1     parameter list address
 + RJ      routine-name
 - VFD     30/line number
```

The parameter list contains a full word of information about each parameter in the USING phrase of the CALL or ENTER statement. The list is terminated by a word of binary zeros. The format of each word in the list is affected by the type of parameter being passed, as shown in figure 11-2.

NOTE

Because of anticipated changes in this product, the use of COMPASS is not recommended. For guidelines, see appendix F.

COBOL COMMUNICATION FACILITY

The COBOL Communication Facility (CCF) is used with the Message Control System (MCS) to pass messages between on-line COBOL programs and user terminals or other COBOL programs. MCS brings teleprocessing capabilities directly into the standard syntax of the COBOL language. It provides COBOL 5 with a generalized method of queuing, routing, and journaling messages in the CYBER Network under the NOS operating system.

MCS uses the Network Access Method (NAM) to communicate with the terminals. The Application Definition Language (ADL) is an MCS feature that is used to define an application to MCS. (See the Message Control System reference manual.)

COBOL programs can be initiated through the normal method of job submission, or by preparing files containing the desired job stream and having MCS submit the job when a specified condition is met (for example, when messages in an input queue are ready to be processed). Programs can also be initiated by other application programs. MCS can serve multiple applications and allows a given program to communicate with multiple terminals or multiple programs to communicate with a single terminal.

The Data Division of a COBOL program contains the Communication Section that contains input buffer and output buffer areas used to interface with MCS. These areas can be referenced by six statements (ACCEPT MESSAGE COUNT, SEND, RECEIVE, PURGE, DISABLE, and ENABLE) in the Procedure Division.

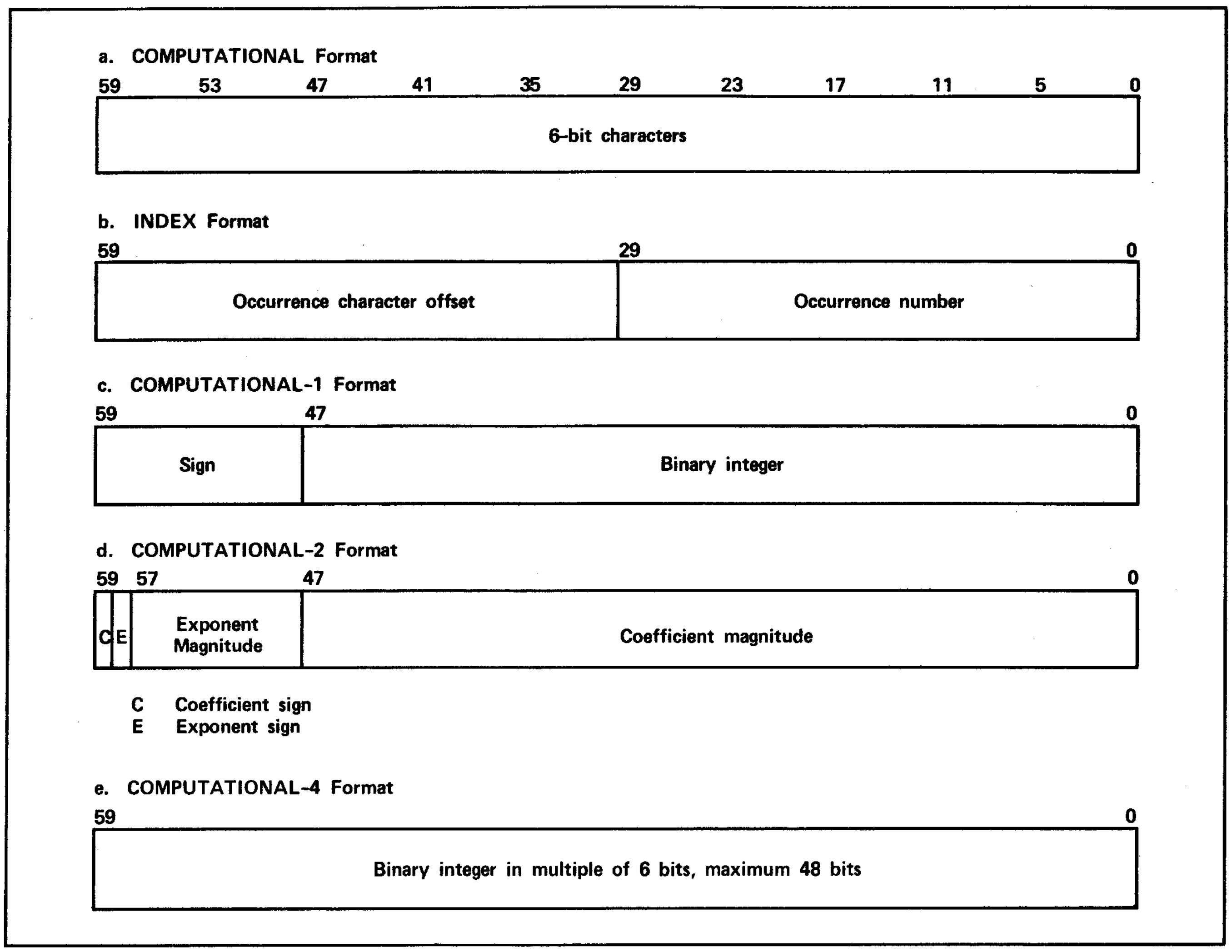


Figure 11-1. Numeric Data Format

Messages are stored by MCS in logical data structures called queues. These message queues allow programs and terminals to communicate; the messages are buffered through the queue before being delivered to specified destinations. The input queues have four levels called queue, sub-queue-1, sub-queue-2, and sub-queue-3. (See Symbolic Queues in the Communication Section.)

The messages or segments of messages are determined with end-of-message indicators. The end of a message on output is determined by an indicator from the program that sends the message. A message on input is determined by a combination of terminal operator action and MCS convention.

The three end-of-message indicators are end-of-segment indicator (ESI), end-of-message indicator (EMI), and end-of-group indicator (EGI). MCS does not forward a message that is not a complete message or does not have an EMI or EGI. Message segments are stored in an input queue and made available to the COBOL program when the entire message has been stored.

Messages or message segments are acquired from a specified queue using the RECEIVE statement in the Procedure Division of the COBOL program. Messages or message segments are released to a specified queue by using the SEND statement. The logical paths or connections between output queues and destinations or between input queues and sources are established with the ENABLE statement. These logical paths are disconnected with the DISABLE statement.

TRANSACTION FACILITY

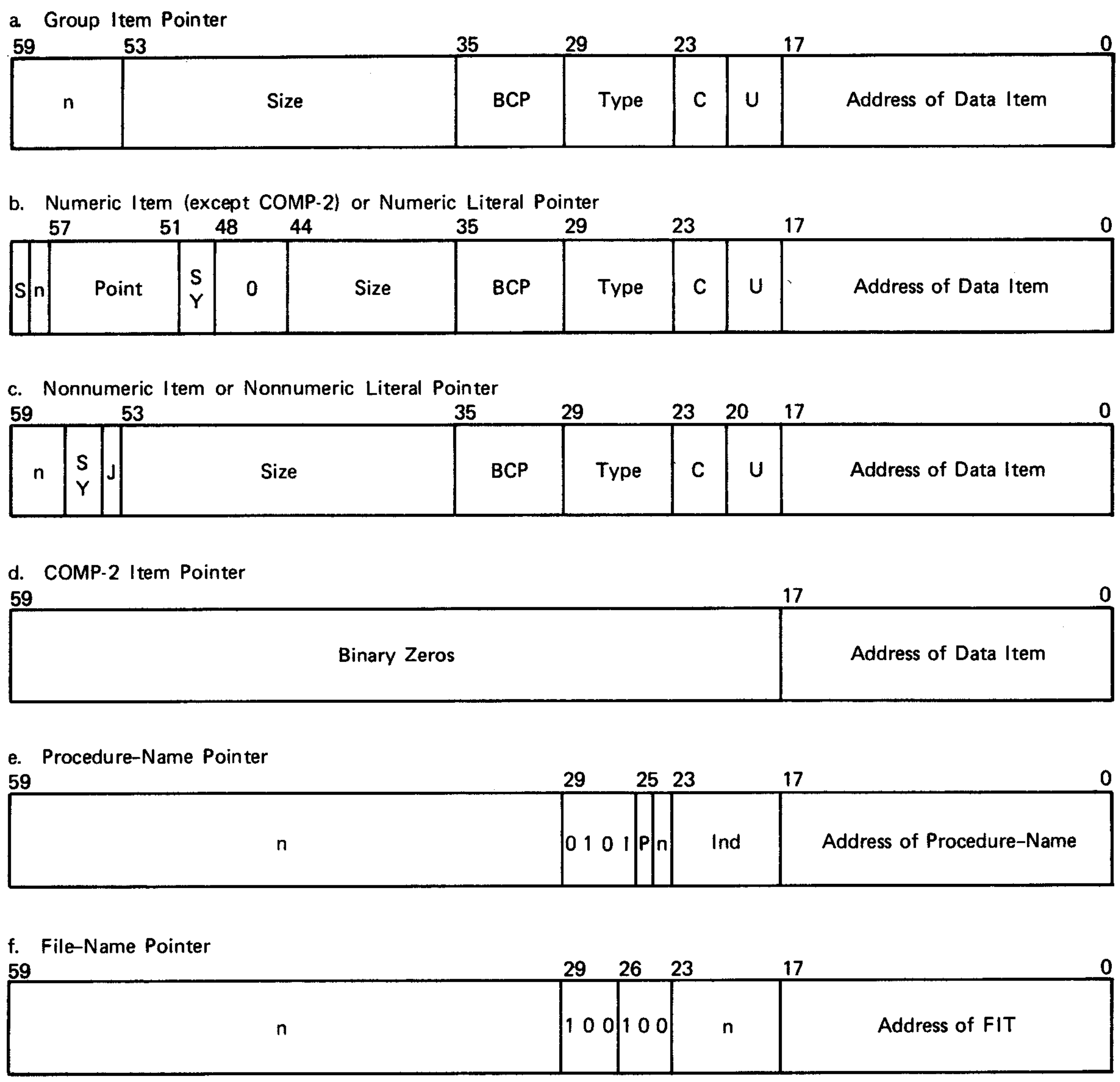
The Transaction Facility (TAF) is supported by CDCS. TAF allows transaction processing (a high-speed handling or repetitive execution of a relatively small number of jobs called tasks). The task can be executed by many different people from many locations. A task usually performs one of the following manipulations on a data base:

Stores a new record

Alters or deletes an existing record

Produces formatted output

Once a task is initiated, it is executed through TAF and CDCS. The task can communicate through TAF and the Network Access Method (NAM) and can initiate subsequent tasks as required by terminal input or data base information. For further details and information regarding other TAF interfaces with COBOL, see the Transaction Facility reference manual.


a. Group Item Pointer
59 53 35 29 23 17 0
n | Size | BCP | Type | C | U | Address of Data Item
b. Numeric Item (except COMP-2) or Numeric Literal Pointer
57 51 48 44 35 29 23 17 0
S | n | Point | SY | 0 | Size | BCP | Type | C | U | Address of Data Item
c. Nonnumeric Item or Nonnumeric Literal Pointer
59 53 35 29 23 20 17 0
n | SY | J | Size | BCP | Type | C | U | Address of Data Item
d. COMP-2 Item Pointer
59 17 0
Binary Zeros | Address of Data Item
e. Procedure-Name Pointer
59 29 25 23 17 0
n | 0 1 0 1 | P | n | Ind | Address of Procedure-Name
f. File-Name Pointer
59 29 26 23 17 0
n | 1 0 0 | 1 0 0 | n | Address of FIT

Field	Bits	Significance
Address	0-17	Address of item or FIT
U	18-20	USAGE of data item (octal) 0 Not specified 1 DISPLAY (nonnumeric) 2 COMPUTATIONAL OR DISPLAY (numeric) 4 COMPUTATIONAL-1 or INDEX 5 COMP-4 7 Mixed (group)
C	21-23	Class of data-name item (octal) 0 Not specified 1 Alphabetic 2 Numeric 3 Alphanumeric 4 Boolean 7 Mixed (group)
Type	24-29	Type of data-name item (octal) 4x Group item with x = 4 44 File-name 5x Elementary item with any x value 24 Paragraph-name 26 Section-name
BCP	30-35	Byte number in address at which item begins. Bytes are each 6 bits and are numbered 0 through 9 left to right.
Size	36-44 or 36-53	Number of characters in item

Field	Bits	Significance
SY	49-50 or 55-56	Synchronization indicator 00 Not synchronized 10 Synchronized left 11 Synchronized right
Point	51-57	Point position (bits 56-57) 00 Assumed point left 01 Assumed point right 10 Actual point left 11 Actual point right Point location (bits 51-55)
S	59	Sign 1 Signed 0 Unsigned
J	54	Justification indicator 0 Justified left 1 Justified right
Ind	18-23	Indirect procedure indicator 0 Procedure is in memory ≠ 0 Procedure is not in memory. Address points to an index word used by the compiler for overlays.
P	25	Section indicator 0 Paragraph procedure-name 1 Section procedure-name
n	—	Field not used, zero filled

Figure 11-2. Pointer Word Format for CALL and ENTER Statements

COMPILER CALL AND EXECUTION 12

Compilation and execution of a COBOL 5 program are initiated by control statements that conform to operating system syntax. The COBOL 5 compiler is called with the COBOL5 statement. If the program is compiled and executed in the same run unit, the execution control statement LGO is the default file name that contains the binary object program and is used to load and execute the program. If the program is compiled and executed in separate run units, the program can be executed with the LOAD and EXECUTE statements.

Optional parameters can be specified on both the compiler call statement and the execution call statement. Parameters can appear in any order and are separated by commas. The control statements cannot be continued beyond one line. Control statements can be entered interactively through a terminal or entered through a card deck.

During source program compilation, the compiler generates various diagnostic messages. The user can control the type of messages listed, the location of the messages, and the system response to the messages through parameters on the compiler call statement. Refer to the COBOL 5 diagnostic handbook for a complete list of compilation diagnostics. Execution diagnostics are listed in appendix B.

COMPILER CALL

The COBOL 5 compiler is called with a control statement that conforms to the operating system syntax. The control statement cannot be continued beyond one line. The compiler call can be specified in one of the following formats in which p indicates a parameter:

```
COBOL5.

COBOL5(p1[,p2] ...)

COBOL5,p1[,p2] ... .
```

More than one program can be compiled by a single call to the compiler. The programs should follow each other without separation when several programs are to be compiled from one source file. The compiler recognizes the Identification Division header as the start of a subsequent program, and terminates the program being processed before beginning compilation of the next program. The first program is compiled as a main program, a subprogram, or a main subprogram, depending on the SB and MSB parameters (described later in this section). The second and all subsequent programs are compiled as subprograms (as if SB were specified on the COBOL5 statement).

During compilation the compiler automatically increases the field length for the job as more space is needed. The compiler can be constrained to operate in a fixed field length by specifying the CM parameter on the job statement. This might result in significantly increased compilation time, however. The compiler cannot be constrained to operate in less than the field length needed for a trivial program plus 2000_8.

PARAMETERS

Specification of any parameter is optional. Any default that results from the omission of a parameter could be changed by an installation.

If no parameters are specified, all default parameters are selected, and the compiler call appears as:

```
COBOL5.
```

A variety of compilation options can be specified in a parameter list following the compiler call name. If the name of the source input file is NEWPR, for example, the compiler call appears as:

```
COBOL5,I=NEWPR.
```

If a parameter list is specified, it must conform to the syntax for job control statements as defined in the operating system reference manual, with the added restriction that a comma is the only valid parameter separator.

PARAMETER OPTIONS

The parameter options are described in the following paragraphs. They are discussed in alaphabetical order of the abbreviation that specifies the parameter.

ANSI Extensions Control

The ANSI parameter specifies whether use of non-ANSI extensions to COBOL as described in the American National Standard X3.23-1974 are to be diagnosed and, if diagnosed, which severity level is diagnosed. Multiple options for the ANSI parameter can be selected by separating the options with a slash. The valid options are:

omitted	Extensions to the American National Standard X3.23-1974 are accepted by the compiler without diagnostics, numeric display items are edited by the DISPLAY statement, and level 77 items are stored SYNC RIGHT. If non-ANSI reserved words are used as user-defined words, diagnostics result.
ANSI	Equivalent to ANSI=T.
ANSI=T	Extensions to the ANSI standard are diagnosed and treated as errors with severity T. The EL=T parameter must also be specified to obtain a diagnostic listing of non-ANSI statements.
ANSI=F	Extensions to the ANSI standard are diagnosed and treated as errors with severity F. ANSI=T and ANSI=F are mutually exclusive.

ANSI=NOEDIT	Numeric display items are not edited by the DISPLAY statement. This parameter requests strict ANSI interpretation. Items that contain embedded decimal points or overpunched signs are displayed without editing.
ANSI=77LEFT	This parameter must be specified if a level 77 item is specified in the USING phrase of a CALL statement and the corresponding item is defined as a level 01 item in the LINKAGE SECTION of the called program. This parameter must also be specified if a level 01 item is specified in the USING phrase of a CALL statement and the corresponding item is defined as a level 77 item in the LINKAGE SECTION of the called program. This parameter causes level 77 items to be stored SYNC LEFT. If ANSI=77LEFT is not specified, level 77 items are stored SYNC RIGHT. If a main program is compiled using the 77LEFT option, then all subprograms should be compiled using the 77LEFT option also to avoid conflicts between two level 77 items.
ANSI=AUDIT	This parameter selects the parameters ANSI=NOEDIT and ANSI= 77LEFT. The parameters ANSI=T or ANSI=F are not compatible with this parameter. When this parameter is used, non-ANSI reserved words are not recognized as reserved words and, therefore, can be user-defined words. If a group item containing a variable-occurrence data item is used as a receiving item, only that part of the table area specified by the value of the DEPENDING ON data item is affected. The collating sequence for an indexed file is the native sequence. If ANSI=AUDIT is not used, the collating sequence of the program (instead of the native sequence) is used for an indexed file.

APO Apostrophe Character

The APO parameter controls the character that can be used as the delimiter of nonnumeric literals in the source program. The valid options are:

omitted	Nonnumeric literals in the source program are delimited by the quotation mark character, which has a display code value of 64.
APO	The ASCII apostrophe character with a display code value of 70 (Hollerith 11-8-5 punch, sometimes punched by an up arrow key) delimits nonnumeric literals in the source program instead of the quotation mark character of display code value 64 (Hollerith 8-4 punch, sometimes punched by a not equal sign). This option reverses the roles of ' and " such that " can be used within an alphanumeric literal the same as any other character. Within a source program, this option can be selected by the QUOTE IS APOSTROPHE clause of the SPECIAL-NAMES paragraph.

B Binary Output

The B parameter specifies the name of the file to which binary output from compilation is written. The valid options are:

omitted	Binary output from compilation is written to file LGO.
B	Binary output from compilation is written to file BIN.
B=0	No binary output is produced during compilation.
B=lfn	Binary output from compilation is written to file lfn, with lfn being one through seven letters or digits beginning with a letter.

BL Burstable Listing

The BL parameter controls page ejects in the listing produced by the compiler. The valid options are:

omitted	Triple space separates the program listing, diagnostics, cross reference listing, and any cross reference map.
BL	Page eject occurs between various parts of the listing.

CC1 COMP Equivalence to COMP-1

The CC1 parameter affects the storage and processing of data items described as COMPUTATIONAL (COMP) items. The parameter is selected when CC1 is specified. Selection of this parameter allows programs written for other compilers to gain the efficiencies of COMPUTATIONAL-1 (COMP-1) processing. The valid options are:

omitted	Data items described as COMP are stored and processed as COMP items.
CC1	Data items described as COMP are stored and processed as COMP-1 items.

D Data Base Sub-Schema File Identification

The D parameter specifies the name of the file on which the sub-schema specified in the SUB-SCHEMA clause resides. Specification of this parameter is valid only for a COBOL program that is to use the CYBER Database Control System (CDCS). The valid options are:

omitted	Sub-schema for CDCS is not used; if the SUB-SCHEMA clause appears in the program, it generates a fatal compilation error.

D=0	Sub-schema for CDCS is not used; if the SUB-SCHEMA clause appears in the program, it generates a fatal compilation error.
D	Sub-schema for the CDCS interface resides on a file with the same name as that of the sub-schema.
D=lfn	Sub-schema for the CDCS interface resides on file lfn, where lfn is one through seven letters or digits beginning with a letter.

DB Debugging Selected

The DB parameter selects debugging options. Multiple options can be specified by separating the options with a slash. The valid options are:

omitted	None of the debugging options applicable to this parameter are selected.
DB=0	None of the debugging opions applicable to this parameter are selected.
DB	Equivalent to DB=DL/SB/B.
DB=B	Binary executable code is produced regardless of all errors in the source program. Lines with errors of severity C or F result in compilation of a call to an execution-time abort routine; execution of those lines aborts the program. If DB=B is not selected, the first occurrence of a C or F error inhibits generation of executable code.
DB=DL	Debugging lines in the source program (lines with a D in column 7) are compiled as executable code. If DB=DL is not selected, all debugging lines are treated as comment lines, unless the DEBUGGING MODE clause appears in the program. The presence of the DEBUGGING MODE clause causes the DL option to be ignored.
DB=RF	Code is compiled such that reference modification values are checked during execution to ensure that the values are within the bounds of the item being reference modified. An invalid value aborts the program with a dayfile message that identifies the line with the incorrect value. If DB=RF is not specified, reference modification values are not checked during execution.
DB=SB	Code is compiled such that subscript and index references are checked during execution to ensure that all references to tables are within the table bounds. An out-of-bounds reference aborts the program with a dayfile message that identifies the line with the incorrect reference. If DB=SB is not selected, subscripted and indexed references are not checked during execution.
DB=TR	Paragraph trace occurs during execution. If DB=TR is not selected, paragraph trace does not occur. If TDF and DB=TR are both specified, the termination dump indicates the last line and paragraph executed. If DB=TR is not specified with TDF, the last input-output statement executed is indicated.

E Error File Name

The E parameter specifies the name of the file to which error information is written. When the LO=S option is specified, information written to the file specified by the E parameter is also written to the file specified by the L parameter. If the lfn specified by the E parameter is the same as that specified by the L parameter, information is written only to the listing file. The valid options are for the E parameter are:

omitted	Error information specified by the EL parameter is written to the file OUTPUT.
E=0	Error information specified by the EL parameter is written to the file OUTPUT.
E	Error information specified by the EL parameter is written to the file ERR.
E=lfn	Error informaton specified by the EL parameter is written to the file with the name lfn, where lfn is one through seven letters or digits beginning with a letter.

EL Error Level to be Reported

The EL parameter indicates the severity level of errors that are listed on the file specified by the E parameter. Specification of a particular level selects that level and all higher levels. The valid options are:

omitted	Errors of levels W, F, and C are listed.
EL	Equivalent to EL=F.
EL=T	Trivial errors, plus all errors of levels W, F, and C, are listed. Federal Information Processing Standard (FIPS) errors are also listed. Level T errors indicate a suspicious usage: although the syntax is correct, the usage is questionable. EL=T is required to obtain a listing of the messages reported as UNLISTED NON-ANSI ERRORS on the diagnostic summary.
EL=W	Warning errors, plus all errors of levels F and C, are listed. Level W errors indicate the syntax of the statement is incorrect and the compiler has made an assumption and continued compilation.

EL=F	Fatal errors, plus all level C errors, are listed. Level F errors prevent compilation of the statement. Unresolvable semantic errors and propagated errors caused by earlier level F errors are among the causes of level F errors.
EL=C	Catastrophic errors are listed. Level C errors are fatal to compilation of the current program. Compilation resumes at the Identification Division header of any program immediately following without an intervening file boundary.

ET Error Termination

The ET parameter specifies the action to be taken by the compiler when complilation has completed. If an error of the specified level or higher occurs, the job skips to the EXIT control statements. The valid options are:

omitted	The next control statement in the job stream is executed after termination, despite any errors diagnosed during compilation.
ET=opt	The compiler aborts if the executable code contains any errors of at least the T, W, F, or C severity indicated by opt. Levels are those indicated by the EL parameter. Level T or level W errors produce executable binary code. Level F and level C errors produce a binary program that causes the loader to inhibit loading, unless the B option of the DB parameter is specified. The job resumes after any EXIT(s) control statement in the job stream.

FDL Fast Dynamic Loader Processing

The FDL parameter specifies the name file that contains directives pertaining to subprogram loading and CDCS 1 data base usage. (See section 15, Inter-Program Communication.) The valid options are:

omitted	All subprograms must be resident at the same time. The CALL statement must specify a literal and the first seven characters must be unique in the run unit. A CDCS 1 sub-schema cannot be used by subprograms. If a termination dump is selected through the TDF parameter, a dump of only the main program is taken.
FDL	Equivalent to FDL=FDLFILE.
FDL=lfn	The FDL file resides on file lfn, where lfn is one through seven letters or digits beginning with a letter. A CDCS 1 sub-schema can be used by subprograms. If a termination dump is selected through the TDF parameter, all programs specified in the FDL file are included in the dump.

FIPS Level Diagnosis

The FIPS parameter specifies diagnostics for language features of the Federal Information Processing Standard (FIPS) and controls the severity level of errors diagnosed. The ANSI parameter and the EL=T parameter must be specified to obtain a listing of these diagnostics. The valid options are:

omitted	No diagnostics for Federal Information Processing Standard levels are issued.
FIPS	Equivalent to FIPS=4.
FIPS=n	Language features above the specified Federal Information Processing Standard level are diagnosed; n specifies level 1, 2, 3, or 4 (as specified in FIPS PUB 21-1).

I Input File Name

The I parameter specifies the name of the file that contains the program source. The valid options are:

omitted	Card images of program to be compiled reside on file INPUT.
I	Card images of program to be compiled reside on file COMPILE.
I=lfn	Card images of program to be compiled reside on file lfn, where lfn is one through seven letters or digits beginning with a letter.

L Listing File Name

The L parameter specifies the name of the file to which the compiler writes source listing and any other requested information except diagnostics (see the LO parameter). The valid options are:

omitted	Source listing, diagnostics, and information selected by the LO parameter are written to file OUTPUT.
L	Source listing, diagnostics, and information selected by the LO parameter are written to file LIST.
L=0	No listing is produced.
L=lfn	Source listing, diagnostics, and information selected by the LO parameter are written to file lfn, where lfn is one through seven letters or digits beginning with a letter.

LBZ Leading Blank Zero

The LBZ parameter affects the treatement of leading blanks in numeric fields. This parameter is selected when LBZ is specified. Selection of this parameter increases the size of the binary output and significantly slows execution time. The valid options are:

omitted	Numeric fields that contain blanks are in error.
LBZ	All leading blanks in numeric fields are treated as zeros in arithmetic statements and comparisons.

LO Listing Options

The LO parameter specifies the information that is to be listed on the compiler output listing file specified by the L parameter. Multiple options for the LO parameter can be selected by separating the options with a slash. The valid options are:

omitted	Equivalent to LO=S.
LO	Equivalent to LO=S/M/R.
LO=0	None of the information that can be selected by O, R, M, or S is listed.
LO=M	A map correlating program entities with attributes such as data class, size, and physical storage is listed. (Source program is also listed.) Data items defined in the Report Section do not appear in the listing.
LO=O	Generated object code with COMPASS mnemonics is listed. (Source program is also listed.)
LO=R	Cross reference of program entities is produced. Each program entity is listed with the location of its definition and all locations of its use. (Source program is also listed.)
LO=S	Source program is listed.
LO= -S	Source program is not listed.

MSB Main Subroutine Indicator

The MSB parameter should be specified on the compiler call for only the first COBOL program called in a group of independently compiled subprograms. The MSB parameter should be used only when the COBOL program is called by a program written in a language other than COBOL. It should not be used for a COBOL subprogram that is called by another COBOL program. The valid options are:

omitted	Normal program is compiled.
MSB	Program is compiled as a subroutine that includes COBOL initiation.

PD Print Density

The PD parameter specifies the print density for all printable output (namely, the files specified by the L and E parameters). The PD parameter is ignored for connected interactive terminal listings. Any option specified by this parameter must be supported by the printer on which the files are output. The valid options are:

omitted	Listing specified by L and E parameters uses that print density specified by IP.PD in IPTEXT.
PD	Equivalent to PD=8.
PD=3	Listing specified by L and E parameters is double spaced at six lines per inch.
PD=4	Listing specified by L and E parameters is double spaced at eight lines per inch.
PD=6	Listing specified by L and E parameters is single spaced at six lines per inch.
PD=8	Listing specified by L and E parameters is single spaced at eight lines per inch.

PS Page Size

The PS parameter specifies the number of lines to be included on a printed page of the output listing. The valid options are:

omitted	Number of lines on a printed output page is the density specified by ((PD parameter) multiplied by (IP.PS/IP.PD)), where IP.PS and IP.PD are two installation parameters.
PS=n	Number of lines on a printed output page is n. Three lines exist at the top and at the bottom of each page, in addition to n.

PSQ Program Sequence

The PSQ parameter specifies the source of the sequence numbers used for diagnostics. The valid options are:

omitted	Compiler-generated sequence numbers are used for all diagnostics; sequence numbers in columns 1 through 6 are not processed.
PSQ	Sequence numbers in columns 1 through 6 are used for all diagnostics.

PW Page Width

The PW parameter specifies the width of a line on output listings. The valid options are:

omitted	Lines of printed output are 132 characters in length.
PW	Lines of printed output are 72 characters in length.

PW=n	Lines of printed output are n characters in length. The compiler reformats listing lines to this length. Lines longer than the page width are continued on the next print line and are prefixed with >>>>.

SB Subcompile Indicator

If the SB parameter is specified, the source program is compiled as a subprogram. The valid options are:

omitted	Program is compiled as a main program.
SB	Program is compiled as a subprogram. If the main program is not written in COBOL, the MSB parameter must also be used for one of the subprograms.

SY Syntax Check

The SY parameter affects the generation of executable code. The parameter is selected when SY is specified. The valid options are:

omitted	Source program is compiled and executable code is generated.
SY	Source program is checked for syntax, but executable code is not generated. When SY is selected, compilation time is approximately half of that required when SY is omitted.

TAF Program

Specification of the TAF parameter indicates that the source program is to be executed through the NOS Transaction Facility (TAF). The valid options are:

omitted	Program is run in a non-TAF environment.
TAF	Program is run as a NOS Transaction Facility task. Calls to unnecessary routines are not generated and the field length used for the task is significantly reduced.

TDF Termination Dump Indicator

If the TDF parameter is specified, the contents of the execution time field length are saved on a system file upon normal or abnormal job termination. The termination dump analysis program uses this system file and the file specified by lfn to produce the formatted dump. To obtain the dump, the control statement C5TDMP must be executed after the COBOL run. See section 10, Debugging Aids for more information. The valid options for the TDF parameter are:

omitted	Termination dump is not taken from this compilation; no extra files are written.
TDF	Equivalent to TDF=TDFILE.
TDF=lfn	Termination dump is taken at a later time. Tables needed for the dump are written to file lfn, where lfn is one through seven letters or digits beginning with a letter.

U Update File Name

The U parameter specifies the name of the file to which an object code listing suitable for assembly by COMPASS is written. The valid options are:

omitted	COMPASS assembly language images are not produced.
U=0	COMPASS assembly language images are not produced.
U	Equivalent to U=COMPS.
U=lfn	COMPASS line images of the generated program are written to file lfn in a format acceptable for the Update utility, where lfn is one through seven letters or digits beginning with a letter. The first seven characters from the name in the PROGRAM-ID paragraph become the deck name on a *DECK image written as the first item on the file. The second image on the file is *IDENT with the same name as the deck name.

UC1 Unpack COMP-1 Items

The UC1 parameter enables the COBOL 5 program to process files created by a COBOL 4 program that contains COMP-1 items. The parameter is selected when UC1 is specified. The valid options are:

omitted	All COMP-1 items are processed in COMP-1 format.
UC1	All COMP-1 items are converted to integer format before they are processed. Conversion occurs through the use of an unpack instruction that removes the exponent. UC1 should be used only when files created by COBOL 4 are being processed under COBOL 5. COMP-1 items in COBOL 4 have a different format in COBOL 5. Larger and slower object programs result from this parameter.

X Copy Text File Name

The X parameter specifies the name of the library file from which text is copied when specified by a COPY statement. The valid options are:

omitted	Equivalent to X=OLDPL.
X=0	Equivalent to X=OLDPL.
X	Equivalent to X=NEWPL.

X=lfn	The Update random program library containing text for COPY statements is on file lfn, where lfn is one through seven letters or digits beginning with a letter.

COMPILATION LISTINGS

The LO parameter of the compiler call selects the listings that are produced during compilation. The listings appear on the standard job print file, OUTPUT, unless another file is selected by the L parameter on the compiler call.

Listings that can be selected by the LO parameter are:

S	Source listing
M	Map of data items and procedures
R	Cross-reference map
O	Object listing

Another listing that appears is the diagnostic summary. The LIST/NOLIST commands control the printing of the source program and object code listings.

SOURCE PROGRAM LISTING

The source listing shows the source program compiled. Each line (card or card image) appears as it exists on the file identified by the I parameter on the compiler call, including any sequence numbers in columns 1 through 6 and any program identification in columns 73 through 80. Lines are individually numbered for correlation with any diagnostic messages.

Source programs with COPY statements appear as shown in section 9, Library Facility.

Figure 12-1 shows an example of a source listing. The header line shows the program name. The end of the source listing is marked by two lines that correspond to the position of columns in the source input.

SELECTIVE LIST/NOLIST OPTIONS

Printing of the source program and object code listings can be turned on and off by using the LIST/NOLIST commands. These commands are specified on comment lines (lines with an asterisk in column 7). The commands are always printed. The commands that can be specified are:

ONSOURCE	Turns on source listing
ONOBJLIST	Turns on object listing
ONALLLIST	Turns on source and object listing
OFFSOURCE	Turns off source listing
OFFOBJLIST	Turns off object listing
OFFALLLIST	Turns off source and object listings

A LIST/NOLIST command must begin in column 8 immediately following the asterisk. The command cannot contain any embedded blanks; however, a blank must follow the command. The remainder of the comment line is ignored. A command can appear anywhere a comment line is legal, including in copied lines.

The source listing is initially turned on for each program in the compilation and can be turned off and on as many times as desired. If the L=0 or LO=-S option is specified on the compiler call, the source listing commands have no effect.

The object listing is initially turned on if the LO=O option is specified on the compiler call; otherwise, it is initially turned off. Object listing commands that appear outside the Procedure Division are ignored. If L=0 is specified no object listing is produced.

```
            SOURCE LISTING OF

    1          1  IDENTIFICATION DIVISION.
    2             PROGRAM-ID.  TBINT.
    3             AUTHOR.  CDC.
    4            *    INTAB TABULATES INTEREST GIVEN AMOUNT BORROWED, RATE OF INT
    5            *    AND DATE OF FIRST PAYMENT..REPORTS ON NUMBER OF PAYMENT,
    6            *    DUE DATE, INTEREST, ACCUMULATED INTEREST, PRINCIPAL,
    7            *    ACCUMULATED PRINCIPAL, AND BALANCE DUE FOR EACH PAYMENT.
    8            *      INPUT TO INTAB IS.. AMOUNT BORROWED (9(5)V99) COL.1-7
    9            *                          RATE OF INTEREST (V99999) COL.8-12
                                                  .
                                                  .
                                                  .
  125         91  WRAPUP.
  126         92      CLOSE IN-DATA-FILE, REPORT-FILE.
  127         93      STOP RUN.
  128         94  CALCULATE-LAST.
  129         95      SUBTRACT L-INT FROM INST GIVING R-PRIN.
  130         96      SUBTRACT R-PRIN FROM W-AMT.
  131         97      PERFORM L1 THRU L2, GO TO FINAL-TEST.

          COLUMN    1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 12-1. Source Listing Example

Any commands issued within the Report Section are ignored because generated code for reports cannot be printed.

MAP

The map listing names each procedure-name and data item defined in the program. The addresses shown are those of the named items within the compiled program.

The map begins and ends with headers:

MAP OF program-name

*** END MAP ***

The listing contains a data map and a procedure map. Figure 12-2 shows an example. The data map lists each data item defined in the Data Division, with items in each section listed separately. Data items defined in the Report Section of the Data Division do not appear in a listing produced by the data map option (LO=M).

In the File Section of the map, the following information is shown for each FD entry:

FD xxxx (yyyy)
: File-name specified in SELECT clause is xxxx. The implementor-name of the ASSIGN clause is yyyy (appears in parentheses below file-name).

BLOCK=PROGRAM
: Block of code has the same name as the program-name truncated, if necessary, to seven characters.

ADDR/BCP=nn/
: Address of block of code is nn octal within the job field length. Beginning character position is the beginning of that address.

LNR=linenum
: Source program line number at which entry is defined is linenum.

* 01
: Identifies beginning of a Record Description entry that starts with level-number 01.

For each data item, the following are listed in the map:

nn/mm	Address of item within block is word nn_8. Within that word, the item begins in byte mm_8. Bytes are numbered 0 through 11_8 (0 through 9_{10}) left to right.
SZ=zz	Number of 6-bit bytes in item is zz_{10}.
category	Category of item as determined by PICTURE or USAGE clause descriptions.
linenum	Source line at which item is defined.

Data items in other than the File Section are described similarly. The block for items defined in the Common-Storage Section is the common block CCOMMON. Common blocks have names delimited by a slash.

The procedure map lists the beginning address for each procedure specified. The addresses are in octal and relative to the beginning of the program block. The source line at which the procedure is defined is also listed. Procedures are listed by any sections defined.

SECTION	Named procedure is a section- name. If SECTION does not appear, the procedure is a paragraph-name.

```
                                        MAP OF TBINT

                                        *** DATA MAP (ADDR/BCP IN OCTAL, SZ IN DECIMAL) ***

                           FILE SECTION
 FD     IN-DATA-FILE                    BLOCK=PROGRAM    ADDR/BCP=000171/                    LNR=22
         (INPUT)
 *  01  INFORMATION                                               000030/00  SZ=81  GROUP        25
    02  AMT                                                       000030/00      7  NUMERIC      26
    02  RATE                                                      000030/07      5  NUMERIC      27
    02  INST                                                      000031/02      5  NUMERIC      28
    02  DAT                                                       000031/07      6  GROUP        29
    03  MO                                                        000031/07      2  NUMERIC      30
                                                  .
                                                  .
                                                  .
          CALCULATE-LINE                                          000301                        104
          L1                                                      000301                        108
          L2                                                      000301                        109
          FINAL-TEST                                              000301                        115
          WRAPUP                                                  000301                        125
          CALCULATE-LAST                                          000301                        128

                                            *** END MAP ***
```

Figure 12-2. Map Listing Example

CROSS REFERENCE MAP

The cross-reference map shows program references to each data item and procedure defined in the program. Figure 12-3 shows an example.

The map is divided into four parts:

Referenced data-names

Unreferenced data-names

Referenced procedure-names

Unreferenced procedure-names

In each part, the following information is shown for each item:

name	Name of data item or procedure
LINE	Source line at which item is defined
COLUMN	Column within source line LINE at which item is defined
REFERENCE(S)	Source lines in which the item is referenced

OBJECT CODE

The object listing shows the COMPASS assembly language line image generated by the compiler. Octal contents of program locations and other information not normally the concern of application programmers is also listed.

DIAGNOSTICS

Diagnostics detected during compilation are controlled by the EL, ANSI, and FIPS parameters on the compiler call, as described in appendix B. Figure 12-4 shows the diagnostic summary.

The listing shows the following:

SEV	Severity of error: N Nonstandard T Trivial W Warning F Fatal C Catastrophic
LINE	Source line in which the condition was detected
COL	Column number within source line LINE at which the condition was detected; or SS if source of error is sub-schema, in which case LINE pertains to sub-schema source statement
ERROR	Message number

```
                                      CROSS  REFERENCE FOR TBINT
      REFERENCED DATA-NAMES            LINE  COLUMN            REFERENCE(S)

    AMT                                26     15               94     96     103
    COL-LINE                           68     12               100
    DA                                 44     18               110
    DA                                 31     18               110
    DASH-LN1                           73     12               99
                                         .
                                         .
                                         .
    H3                                 102    8
    INIT-IT                            90     8
    TRY-IT                             89     8

      UNREFERENCED PROCEDURE-NAMES LINE COLUMN
```

Figure 12-3. Cross Reference Listing Example

```
                                            DIAGNOSTICS IN COPYEXA
     SEV LINE COL ERROR

      T     2  21  1057  ONLY THE FIRST 7 CHARACTERS OF THE PROGRAM
                         NAME ARE USED TO INTERACT WITH THE SYSTEM.

      N    48  12  7284  THE INITIALIZE STATEMENT IS NON-STANDARD.

      N    49  12  7284  THE INITIALIZE STATEMENT IS NON-STANDARD.

                           **      3 ERRORS LISTED                **

                           **     NO UNLISTED ERRORS              **
```

Figure 12-4. Diagnostic Listing Example

The full text of the error message completes each line.

At the end of the individual diagnostics, the listed and unlisted messages are summarized.

EXECUTION

COBOL programs are executed through operating system control statements. Several parameters can be specified on the execution call control statement that cause certain actions to occur at execution time.

EXECUTION CALL CONTROL STATEMENTS

After a COBOL program has been compiled, it can be executed through a control statement that specifies the logical file name of the file containing the object program. The file is specified by the B parameter on the compiler call. A compiled program can be executed immediately after compilation in the same run unit or later in a separate run unit.

If a program is compiled and executed in the same run unit and the B parameter is not specified on the compiler call, the execution call control statement must specify the system file LGO.

If a program is compiled and executed in separate run units, the program is executed either by specifying the name of the file containing the program or by using the LOAD and EXECUTE control statements. See the appropriate operating system reference manual and the CYBER Loader reference manual for more information about program execution.

EXECUTION PARAMETERS

Parameters can be added to an execution call control statement to cause several functions to be performed by the system at execution time. The parameters conform to operating system syntax and cannot exceed one line. An asterisk must appear as the first character of each parameter except the file equivalence parameter. All parameters are optional and can appear in any order. Any unrecognized parameters are ignored by the system and are assumed to be user parameters.

The user parameters can be processed within the program by entering the utility routine C.GETEP. See section 15, Inter-Program Communication Facility.

APPL

The APPL parameter specifies the name of a Message Control System (MCS) application that communicates with the COBOL program. This feature is only allowed under NOS. The APPL parameter is required as part of the execution control statement of any program using MCS. The execution statement passes the application name to MCS during the program initialization. The application name is used by MCS to associate the COBOL program with the correct application.

CORE

The CORE parameter causes the maximum amount of central memory used during execution to be displayed on the job dayfile when the program completes execution. The amount is displayed as an octal value.

MSGS

The MSGS parameter causes the messages generated by the Sort/Merge facility, which indicate the number of records processed, to be written to the job dayfile. If the parameter is omitted, the messages are not displayed.

NOTRIP

The NOTRIP parameter prevents the generation of the triple space carriage control character (hyphen) in lines created by the WRITE statement with the ADVANCING phrase or by the Report Writer. The character is not legal on some devices under the NOS operating system. If the character is legal, NOTRIP might cause a larger print file and slower printing.

TIME

The TIME parameter causes the amount of time used for execution to be displayed on the job dayfile when the program completes execution. Only central processing time is given.

Figure 12-5 shows an example of the CORE and TIME parameters.

File Equivalence

One file-name can be substituted for another file-name through the file equivalence parameter. The format of the parameter is:

file-name-1=file-name-2 [,file-name-3=file-name-4]...

File-name-1 is the implementor-name specified in the ASSIGN clause of the File-Control entry of any program in the run unit or the implementor-name specified in the SPECIAL-NAMES paragraph with the mnemonic name used in any DISPLAY or ACCEPT statement in the run unit. File-name-2 is substituted for file-name-1 when the file is opened.

The file equivalence mechanism of the FILE control statement must not be used. If a FILE control statement is used, it must specify file-name-2.

```
08.24.32.LGO,*TIME,*CORE.
08.24.35.0000. 12 CPU SECONDS EXECUTION TIME
08.24.35.    015700B SCM MAXIMUM USED
```

Figure 12-5. Execution Parameter Example

DECK STRUCTURE

A job deck submitted for execution through a card reader begins with a job statement and ends with a card with a 6/7/8/9 multiple punch in column 1. Between the job statement and the end-of-information card, the deck is divided into sections separated by a card with a 7/8/9 multiple punch in column 1.

Structure for a simple compilation of a program submitted in the deck and execution with data that is also part of the deck is shown in figure 12-6.

In the deck, all input data is part of the deck itself and is known to the operating system by the name INPUT. If input files were stored on magnetic tape or permanent files, control statements to attach the input files to the job would be required before the LGO control statement.

Output data from the deck above is written to the standard job output file, OUTPUT, and is printed upon job termination unless a control statement indicates another disposition. If output files were to be stored on magnetic tape or permanent files, appropriate operating system control statements would be required before and/or after the LGO control statement.

```
Job Statement
USER statement                      NOS only
CHARGE statement                    NOS only
ACCOUNT statement                   NOS/BE only
COBOL5 compiler call
LGO execution call
7/8/9

    Source program to be compiled

7/8/9

    Data to be used during execution

6/7/8/9
```

Figure 12-6. Compilation and Execution Deck Structure Example

See the appropriate operating system reference manual for control statements required to store and retrieve files and otherwise manipulate data for program execution.

EXAMPLES

Program TBINT (figure 13-1) tabulates interest paid on a loan and produces a table for each payment that shows the date the payment is due, interest, accumulated interest, principal, accumulated principal, and balance due.

Input to the program is a series of cards in the following format:

Column	Contents	Picture
1-7	Amount borrowed	9(5)V99
8-12	Annual interest rate	V99999
13-17	Payment amount	9(3)V99
18-23	First payment date:	
	month	99
	day	99
	year	99

One table is produced for each input card.

Line 10 specifies that the input data is known within the program as IN-DATA-FILE and known to the operating system as the file INPUT. Since INPUT is a reserved COBOL word, it should be specified as a nonnumeric literal (for strict ANSI usage). It is accepted, however, as a legal implementor name, if it is specified without the quotes.

Line 11 equates the output file, REPORT-FILE, to the standard system job output that is printed at the end of the job. As with INPUT, OUTPUT should be specified as a nonnumeric literal (for strict ANSI usage). It is accepted, however, as a legal implementor name, if it is specified without the quotes.

Lines 14 through 16 are the FD entry for file IN-DATA-FILE. Lines 17 through 214 are the Record Description entry for the file.

Lines 44 through 52 describe a header line for the table.

Lines 351 through 43 describe level 77 items that are not part of a record. Level 77 items need not appear at the beginning of the Working-Storage Section.

Line 61 begins the Procedure Division. Both the input file and output file are opened, the first data card is read, and the information for the table is calculated from the card data.

Line 67 initializes the line count. The MOVE statement could be replaced by the following statement when the program need not conform to strict ANSI usage. Since LINE-CT is specified with a PICTURE clause that defines a numeric item, INITIALIZE sets the item to zeros.

```
    INITIALIZE LINE-CT.
```

Lines 71 through 712 show PERFORM statement use, as does line 97.

Line 791 shows the CORRESPONDING phrase of MOVE that results in the date from the input card being moved to appropriate positions in the table header line.

The following data cards were used as input to the program:

```
    00500001815702500110575

    03000000750009332041572
```

a. Source Program Listing

```
1    IDENTIFICATION DIVISION.
     PROGRAM-ID.  TBINT.
     AUTHOR.  CDC.
     *     INTAB TABULATES INTEREST GIVEN AMOUNT BORROWED, RATE OF INT
     *     AND DATE OF FIRST PAYMENT..REPORTS ON NUMBER OF PAYMENT,
     *     DUE DATE, INTEREST, ACCUMULATED INTEREST, PRINCIPAL,
     *     ACCUMULATED PRINCIPAL, AND BALANCE DUE FOR EACH PAYMENT.
     *       INPUT TO INTAB IS.. AMOUNT BORROWED (9(5)V99) COL.1-7
     *                           RATE OF INTEREST (V99999) COL.8-12
     *                           INSTRUMENT      (9(3)V99) COL.13-17
     *                           DATE OF PAYMENT 1 (9(6))  COL.18-23
     REPLACE ==TEST== BY ==TEST-AMT==.
4    ENVIRONMENT DIVISION.
5    CONFIGURATION SECTION.
     SOURCE-COMPUTER.  CYBER-170.
     OBJECT-COMPUTER.  CYBER-170.
8    INPUT-OUTPUT SECTION.
9    FILE-CONTROL.
         SELECT IN-DATA-FILE ASSIGN ≠INPUT≠.
         SELECT REPORT-FILE ASSIGN ≠OUTPUT≠.
12   DATA DIVISION.
13   FILE SECTION.
14   FD  IN-DATA-FILE
15       LABEL RECORD OMITTED
16       DATA RECORD IS INFORMATION.
17   01  INFORMATION.
18       02 AMT PICTURE  9(5)V99.
19       02 RATE PICTURE  V99999.
20       02 INST PICTURE  999V99.
21       02 DAT.
211         03 MO       PICTURE 99.
212         03 DA       PICTURE 99.
213         03 YR       PICTURE 99.
214      02 FILLER PICTURE  X(57).
22   FD  REPORT-FILE
23       LABEL RECORD OMITTED
24       DATA RECORD IS PRINT-RECORD.
25   01  PRINT-RECORD.
26       02 FILLER      PICTURE X(14).
27       02 RPT-MO      PICTURE ZZZ.
28       02 FILLER      PICTURE X(10).
281      02 DUE-DT.
282         03 MO       PICTURE Z9.
283         03 FILL1    PICTURE X.
284         03 DA       PICTURE Z9.
285         03 FILL2    PICTURE X.
286         03 YR       PICTURE 99.
287      02 FILLER      PICTURE X(11).
29       02 RPT-INT     PICTURE $$$$.99.
30       02 FILLER      PICTURE X(14).
31       02 TOTL-INT    PICTURE $$$,$$$.99.
32       02 FILLER      PICTURE X(11).
33       02 RPT-PRIN    PICTURE $$$$.99.
34       02 FILLER      PICTURE X(15).
341      02 TOTL-PRIN   PICTURE $$$,$$$.99.
342      02 FILLER      PICTURE X(5).
343      02 RPT-BAL     PICTURE $$$,$$$.99.
35   WORKING-STORAGE SECTION.
44   01  HDR-LINE.
45       02 FILLER      PICTURE X(37)  VALUE ≠1                        INITI
46 -     ≠AL AMOUNT   ≠.
47       02 HDR-AMT     PICTURE $(6).99.
48       02 FILLER      PICTURE X(17)  VALUE ≠            RATE    ≠.
49       02 HDR-RATE    PICTURE .99999.
50       02 FILLER      PICTURE X(10)  VALUE ≠      AT    ≠.
51       02 HDR-INS     PICTURE $$$$.99.
52       02 FILLER      PICTURE X(52)  VALUE ≠    PER MONTH
53 -     ≠                              ≠.
```

Figure 13-1. Program TBINT (Sheet 1 of 3)

a. Source Program Listing (Contd)

```
54  01  COL-LINE.
55      02 FILLER     PICTURE X(68)  VALUE ≠           MONTHS
56 -    ≠  DUE DATE           INTEREST         INTERES≠.
57      02 FILLER     PICTURE X(68)  VALUE ≠T-TO-DATE          PRINCIP
58 -    ≠AL        PRINCIPAL-TO-DATE        BALANCE ≠.
581 01  DASH-LN1      PICTURE X(137) VALUE ≠                     -----
582-    ≠-------------------------------------------------------------
583-    ≠------------                                        ≠.
584 01  DASH-LN2      PICTURE X(137) VALUE ≠           ------
585-    ≠  --------        --------       ----------------
586-    ≠ ---------      -----------------         -------  ≠.
351 77  TOT-INT       PICTURE 9(5)V99  VALUE ZEROES.
352 77  TOT-PRIN      PICTURE 9(5)V99  VALUE ZEROES.
36  77  LINE-CT       PICTURE 99     VALUE ZERO.
38  77  RR            PICTURE V99999.
39  77  W-AMT         PICTURE 9(5)V99.
40  77  W-MO          PICTURE 999.
41  77  L-INT         PICTURE 999V99.
42  77  R-PRIN        PICTURE 999V99.
43  77  TEST          PICTURE 999V99.
61  PROCEDURE DIVISION.
    TRY-IT SECTION.
    INIT-IT.
        OPEN INPUT IN-DATA-FILE OUTPUT REPORT-FILE.
64  READ-IN-DATA.
65      READ IN-DATA-FILE AT END GO TO WRAPUP.
66      IF AMT = INST GO TO WRAPUP.
67  HD. MOVE ZEROES TO LINE-CT.
68      MOVE AMT TO HDR-AMT, MOVE RATE TO HDR-RATE.
69      MOVE INST TO HDR-INS, MOVE HDR-LINE TO PRINT-RECORD.
70  PR. WRITE PRINT-RECORD, MOVE SPACES TO PRINT-RECORD.
71  H2. MOVE DASH-LN1 TO PRINT-RECORD, PERFORM PR.
711     MOVE COL-LINE TO PRINT-RECORD, PERFORM PR.
712     MOVE DASH-LN2 TO PRINT-RECORD, PERFORM PR.
72  H3. DIVIDE 12 INTO RATE GIVING RR ROUNDED.
73      MOVE AMT TO W-AMT, MOVE 1 TO W-MO.
74  CALCULATE-LINE.
75      MULTIPLY W-AMT BY RR GIVING L-INT ROUNDED.
76      SUBTRACT L-INT FROM INST GIVING R-PRIN.
77      SUBTRACT R-PRIN FROM W-AMT.
78  L1. MOVE W-AMT TO RPT-BAL, MOVE R-PRIN TO RPT-PRIN.
79  L2. MOVE W-MO TO RPT-MO, MOVE L-INT TO RPT-INT.
791     MOVE CORRESPONDING DAT TO DUE-DT, MOVE ≠/≠ TO FILL1, FILL2.
792     ADD L-INT TO TOT-INT, ADD R-PRIN TO TOT-PRIN.
793     MOVE TOT-INT TO TOTL-INT, MOVE TOT-PRIN TO TOTL-PRIN
80      PERFORM PR, ADD 1 TO LINE-CT.
82      IF LINE-CT GREATER THAN 51 PERFORM HD THRU H2.
83  FINAL-TEST.
84      ADD 1 TO W-MO, ADD 1 TO MO OF DAT.
841     IF MO OF DAT GREATER THAN 12, MOVE 1 TO MO OF DAT,
842     ADD 1 TO YR OF DAT.
85      IF INST LESS THAN W-AMT GO TO CALCULATE-LINE.
86      MULTIPLY W-AMT BY RR GIVING L-INT ROUNDED.
87      ADD W-AMT, L-INT GIVING TEST.
88      IF INST LESS THAN TEST GO TO CALCULATE-LAST.
881     MOVE ZEROES TO RPT-BAL, MOVE W-AMT TO RPT-PRIN, PERFORM L2.
90      MOVE ZEROES TO TOT-INT, TOT-PRIN,  GO TO READ-IN-DATA.
91  WRAPUP.
92      CLOSE IN-DATA-FILE, REPORT-FILE.
93      STOP RUN.
94  CALCULATE-LAST.
95      SUBTRACT L-INT FROM INST GIVING R-PRIN.
96      SUBTRACT R-PRIN FROM W-AMT.
97      PERFORM L1 THRU L2, GO TO FINAL-TEST.
```

Figure 13-1. Program TBINT (Sheet 2 of 3)

b. Program Output

```
                INITIAL AMOUNT    $500.00         RATE   .10157     AT     $25.00   PER MONTH
                --------------------------------------------------------------------------
MONTHS      DUE DATE       INTEREST      INTEREST-TO-DATE        PRINCIPAL       PRINCIPAL-TO-DATE         BALANCE
------      --------       --------      ----------------        ---------       -----------------         -------
     1      11/ 5/75          $7.57                 $7.57           $17.43                  $17.43         $482.57
     2      12/ 5/75          $7.30                $14.87           $17.70                  $35.13         $464.87
     3       1/ 5/76          $7.03                $21.90           $17.97                  $53.10         $446.90
     4       2/ 5/76          $6.76                $28.66           $18.24                  $71.34         $428.66
     5       3/ 5/76          $6.49                $35.15           $18.51                  $89.85         $410.15
     6       4/ 5/76          $6.21                $41.36           $18.79                 $108.64         $391.36
     7       5/ 5/76          $5.92                $47.28           $19.08                 $127.72         $372.28
     8       6/ 5/76          $5.63                $52.91           $19.37                 $147.09         $352.91
     9       7/ 5/76          $5.34                $58.25           $19.66                 $166.75         $333.25
    10       8/ 5/76          $5.04                $63.29           $19.96                 $186.71         $313.29
    11       9/ 5/76          $4.74                $68.03           $20.26                 $206.97         $293.03
    12      10/ 5/76          $4.43                $72.46           $20.57                 $227.54         $272.46
    13      11/ 5/76          $4.12                $76.58           $20.88                 $248.42         $251.58
    14      12/ 5/76          $3.81                $80.39           $21.19                 $269.61         $230.39
    15       1/ 5/77          $3.49                $83.88           $21.51                 $291.12         $208.88
    16       2/ 5/77          $3.16                $87.04           $21.84                 $312.96         $187.04
    17       3/ 5/77          $2.83                $89.87           $22.17                 $335.13         $164.87
    18       4/ 5/77          $2.49                $92.36           $22.51                 $357.64         $142.36
    19       5/ 5/77          $2.15                $94.51           $22.85                 $380.49         $119.51
    20       6/ 5/77          $1.81                $96.32           $23.19                 $403.68          $96.32
    21       7/ 5/77          $1.46                $97.78           $23.54                 $427.22          $72.78
    22       8/ 5/77          $1.10                $98.88           $23.90                 $451.12          $48.88
    23       9/ 5/77           $.74                $99.62           $24.26                 $475.38          $24.62
    24      10/ 5/77           $.37                $99.99           $24.62                 $499.64            $.00
                INITIAL AMCUNT   $3000.00         RATE   .07500     AT     $93.32   PER MONTH
                --------------------------------------------------------------------------
MONTHS      OLE DATE       INTEREST      INTEREST-TO-DATE        PRINCIPAL       PRINCIPAL-TO-DATE         BALANCE
------      --------       --------      ----------------        ---------       -----------------         -------
     1       4/15/72         $18.75                $18.75           $74.57                  $74.57       $2,925.43
     2       5/15/72         $18.28                $37.03           $75.04                 $149.61       $2,850.39
     3       6/15/72         $17.81                $54.84           $75.51                 $225.12       $2,774.88
     4       7/15/72         $17.34                $72.18           $75.98                 $301.10       $2,698.90
     5       8/15/72         $16.87                $89.05           $76.45                 $377.55       $2,622.45
     6       9/15/72         $16.39               $105.44           $76.93                 $454.48       $2,545.52
     7      10/15/72         $15.91               $121.35           $77.41                 $531.89       $2,468.11
     8      11/15/72         $15.43               $136.78           $77.89                 $609.78       $2,390.22
     9      12/15/72         $14.94               $151.72           $78.38                 $688.16       $2,311.84
    10       1/15/73         $14.45               $166.17           $78.87                 $767.03       $2,232.97
    11       2/15/73         $13.96               $180.13           $79.36                 $846.39       $2,153.61
    12       3/15/73         $13.46               $193.59           $79.86                 $926.25       $2,073.75
    13       4/15/73         $12.96               $206.55           $80.36               $1,006.61       $1,993.39
    14       5/15/73         $12.46               $219.01           $80.86               $1,087.47       $1,912.53
    15       6/15/73         $11.95               $230.96           $81.37               $1,168.84       $1,831.16
    16       7/15/73         $11.44               $242.40           $81.88               $1,250.72       $1,749.28
    17       8/15/73         $10.93               $253.33           $82.39               $1,333.11       $1,666.89
    18       9/15/73         $10.42               $263.75           $82.90               $1,416.01       $1,583.99
    19      10/15/73          $9.90               $273.65           $83.42               $1,499.43       $1,500.57
    20      11/15/73          $9.38               $283.03           $83.94               $1,583.37       $1,416.63
    21      12/15/73          $8.85               $291.88           $84.47               $1,667.84       $1,332.16
    22       1/15/74          $8.33               $300.21           $84.99               $1,752.83       $1,247.17
    23       2/15/74          $7.79               $308.00           $85.53               $1,838.36       $1,161.64
    24       3/15/74          $7.26               $315.26           $86.06               $1,924.42       $1,075.58
    25       4/15/74          $6.72               $321.98           $86.60               $2,011.02         $988.98
    26       5/15/74          $6.18               $328.16           $87.14               $2,098.16         $901.84
    27       6/15/74          $5.64               $333.80           $87.68               $2,185.84         $814.16
    28       7/15/74          $5.09               $338.89           $88.23               $2,274.07         $725.93
    29       8/15/74          $4.54               $343.43           $88.78               $2,362.85         $637.15
    30       9/15/74          $3.98               $347.41           $89.34               $2,452.19         $547.81
    31      10/15/74          $3.42               $350.83           $89.90               $2,542.09         $457.91
    32      11/15/74          $2.86               $353.69           $90.46               $2,632.55         $367.45
    33      12/15/74          $2.30               $355.99           $91.02               $2,723.57         $276.43
    34       1/15/75          $1.73               $357.72           $91.59               $2,815.16         $184.84
    35       2/15/75          $1.16               $358.88           $92.16               $2,907.32          $92.68
    36       3/15/75           $.58               $359.46           $92.68               $2,999.40            $.00
```

Figure 13-1. Program TBINT (Sheet 3 of 3)

SUB-SCHEMA Facility 14

The Sub-Schema facility is an interface to the CYBER Database Control System (CDCS). It allows a COBOL program to access a sub-schema in lieu of defining files through FD entries. Through the sub-schema, the COBOL program can also read files that have been logically joined together in a relation. The material in this section discusses separately both versions of CDCS: CDCS 1 and CDCS 2. For detailed information, refer to the appropriate CDCS reference manual and the appropriate DDL reference manual.

COBOL SUB-SCHEMA - CDCS 1

The COBOL sub-schema is created by the Data Description Language (DDL 2) compiler and resides on mass storage as a permanent file. It contains descriptions of one or more data base files that are made available to the COBOL program. Only one sub-schema can be referenced in a COBOL program. Data base files can have sequential, initial indexed, initial direct, or initial actual-key file organization.

During compilation of a COBOL program that accesses a sub-schema, file descriptions are obtained from the sub-schema and incorporated into compilation. The program can reference items in the data base files with conventional input and output statements and item references in the Procedure Division.

A sub-schema source listing (or LO=M specified on the compiler call statement) can be used to obtain data-names, item descriptions, and other information needed to process the data base files. The listing contains the sub-schema-name, file-names, record-names, data-names of items, and relation-names. Primary and alternate keys are also listed for the applicable files. The files joined in a relation are shown at the end of the listing.

PROGRAM CODING

Coding a program that accesses a sub-schema is basically the same as coding any other program. A few requirements and restrictions are imposed on the program.

Environment Division

In the Environment Division, the SPECIAL-NAMES paragraph and the FILE-CONTROL paragraph are affected by sub-schema use. The SUB-SCHEMA clause is required in the SPECIAL-NAMES paragraph. Only restricted File-Control entries can be specified for data base files.

SUB-SCHEMA Clause

The SUB-SCHEMA clause (figure 14-1) in the SPECIAL-NAMES paragraph is required to identify the sub-schema and acknowledge sub-schema use.

Sub-schema-name specifies the name of the sub-schema, which is obtained from the Title Division of the sub-schema. The sub-schema must reside on the file identified by the D parameter of the compiler call statement.

```
SUB-SCHEMA IS sub-schema-name
```

Figure 14-1. SUB-SCHEMA Clause Format

File-Control Entry

In the FILE-CONTROL paragraph, a limited File-Control entry must be included for each file that has a description obtained from the sub-schema. The SELECT clause and the ASSIGN clause must be specified. The FILE STATUS clause can be specified. Any other clause is ignored. The file-name for the SELECT clause is obtained from the Realm Division of the sub-schema. The ASSIGN clause includes implementor-name-2 when alternate keys are defined for the file.

Data Division

In the Data Division, the File Section must not include an FD entry or Record Description entry for any file with a description in the sub-schema. Files that are not data base files can be described in the File Section. The section header must be specified even if the only files used by the program are data base files.

Procedure Division

In the Procedure Division, a data base file is processed as if it had been described in the File Section of the Data Division. Record-names and data-names are obtained from the Record Division of the sub-schema. Some statements cannot be used with data base files. Format 3 and format 4 of the READ statement are used to read relations.

Statement Restrictions

Data base files can be referenced in any of the Procedure Division statements except the following:

ACCEPT

DISPLAY

GIVING phrase of SORT or MERGE

USING phrase of MERGE

READ Statement

Format 3 and format 4 of the READ statement (figure 14-2) are used to read data base relations. A relation can be read sequentially or randomly. The rules discussed in section 5 for reading a file also apply when reading a relation.

```
Format 3

     READ relation-name [NEXT] RECORD

          [; AT END imperative-statement]


Format 4

     READ relation-name RECORD

          [; KEY IS data-name]

          [; INVALID KEY imperative-statement]
```

Figure 14-2. READ Statement Format 3 and Format 4

A relation is the means by which a set of files within the sub-schema are joined together. The files are joined on the basis of relationships that exist between data items in the respective files. For example, one file in a three-file relationship is joined to a second file through a data item contained in both files; the second file is then joined to a third file through the same or another data item that is contained in both the second and third files. The first file is the root file, which is the file at which the relation is entered. The files in the relation are traversed from the first file to the second file to the third file.

Relations are referenced only in the Procedure Division; however, each file in the relation must be specified in the FILE-CONTROL paragraph by the SELECT and ASSIGN clauses.

Relation-name is the name of a relation specified in the sub-schema. Format 3 reads a relation sequentially; format 4 reads a relation randomly. Data-name must reference a primary or alternate key in the root file. When the READ statement is executed, a record from each file in the relation is available to the COBOL program.

COBOL SUBPROGRAMS

Data base files can be accessed by COBOL subprograms. The main program and the subprogram must include the SUB-SCHEMA clause. File-Control entries are specified in the main program for all data base files referenced in the run unit. A subprogram specifies File-Control entries only for those data base files it references. The FDL parameter must be specified on the compiler call statement and an FDL file must be created. See section 15, Inter-Program Communication Facility.

COMPILATION AND EXECUTION

Compilation and subsequent execution of a COBOL program are initiated by control statements. When a COBOL sub-schema is accessed by the program, certain requirements must be fulfilled before compiling or executing the program.

Before a program referencing a sub-schema can be compiled, the file on which the sub-schema resides must be attached. The D parameter on the compiler call must specify the logical file name of the sub-schema file. If a COBOL subprogram that accesses the sub-schema is being compiled, the FDL file must have been created and the FDL parameter must be included on the compiler call statement.

When the compiled program is executed, CDCS monitors and interprets all requests for action on relations and on data base files, directing all operations involving the storage and retrieval of data to and from the file through CYBER Record Manager. The following files must be attached for execution of the program:

The file containing the schema

The file containing the sub-schema

The data base files specified in SELECT clauses and associated index files, if applicable

The log file and data base procedure library, if required by the schema

RELATION ERROR PROCESSING

The status of data base files in a relation can be obtained by entering the COMPASS object routine C.DMRST. This routine scans the files in the order the files are traversed, beginning with the root file and continuing until a nonzero error code is encountered or all files have been scanned. C.DMRST should be entered after a READ statement is executed for a relation in order to determine whether an error occurred during the read operation and on which file the error occurred. To learn the status of files after a relational read, the following statement must be executed:

```
ENTER "C.DMRST" USING relation-name,
    data-name-1, data-name-2.
```

The USING phrase specifies the name of the relation to be checked for file status and two data items for returned information. Data-name-1 must be an elementary item described as PICTURE X(7). Data-name-2 must be a numeric integer data item with a size greater than two and a usage of COMPUTATIONAL-1.

If an error, a null condition, or a control break occurred on a file, the implementor-name (logical file name) of the file is returned in data-name-1; the CDCS status or error code is returned in data-name-2. The statement can be executed again to check for errors or relation conditions on the remaining files in the relation. If no errors, null records, or control breaks occurred on the files in the relation, zero is returned to data-name-2; the contents of data-name-1 are undefined.

COBOL SUB-SCHEMA - CDCS 2

The COBOL sub-schema is created by the Data Description Language (DDL3) compiler and resides on mass storage as a permanent file. It contains descriptions of one or more data base files that are made available to the COBOL program. Only one sub-schema can be referenced in a COBOL program. Data base files can have extended indexed, extended direct, or extended actual-key file organization.

During compilation of a COBOL program that accesses a sub-schema, file descriptions are obtained from the sub-schema and incorporated into compilation. The program can reference items in the data base files with conventional input and output statements and item references in the Procedure Division.

A sub-schema source listing (or LO=M specified on the compiler call) can be used to obtain data-names, item descriptions, and other information needed to process the data base files. The listing contains the sub-schema-name, file-names, record-names, data-names of items, and relation-names. Primary and alternate keys are also listed for the applicable files. The files joined in a relation are shown at the end of the listing.

PROGRAM CODING

Coding a program that accesses a sub-schema is basically the same as coding any other program. A few requirements and restrictions are imposed on the program.

Environment Division

In the Environment Division, the SPECIAL-NAMES paragraph and the FILE-CONTROL paragraph are affected by sub-schema use. The SUB-SCHEMA clause is required in the SPECIAL-NAMES paragraph. Only restricted File-Control entries can be specified for data base files.

SUB-SCHEMA Clause

The SUB-SCHEMA clause in the SPECIAL-NAMES paragraph is required to identify the sub-schema and acknowledge sub-schema use. The SUB-SCHEMA clause format is identical to that shown in figure 14-1 for CDCS 1.

Sub-schema-name specifies the name of the sub-schema, which is obtained from the Title Division of the sub-schema. The sub-schema must reside on the file identified by the D parameter of the compiler call statement.

File-Control Entry

The SELECT clause and the ASSIGN clause are required only if the FILE STATUS clause is used. Any other clause is ignored. If no File-Control entries are specified, the FILE-CONTROL and the INPUT-OUTPUT SECTION headers are not required.

The file-name for the SELECT clause is obtained from the Realm Division of the sub-schema. The ASSIGN clause includes implementor-name-2 when alternate keys are defined for the file.

Data Division

In the Data Division, the File Section must not include an FD entry or Record Description entry for any file with a description in the sub-schema. Files that are not data base files can be described in the File Section. The section header must be specified even if the only files used by the program are data base files.

Procedure Division

In the Procedure Division, a data base file is processed as if it had been described in the File Section of the Data Division. Record-names and data-names are obtained from the Record Division of the sub-schema. Some statements cannot be used with data base files. Format 3 and format 4 of the READ statement are used to read relations. Format 2 of the OPEN statement is used to open all files within a relation. Format 2 of the CLOSE statement is used to close all files within a relation. Format 2 of the START statement is used to position the root file in a relation and to establish a key of reference for the root file. The USE FOR ACCESS CONTROL declarative statement specifies a procedure used in generating access control keys. The USE FOR DEADLOCK declarative statement specifies a procedure used in recovering from a deadlock situation.

Statement Restrictions

Data base files can be referenced in any of the Procedure Division statements except the following:

ACCEPT

DISPLAY

GIVING phrase of SORT or MERGE

USING phrase of MERGE

CLOSE Statement

The format 2 CLOSE statement (figure 14-3) closes all files that are associated with a specified relation. The statement is performed as if a separate close were executed for each file, in the order of the rank of the files in the relation. Files closed by a relation CLOSE statement (format 2) should not be explicitly closed by a file CLOSE statement (format 1).

Format 2

CLOSE relation-name [WITH **LOCK**] . . .

Figure 14-3. CLOSE Statement Format 2

OPEN Statement

The format 2 OPEN statement (figure 14-4) opens all files that are associated with a specified relation. The statement is performed as if a separate open were executed for each file, in the order of the rank of the files in the relation. Files opened by a relation OPEN statement (format 2) should not be explicitly opened by a file OPEN statement (format 1).

Relation-name is defined in the sub-schema for the program. Relations are normally opened for input only, but can be opened for I-O if locking of records read is desired or if updating of the individual files is desired.

READ Statement

Format 3 and format 4 of the READ statement are used to read data base relations. READ statement formats are identical to those shown in figure 14-2 for CDCS 1. A relation can be read sequentially or randomly. The rules discussed in section 5 for reading a file also apply when reading a relation.

A relation is the means by which a set of files within the sub-schema are joined together. The files are joined on the basis of relationships that exist between data items in the respective files. For example, one file in a three-file relationship is joined to a second file through a data item contained in both files; the second file is then joined to a third file through the same or another data item that is contained in both the second and third files. The first file is the root file, which is the file at which the relation is entered. The files in the relation are traversed from the first file to the second file to the third file.

Relations are referenced only in the Procedure Division. SELECT and ASSIGN clauses for each of the data base files are optional and are required only when the program uses the FILE STATUS clause. Each file in the relation must be named in the sub-schema.

Relation-name is the name of a relation specified in the sub-schema. Format 3 reads a relation sequentially; format 4 reads a relation randomly. Data-name must reference a primary or alternate key in the root file. When the READ statement is executed, a record from each file in the relation is available to the COBOL program.

START Statement

Format 2 of the START statement (figure 14-5) positions the root file of the relation for subsequent sequential retrieval of records through relational reads. The files within the relation must be open in the INPUT or I-O mode. Execution of START establishes the current position of the root file as the beginning of the first record that has a key satisfying the START condition.

Format 2 of the START statement also establishes the key of reference for the root file. A READ relation-name NEXT following the format 2 START statement references the root file according to the key established by the statement. Refer to section 5, Procedure Division, for further discussion of the START statement.

```
Format 2

       OPEN  { INPUT relation-name [WITH NO REWIND] . . . } . . .
             { I-O relation-name                          }
```

Figure 14-4. OPEN Statement Format 2

```
Format 2

                                   [      ( IS EQUAL TO          )           ]
                                   [      | EQUALS               |           ]
                                   [      | IS =                 |           ]
       START relation-name         [ KEY  < EXCEEDS              > data-name ]
                                   [      | IS GREATER THAN      |           ]
                                   [      | IS >                 |           ]
                                   [      | IS NOT LESS THAN     |           ]
                                   [      ( IS NOT <             )           ]

           [; INVALID KEY imperative-statement]
```

Figure 14-5. START Statement Format 2

USE FOR ACCESS CONTROL Declarative Statement

The USE FOR ACCESS CONTROL statement (figure 14-6) identifies the access control key required to gain access to the specified files. It must be the first sentence after a section header within the Declaratives portion of the Procedure Division.

```
USE FOR ACCESS CONTROL

[     ( INPUT     ) ]
[ ON  ( I-O       ) ]
[     ( INPUT I-O ) ]
[     ( I-O INPUT ) ]

; KEY IS data-name

[ FOR { realm-name-1 [, realm-name-2] . . . } ] .
[     { REALMS                              } ]
```

Figure 14-6. USE FOR ACCESS CONTROL Statement Format

Within the schema, access control locks can be defined to provide privacy at the file level. The USE FOR ACCESS CONTROL statement must be used within the COBOL program to supply the appropriate key to gain access to a locked file.

Each USE FOR ACCESS CONTROL procedure is executed one time, at the start of program execution and before any data base file is opened.

Any given combination of access mode and file can be included in only one USE FOR ACCESS CONTROL statement.

The ON phrase specifies the type of access for which the key applies. INPUT indicates read access, I-O indicates update access, and INPUT I-O or I-O INPUT indicates read and update access. INPUT I-O and I-O INPUT are synonomous and interchangeable.

If the ON phrase is not specified, ON INPUT I-O is assumed.

The KEY phrase names a data item containing the access control key applicable to the specified access mode and files. The key value required to obtain a given access mode is specified in the ACCESS-CONTROL clause in the schema.

Data-name must be an alphanumeric item of from one to 30 characters.

When the USE FOR ACCESS CONTROL procedure completes execution, the key value contained in the KEY phrase is passed to the CDCS privacy module.

The FOR phrase specifies the realms, or files, for which the access control key applies. Any realm-name specified must also appear in the sub-schema. A realm in the sub-schema is equivalent to a file in the COBOL program.

If REALMS is specified in the FOR phrase, CDCS grants access to all data base files named in the sub-schema that have lock values equal to the access control key value.

If the FOR phrase is omitted, FOR REALMS is assumed.

USE FOR DEADLOCK Declarative Statement

The USE FOR DEADLOCK statement (figure 14-7) identifies the procedure to be executed when a deadlock situation occurs. It must be the first sentence after a section header within the Declaratives portion of the Procedure Division. The meaning of a deadlock situation is discussed in the following paragraphs.

```
USE FOR DEADLOCK

    ON { realm-name-1 [, realm-name-2] . . . }  .
       { REALMS                              }
```

Figure 14-7. USE FOR DEADLOCK Statement Format

In the statement, realm-name specifies the name of the realm, or file, for which the deadlock situation occurs. Any realm-name specified must also appear in the sub-schema. A realm in the sub-schema is equivalent to a file in the COBOL program.

If REALMS is specified, the USE FOR DEADLOCK procedure is executed when a deadlock situation occurs involving any data base file named in the sub-schema.

If the USE FOR DEADLOCK statement is omitted for a given file and a deadlock situation occurs for that file, the run is aborted. A deadlock situation occurs when two or more programs try to access a file or record that is in a locked state.

To protect the integrity of a data base while it is being accessed by two or more programs, CDCS provides a locking and unlocking mechanism. When a file or record has been locked, it is available for reading and updating purposes only to the program possessing the lock. Other programs can read the file or record, but cannot update it.

A file or record can be locked in one of two ways: either through an automatic function of CDCS or through an instruction in the COBOL program. Automatic locking of a record occurs during a normal read operation when the file involved has been opened with the I-O option specified in the OPEN statement. The record is unlocked when a subsequent input-output statement for that file is executed.

An entire file can be locked by specifying the following statement within the COBOL program:

```
ENTER "C.LOK" USING file-name.
```

The USING phrase indicates the name of the file that is to be locked. Once the file has been locked, other programs can access it through read, but not update, operations. The file must be successfully opened before it can be locked.

To unlock a file that has been locked, the following statement must be specified:

```
ENTER "C.UNLOK" USING file-name.
```

The USING phrase indicates the name of the file that is to be unlocked.

When a relation is read, CDCS locks all records in a given relation occurrence if the files joined in the relation have been opened for input-output processing. When the next relational read statement is executed, the locks for the previous relation occurrence are released.

Recovery Points

Within a COBOL program that accesses a sub-schema, recovery points can be specified to indicate convenient locations for restarting the program if recovery of the data base is necessary. To mark a point for recovery purposes, the following statement must be executed:

ENTER "DB$RPT" USING data-name-1, data-name-2.

In the USING phrase, data-name-1 must be a numeric integer data item with a usage of COMPUTATIONAL-1. Upon return from the routine DB$RPT, it will contain a unique recovery point number assigned by CDCS. Data-name-2 must be an alphanumeric item of 30 characters in length. It usually contains a user-supplied explanatory message for the recovery point. The informative message is written to the CDCS journal log file along with the recovery point number. For further details on data base recovery, consult the CDCS 2 reference manual.

COBOL SUBPROGRAMS

Data base files can be accessed by COBOL subprograms. The main program and the subprogram must include the SUB-SCHEMA clause. File-Control entries need not be specified in the main program or in the subprogram for data base files referenced in the run unit. See section 15, Inter-Program Communication Facility.

COMPILATION AND EXECUTION

Compilation and subsequent execution of a COBOL program are initiated by control statements. When a COBOL sub-schema is accessed by the program, certain requirements must be fulfilled before compiling or executing the program.

Before a program referencing a sub-schema can be compiled, the file on which the sub-schema resides must be attached. The D parameter on the compiler call must specify the logical file name of the sub-schema file.

When the compiled program is executed, CDCS monitors and interprets all requests for action on relations and on data base files, directing all operations involving the storage and retrieval of data to and from the file through CYBER Record Manager.

File attachment rules for program execution are as follows:

The file containing the schema can be attached, but is not required.

The file containing the sub-schema can be attached, but is not required.

The data base files and associated index files must not be attached.

The log file, master directory, and data base procedure library must not be attached. Exceptions exist for program execution with the CDCS Batch Test Facility. Log files and the master directory file must be attached. Data base procedure library must not be attached. All other rules apply. Refer to appendix F of the CDCS 2 reference manual.

DATA BASE FILE ERROR PROCESSING

Data base file and relation error processing can be handled in two ways:

By defining a status block with the DB$DBST routine

By entering the COMPASS object routine C.DMRST for CDCS relation operations, or C.IOST for CDCS non-relation operations

By calling the DB$DBST routine, the COBOL applicatons program can define a status block in memory which is automatically updated by CDCS with error status information. In order to provide compatibility with earlier versions of CDCS, the COBOL applications program can still call C.DMRST or C.IOST to determine the status of CDCS operations. Refer to the CDCS reference manual for further discussion of CDCS file status errors.

DB$DBST Routine

The COBOL application program can provide CDCS with a block in memory (a group item or an array) to which data base status information can be returned. The data base status block is updated automatically after every operation on a data base file or relation. The following information is returned to the data base status block:

CDCS or CYBER Record Manager error codes

The sub-schema item ordinal for item level error

CRM file position codes

The function that was being executed when an error occurred

The rank of the file in a relation for which an error, a control break, or a null record condition occurred

The name of the area in which an error occurred

The COBOL application program communicates the location and length of the data base status block to CDCS by a call to the CDCS routine, DB$DBST. The routine can be called at any point in a COBOL application program. Only one data base status block can exist at a time for a program. If DB$DBST is called more than once in a run unit, the data base status block defined in the last call is the one that is updated by CDCS.

Data base status block error codes and file position codes are returned to the user as decimal values. The user must convert to octal those values returned that are CRM codes, in order to correlate the code with those listed in the CRM reference manual. Error codes 384 through 447 indicate CDCS errors; these codes and corresponding message are listed in the CDCS reference manual.

The length of the data base status block is specified in words. One word provides space for either a 10-character DISPLAY item or a COMP-1 item. The length of the data base status block must be at least 1 word and should not be greater than 11 words. CDCS returns as much information as possible in the given length. The call that communicates the location and length of the data base status block is:

ENTER "DB$DBST" USING status-block, length

(The length must be specified by a data item defined as COMP-1; status-block indicates the 01 level data-name in the description of the data base status block.)

A possible COBOL description of the data base status block is shown in figure 14-8. Each elementary item of DATABASE-STATUS-BLOCK corresponds to a word of memory in which particular information is returned, except DB-REALM which corresponds to three words. The following list specifies each elementary item of the data base status block, indicates the corresponding word number for each item, and details the information returned in each item.

DATABASE-STATUS - Word 1
: The decimal value of the CRM or CDCS error code for the last data base operation on a file or relation. If no error has occurred, the value is zero. Note that only error codes are returned. If the last operation was a relation retrieval for which a null record or control break occurred, the value is zero; the status codes for these conditions are not returned in DATABASE-STATUS.

DB-ITEM-ORDINAL - Word 2
: Sub-schema item ordinal for CDCS item-level errors. Item-level errors include data validation errors, record mapping errors, and item-level data base procedure errors. The value is zero if no errors have occurred. The item ordinal is assigned by the DDL compiler and is identified on the sub-schema listing.

DB-FILE-POSITION - Word 3
: CRM code indicating the file position of the realm when the last data base operation was performed. For a relation operation, the code indicates the position of the root realm. A file position code is returned when a read, open, close, or start operation is performed. The codes most commonly returned during data base operations are shown in table 14-1. Further information on file position (the FP field of the file information table) is in the CYBER Record Manager Advanced Access Methods reference manual.

TABLE 14-1. FILE POSITION CODES

Decimal Codes	Definition
8	End-of-key list, which occurs when the last primary key value associated with a given alternate key has been returned during a read operation using an alternate key value.
16	End-of-record, which occurs when a record has been returned during a read operation.
64	End-of-information, which occurs when a sequential read operation is attempted after the previous read operation returned the last record in the file.

FILLER - Word 4
: Reserved space. This space must be specified as either FILLER consisting of 10 characters or a COMP-1 item.

DB-FUNCTION - Word 5
: The function being performed when an error occurred. One of the following display code strings is placed in this word to indicate the function:

UNDEFINED	OPEN
RAN-READ	CLOSE
SEQ-READ	START
WRITE	RECVR-PT
REWRITE	TIME
DELETE	PRIVACY
REL-READ	LOCK
REL-NEXT	UNLOCK
REL-STRT	END

```
          01  DATABASE-STATUS-BLOCK.
              02  DATABASE-STATUS                  PICTURE 9(5) USAGE IS COMP-1.
              02  AUXILIARY-STATUS.
                  03  DB-ITEM-ORDINAL              PICTURE 9(5) USAGE IS COMP-1.
                  03  DB-FILE-POSITION             PICTURE 9(3) USAGE IS COMP-1.
                  03  FILLER                       PICTURE X(10).
              02  DB-FUNCTION                      PICTURE A(10).
              02  RELATION-RANK-STATUS.
                  03  DB-REL-RANK-ERROR            PICTURE 9(3) USAGE IS COMP-1.
                  03  DB-REL-RANK-CTLBK            PICTURE 9(3) USAGE IS COMP-1.
                  03  DB-REL-RANK-NULL             PICTURE 9(3) USAGE IS COMP-1.
              02  DB-REALM                         PICTURE X(30).
```

Figure 14-8. Data Base Status Block Description

DB-REL-RANK-ERROR - Word 6

For a relation operation, the rank of the file on which a CRM or CDCS error occurred. The root file has a rank of one. Zero is returned if there is no error. The CRM or CDCS error code is returned in the item DATABASE-STATUS.

An error on a file during a relation read terminates the operation. Consequently, there is never more than one rank in the relation that has a CRM or CDCS error.

DB-REL-RANK-CTLBK - Word 7

For a relation operation, the lowest numbered rank on which a control break occurred. The value is zero if no control break exists. The root file of the relation has a rank of one. All files in the relation with a rank greater than the rank stored in this word have control break status or null status. Null status overrides control break status.

DB-REL-RANK-NULL - Word 8

For a relation operation, the lowest numbered rank for which there is a null record. If zero, there is no null record. The root file of the relation has a rank of one. All files in the relation with a rank greater than the rank stored in DB-REL-RANK-NULL also have null record occurrences.

DB-REALM - Words 9, 10, and 11

The display code name of the realm or area in which an error occurred.

The length of the data base status block is variable; the length can range from 1 to 11 words. As a minimum, one word must be provided for the error code (for example, the item DATABASE-STATUS shown in figure 14-8). The other items are optional; however, any specification of the data base status block must adhere to the following list of the rules (the rules reference the description of the data base status block shown in figure 14-8):

The items must be defined with the length shown and in the order shown.

For any particular item that is defined, all items that precede it must also be defined.

COMP-1 must be used to define the items as indicated.

For information to be returned to any elementary item of the group items AUXILIARY-STATUS or RELATION-RANK-STATUS, length must be provided for all three items of the particular group.

For example, if the user wants the data base status block to include only the items DATABASE-STATUS and DB-ITEM-ORDINAL, the user must specify a length of four words.

C.DMRST Routine

The status of data base files within a relation can be obtained by entering the COMPASS object routine C.DMRST. This routine scans the files in the order the files are traversed, beginning with the root file and continuing until a nonzero error code is encountered or all files have been scanned. For use of this routine, C.DMRST should be entered after a READ statement is executed for a relation in order to determine whether an error occurred during the read operation and on which file the error occurred. To learn the status of files after a relational read, the following statement must be executed:

```
    ENTER "C.DMRST" USING relation-name,
        data-name-1, data-name-2.
```

The USING phrase specifies the name of the relation to be checked for file status and two data items for returned information. Data-name-1 must be an elementary item described as PICTURE X(7). Data-name-2 must be a numeric integer data item with a size greater than two and a usage of COMPUTATIONAL-1.

If an error, a null condition, or a control break occurs in a file, the implementor-name (logical file name) of the file is returned in data-name-1; the CDCS status or error code is returned in data-name-2. The statement can be executed again to check for errors or relation conditions on the remaining files in the relation. If no errors, null records, or control breaks occur on the files in the relation, zero is returned to data-name-2; the contents of data-name-1 are undefined.

The use of the DB$DBST routine provides the user with more relevant information about relation processing than the use of the C.DMRST routine can provide.

PROGRAM DEBUGGING

The automatic generation by COBOL of SELECT statements for sub-schema files poses special debugging problems. The logical file name from the sub-schema FIT is used for implementor-name-1 and implementor-name-2 in the generated SELECT statements. This logical file name must not appear as a user-defined word elsewhere in the COBOL program or sub-schema description. If it does, the fatal errors 4136, 4102, and 3179 are generated.

Errors occurring on the code for the generated SELECT statements have line and column numbers pointing to the sub-schema name in the SUB-SCHEMA clause.

INTER-PROGRAM COMMUNICATION FACILITY

Communication between separately compiled programs is provided through the Inter-Program Communication facility. This facility allows a COBOL main program to transfer control to a subprogram written in COBOL, COMPASS, or FORTRAN Extended. The CALL statement is used to transfer control to a COBOL subprogram. Control is transferred to a non-COBOL subprogram through the ENTER statement.

The same data can be accessed by both programs. A variety of ways are provided for specifying data to be passed between programs. Shared data can be specified through the Common-Storage Section or through parameters in the CALL or ENTER statement; files or means external to COBOL can also be used for sharing data.

DATA DIVISION CODING

Data described in the Common-Storage Section of a COBOL main program can be shared with any subprogram. A COBOL subprogram can describe data to be passed between programs in the Common-Storage Section or the Linkage Section.

COMMON-STORAGE SECTION

The Common-Storage Section describes data that is shared by independently compiled subprograms. If the Common-Storage Section is specified in a main program, it must appear in each COBOL subprogram that shares the data. Data described in this section is allocated to a labeled common block with the name CCOMMON. The Common-Storage Section is not presently in the ANSI standard; however, it can be used to provide compatibility with future standards.

The Common-Storage Section is also used for External files. If any data items are specified in the File-Control entry or FD entry for the file, those data items must be described in this section of the Data Division. Refer to the discussion on External files for further detail.

The most efficient method of passing data is by means of the Common-Storage Section. It allows direct references to the same data in all programs that specify the same Common-Storage Section. Programs in COMPASS or FORTRAN Extended can refer to the same data by referencing the common block CCOMMON. All COBOL programs sharing the data should use the COPY statement or Update common decks for describing the data to ensure that all descriptions are the same. Refer to the Update reference manual for further information.

Descriptions of data in different subprograms need not be identical, but the data must be identical. For instance, a table described item-by-item in one subprogram can be described with an OCCURS clause in another subprogram, as long as the total size of the table is the same.

The Common-Storage Section is composed of the section header followed by any number of level 77, 66, or 88 Data Description entries or Record Description entries of levels 01 thrugh 49. The section header is:

<u>COMMON-STORAGE SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

The VALUE clause can be used to set the initial value of an item in the Common-Storage Section only in a COBOL main program. If an initial value is not specified for a data item, its initial content might be unpredictable. If the VALUE clause is used with an item in the Common-Storage Section of a COBOL subprogram, the clause is ignored and a warning diagnostic is issued.

LINKAGE SECTION

The Linkage Section is valid only in a called subprogram. It describes data that is passed between the calling program and the called program through the USING phrases in the CALL statement of the calling program and in the Procedure Division header of the called program. No space is allocated in the subprogram for the data described in this section; rather, the calling program item is used during execution of the called program.

The Linkage Section is composed of a section header followed by Data Description entries. The section header is:

<u>LINKAGE SECTION.</u>

The header must appear on a separate line, beginning in area A, and must be terminated by a separator period.

Level 77, 66, and 88 items and Record Description entries can be specified in the Linkage Section. Level 77 items must contain either a PICTURE clause, a USAGE IS INDEX clause, or a USAGE IS COMP-2 clause. Other Data Description entry clauses are optional. An initial value cannot be assigned in the Linkage Section. Any VALUE clause is allowed, but its meaning is ignored and a trivial diagnostic is issued.

Data items defined in the Linkage Section can be referenced within the Procedure Division only if the items are specified in the USING phrase of the Procedure Division header or are subordinate to items so specified. Results are undefined for subprogram references to a Linkage Section item that is not named in a CALL statement in the calling program.

Frequent reference to Linkage Section items creates a significant amount of overhead. If a Linkage Section item is referenced often, it should be moved to a Working-Storage Section item within the called subprogram.

SHARED FILES

Files can be shared between programs in the same run unit. A file can be written on in one program and read in another program; however, this method should be avoided because the file must be closed before it can be opened again and processed in another program. A file that is declared an External file is shared between the main program and all subprograms that reference it. Once an External file is opened, it need not be closed and reopened for processing in another program. Data base files can also be shared between programs.

EXTERNAL FILES

An External file is identified by the EXTERNAL clause in the FD entry for the file. It is shared by all programs in the run unit that describe the file. File information exists external to the programs. All External files must be declared in the main program.

A file is declared an External file by specifying EXTERNAL in the FD entry. See EXTERNAL Clause in section 4. Any subprogram referencing the External file must describe the file exactly as it is described in the main program. The COPY statement or Update common decks should be used to ensure identical descriptions.

The record area for an External file is shared between programs in the same manner as Common-Storage Section items are shared. Data items specified in the File-Control entry and FD entry for an External file must be defined in the Common-Storage Section of the Data Division. For example, data-names specified in clauses such as the RECORD KEY clause (if data-name is not in a Record Description entry for the file), the BLOCK COUNT clause, and the RECORD clause must be defined in the Common-Storage Section. The Record Description entries of level 01 through 49 must not be defined in the Common-Storage Section and must follow the FD entry.

DATA BASE FILES

Data base files, which are accessed through the CYBER Database Control System (CDCS), can be shared between programs. The FDL parameter must be specified on the compiler call statement if either the Program Equivalence section or the Database Usage section is used. The Database Usage section is used for CDCS 1 only. Each data base area (file) that is referenced by a program must be specified in the sub-schema. The main program and any subprogram using the data base must include the SUB-SCHEMA clause in the SPECIAL-NAMES paragraph of the Environment Division.

When CDCS 1 data base files are used by subprograms, an FDL file must be made available to the run unit. The FDL file is described in the paragraphs under the heading Fast Dynamic Loader Processing in this section. See section 14, Sub-Schema Facility, for additional information on data base files.

FAST DYNAMIC LOADER PROCESSING

Fast Dynamic Loader processing and CDCS subprogram usage (for CDCS 1 only) is made available by specifying the FDL parameter on the compiler call. This parameter indicates the FDL file, which contains information related to COBOL subprogram names and data base usage. The Program Equivalence section header can also be used to equate a long program name with a shorter program name (less than seven characters) regardless of whether or not subprograms are involved.

The FDL file consists of card images with one statement per card image. Each statement begins in column 1; separators are as defined for COBOL. Only the first 72 columns of each statement are examined. A statement can, but need not, be terminated by a separator period. A comment can follow a statement on the same line if the statement is terminated by a period. The FDL file contains two sections: Program Equivalence and Database Usage.

PROGRAM EQUIVALENCE SECTION

The Program Equivalence section contains a series of statements that equate program names with internal system names and indicate the programs to be statically or dynamically loaded. A static program is loaded with the base module. A dynamic program is loaded at the time the first CALL statement referencing the program is executed and is unloaded with CANCEL.

The Program Equivalence section is composed of a section header followed by any number of program equivalence statements. The section header is:

<u>PROGRAM EQUIVALENCE</u>

The header must appear as a separate card image, beginning in column 1, and can be terminated by a separator period.

All program equivalence statements follow the header. The format of a statement is:

program-name = internal-name. [<u>STATIC</u>]

Program-name is the name used in a CALL statement and CANCEL statement, and in the PROGRAM-ID paragraph of the subprogram; it can be up to 30 characters in length. Internal-name defines the name to be used internally by the system. It must be seven characters or less and must be unique within the run unit. Internal-name is used as the entry point and program name (loader name) for any program with a matching PROGRAM-ID and as the external symbol for the matching CALL statement and CANCEL statement.

Static subprograms need not be specified in the Program Equivalence section. If the section is used, STATIC indicates that the subprogram is to be loaded with the base module. If STATIC is not specified, the program is to be dynamically loaded using FDL processing. All dynamic programs must be specified in the Program Equivalence section of the FDL file. Initially all programs in the run unit must be compiled in a single stacked compilation in the following order:

Main program
Static subprograms
Dynamic subprograms

The FDL parameter must be specified in the COBOL5 control statement for the compilation when the Program Equivalence section is used. In a later run unit, a subprogram can be compiled using the SB parameter and the same FDL file used for the initial compilation. If CALL identifier or any equivalence names are used in a subprogram, the original FDL file must be used in compilation of the subprogram. The subprogram can then be replaced on the initial binary file using the COPYL utility.

During the compilation of a COBOL 5 main program and dynamic subprograms, overlay capsules are generated for the dynamic subprograms. The compiler also creates relocatable binaries for each program. The binaries must in turn be made into an absolute load file before execution can begin. The load file is the default file CB5CODE.

Once the absolute load file has been created, the main program and its overlay capsules must be maintained as a unit. It can be executed from a standard sequential file or it can be placed on a user library.

Figure 15-1 shows an example of a job deck that compiles and executes a COBOL 5 main program and subprogram using FDL processing. An absolute version of the application program is captured. The Program Equivalence section, in which the program-name SUBPROGRAM is equated with the internal-name SUBPROG, is copied onto the FDL file with execution of the COPYBR statement. The load sequence generated by the LOAD(LGO) and NOGO statements produces an absolute load file from the relocatable binaries on the default file CB5CODE. Execution of the main program begins when the control statement CB5CODE is encountered and processed. The permanent file statements (DEFINE, REQUEST, and CATALOG) are used optionally to create a permanent file of the main program and subprogram. The LDSET(NOEPT) statement might be needed under the NOS/BE system, in order to avoid a job abort with duplicate entry points on a later EDITLIB.

For execution only, the LOAD(LGO), NOGO, and CB5CODE statements can be replaced by the LGO statement in the example shown in figure 15-1.

For additional information on the Fast Dynamic Loader facility, see the CYBER Loader reference manual.

Dynamic loading should be used with care because it requires a significant amount of time. In addition, numerous calls and cancels can cause core fragmentation problems. In general, heavily used programs should be statically loaded.

DATABASE USAGE SECTION (CDCS 1 ONLY)

The Database Usage section contains statements that identify the areas to be used by subprograms and the operations to be performed on the specified areas. In addition, statements are included in this section to indicate the data base relations that can be read. See section 14 for a discussion of relations.

The Database Usage section is composed of a section header followed by any number of database usage statements. The section header is:

<u>DATABASE USAGE</u>

The header must appear as a separate card image, beginning in column 1, and can be terminated by a separator period.

Two types of statements follow the Database Usage section header. One type designates the data base areas to be used and the operations to be performed on each area. The format of this statement is:

<u>AREA</u> area-name <u>USES</u> operation [operation] . . .

Area-name is the name of a data base area that is used by a program in the run unit. One statement must be included for each area referenced by any program in the run unit; all operations to be used must be specified. Operation is one of the following keywords:

CLOSE	READ	WRITE
DELETE	REWRITE	
OPEN	START	

<u>NOS/BE Operating System</u>

```
FDLRUN,TIIO
COPYBR(INPUT,FDLFILE)
REWIND(FDLFILE)
COBOL5(FDL)
REQUEST(CB5CODE,*PF)
LOAD(LGO)
NOGO.
CATALOG(CB5CODE,ID=MINE)
CB5CODE.
7/8/9
PROGRAM EQUIVALENCE.
SUBPROGRAM = SUBPROG
.
.
.
7/8/9
    IDENTIFICATION DIVISION.
    PROGRAM-ID. MAIN.
    .
    .
    .
    IDENTIFICATION DIVISION.
    PROGRAM-ID. SUBPROGRAM.
    .
    .
    .
7/8/9
6/7/8/9
```

<u>NOS Operating System</u>

```
FDLRUN.
COPYPBR(INPUT,FDLFILE)
REWIND(FDLFILE)
DEFINE(CB5CODE)
COBOL5(FDL)
LOAD(LGO)
NOGO.
CB5CODE.
7/8/9
PROGRAM EQUIVALENCE.
SUBPROGRAM=SUBPROG
.
.
.
7/8/9
    IDENTIFICATION DIVISION.
    PROGRAM-ID. MAIN.
    .
    .
    .
    IDENTIFICATION DIVISION.
    PROGRAM-ID. SUBPROGRAM.
    .
    .
    .
7/8/9
6/7/8/9
```

Figure 15-1. Example of Fast Dynamic Loader Usage

Any of these operations that are performed by subprograms within the run unit must be specified. If a program references an area or an operation that is not specified in a database usage statement, a diagnostic is issued and the run is aborted.

When a relation is referenced by a program, each area joined in the relation must be specified in an AREA statement. The CLOSE and OPEN operations must be specified because the areas have to be opened and closed by the program.

The second type of database usage statement identifies a relation that can be read by any subprogram in the run unit. The format of this statement is:

RELATION relation-name

Relation-name is the name of the relation as specified in the sub-schema. One statement must be included for each relation referenced by the programs in the run unit.

PROCEDURE DIVISION STATEMENTS

The CALL statement and the ENTER statement transfer control to a subprogram. The EXIT PROGRAM statement is used within a COBOL subprogram to return control to the calling program. The CANCEL statement removes a called program from memory. In addition to these statements, the Procedure Division header is used to identify shared data in a subprogram executing under control of a CALL statement that includes the USING phrase.

PROCEDURE DIVISION HEADER

The Procedure Division header in a COBOL subprogram includes the USING phrase when the program is called by a CALL statement that also specifies the USING phrase. The format of the header is:

```
PROCEDURE DIVISION USING data-name-1

     [, data-name-2 ...].
```

The order of the operands in the header must be the same order as the operands in the CALL statement. A data-name must not appear more than once in the division header, although it can appear more than once in the CALL statement.

Each of the operands in the USING phrase of the Procedure Division header must be defined as a level 01 or level 77 data item in the Linkage Section of the subprogram. Within the called program, Linkage Section data items are processed according to their descriptions in the called program.

CALL STATEMENT

The CALL statement (figure 15-2) transfers control to a COBOL subprogram. (Use the ENTER statement to transfer control to subprograms that are not written in COBOL. If the CALL statement is used for a FORTRAN subprogram, mode errors could occur.)

```
CALL { identifier }
     { literal    }

   [USING data-name-1   [, data-name-2] . . .]

   [; ON OVERFLOW imperative-statement]
```

Figure 15-2. CALL Statement Format

The program in which the CALL statement appears is the calling program; the program referenced in the CALL statement is the called program. Called programs can contain CALL statements; however, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

When a CALL statement appears in a section with a segment-number 50 through 99, the segment is in its last used state when the EXIT PROGRAM statement in the called program returns control to the calling program. See section 8, Segmentation Facility, for a discussion of called programs in overlayable segments.

On the first entry into a called program, or on the first entry after a CANCEL statement specifying that program, a dynamic program is loaded in its initial state. All data items are set to their initial values as specified by VALUE clauses, and all altered GO TO statements are cleared.

On all subsequent entries into the called program, the state of the program remains unchanged from its state when last exited. All data items and all alterable switch settings exist in the called program as they were immediately before the last EXIT PROGRAM execution in the called program. In addition, files local to the called program are in their last used state. Any External files might have been changed by other programs.

In the CALL statement, literal or identifier specifies the subprogram to be called. The value of literal or the contents of the data item referenced by identifier must correspond to the program-name specified in the PROGRAM-ID paragraph of the called program. Literal must be a nonnumeric literal; identifier must be an alphanumeric data item.

The FDL parameter must be specified on the compiler call if identifier is specified or if literal is longer than seven characters. Upon execution of the CALL statement, a literal that is longer than seven characters or the contents of identifier must correspond to one of the program-names in the FDL file. When the FDL file is used, the value of literal or the contents of identifier must be a program-name of less than 31 characters, excluding trailing spaces. If literal is less than eight characters in length and is not in the FDL file, it is assumed to be the same as the entry point of the called program and is static.

If the FDL parameter is not specified, only literal can be specified and it must be less than eight characters in length. There is significant overhead when the FDL file is used; therefore, it should be used only where necessary.

See section 11, Product Interfaces, for the calling sequence generated by the CALL statement.

USING Phrase

The USING phrase passes parameters to the called program. Each data-name in the USING phrase must be:

Defined in the calling program as a level 01 or level 77 item in the File Section, Working-Storage Section, or Linkage Section.

Specified in the called program in the USING phrase of the Procedure Division header.

Defined in the called program as a level 01 or level 77 item in the Linkage Section.

The number of data-names in the USING phrases of the CALL statement and the Procedure Division header must be identical. The order of data-names is critical: both lists refer to a single set of data with correspondence by position, not name.

Corresponding pairs of data-names should be defined at the same level because a level 01 item and a level 77 item are synchronized differently. If items of different levels are in corresponding positions, the ANSI=77LEFT parameter must be specified on the compiler call to ensure left synchronization of the level 77 items.

Index-names cannot be used in the parameter list since index-name in the called and calling programs always refer to separate indexes.

Passing parameters in the CALL statement is the most common method of sharing data. A significant amount of overhead, both in memory requirements and in time, is associated with this method. All references to Linkage Section items have to be set to the real address each time the CALL statement is executed. The more parameters that are passed and the more frequently a subprogram is called, the more system resources are used.

ON OVERFLOW Phrase

The ON OVERFLOW phrase specifies the imperative-statement that is executed if a dynamic program is called and there is insufficient room to load it.

This condition might occur if either the maximum field length for the job or the field length specified in the CM parameter in the job statement is reached. If an imperative-statement is not specified and this condition occurs, the run is aborted.

CANCEL STATEMENT

The CANCEL statement (figure 15-3) unloads a subprogram, thereby releasing the memory space occupied by the named program. When a subsequent CALL statement specifying the same program is executed, the program is reloaded into memory in its initial state.

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

Figure 15-3. CANCEL Statement Format

In the CANCEL statement, identifier-1 and identifier-2 must each be an alphanumeric data item; literal-1 and literal-2 must each be a nonnumeric literal.

A program named in the CANCEL statement must be a dynamic program and it must have executed an EXIT PROGRAM statement after it was called. If the program specified in the CANCEL statement has not been called or has been called and already cancelled, no action takes place and execution continues with the next statement.

ENTER STATEMENT

The ENTER statement (figure 15-4) transfers control to a subprogram written in the COMPASS assembly language, the FORTRAN 5 compiler language, or the FORTRAN Extended 4 compiler language. The ENTER statement should be used only with COMPASS and FORTRAN; the CALL statement should be used only with COBOL.

When both COMPASS and FORTRAN-X are omitted from the statement, COMPASS is assumed. Routine-name must be an entry point into the subprogram. When routine-name contains characters other than letters or digits, or when routine-name is the same as a COBOL reserved word, routine-name must be specified as a nonnumeric literal.

When the calling program is segmented, the subprograms must reside in a user library. See section 8, Segmentation Facility.

See section 11, Product Interfaces, for the calling sequence generated by the ENTER statement.

USING Phrase

The USING phrase specifies parameters to be passed to the subprogram. The parameter list can include data-names with qualifiers or literal subscripts, file-names, procedure-names, and literals; the specific parameters required depend on the needs of the subprogram being called. The word ALL in a nonnumeric literal is ignored. The compiler does not check for errors in the parameter list.

```
ENTER [COMPASS  ] routine-name [USING {data-name-1     } [, data-name-2     ]     ]
      [FORTRAN-X]              [      {file-name-1     } [, file-name-2     ]     ]
                               [      {procedure-name-1} [, procedure-name-2]     ]
                               [      {literal-1       } [, literal-2       ] ... ]
```

Figure 15-4. ENTER Statement Format

Utility Routines

Several utility routines are available and can be accessed by the ENTER statement with USING parameters. The routines are written in the COMPASS assembly language. Utility-name must be enclosed in quotes. Each routine is shown with its appropriate USING parameters.

C.CMMMV USING data-name-1, data-name-2, data-name-3

Moves data from one location to another. The contents of the data names specified determine the move that is performed. The possible moves are:

To a CMM block from the Data Division

To the Data Division from a CMM block

From one CMM block to another

Data-name-1 specifies the sending location. If data-name-1 is a COMP-1 item, it is assumed to contain the address of the sending item (usually a CMM block). If data-name-1 is not a COMP-1 item, it is assumed to contain the data being sent. Data-name-2 specifies the receiving location. The interpretation of its contents is the same as for data-name-1 except that it indicates the receiving field. Data-name-3, which must be a COMP-1 item, specifies either the number of characters to move or a value to be used as a subscript for the CMM block.

When data-name-1 and data-name-2 are both COMP-1 items (therefore indicating addresses of CMM blocks), data-name-3 is the number of characters to move from one block to another.

When either data-name-1 or data-name-2 is not a COMP-1 item (one of these items must be a COMP-1 item), the CMM block is treated as if it were a table with the length of the occurring item being the length of the non-COMP-1 item. The item that is the COMP-1 item indicates the beginning address of the CMM block. In this situation, data-name-3 acts as a subscript for the table (the CMM block) and locates the sending or receiving field.

No checking is done on any parameter; the user is responsible for the validity of the parameters.

C.DMRST USING relation-name, data-name-1, data-name-2

Checks the status of files traversed by a relational read operation. See section 14, Sub-Schema Facility, for a discussion of this routine.

C.DSPDN USING data-name-1 [, data-name-2]...

Prints the line number of the line that contains the ENTER statement and the contents of the item data-name-1 and data-name-2 (if specified). The line number is printed in the form $$$L=nnnn and is followed by two spaces, the contents of data-name-1 printed in octal and, if specified, the contents of data-name-2 preceded by a comma. Below this line is printed a line that translates the octal characters to display code with each display code character being printed under the appropriate octal characters. Multiple lines are printed if the contents of the specified data names exceed 120 characters.

C.DTCMP USING data-name-1

Returns the date the program was compiled in a form that depends on the operating system. The date is represented by eight characters with the slash characters included in the third and sixth positions. The standard default for the NOS/BE operating system is MM/DD/YY. The standard default for the NOS operating system is YY/MM/DD. Data-name-1 must be an alphanumeric data item of eight or more characters in length. Any field larger than eight characters is filled on the right with spaces.

C.FILE USING file-name-1, data-name-1

Allows specification of several parameters normally specified through the USE clause or the FILE control statement. The parameters that can be included are:

BBH
CF
ERL
TRC

A complete description of the meaning and use of these parameters appears in the CYBER Record Manager manuals.

File-name-1 is the file-name of a file described in a SELECT clause in the Environment Division. Data-name-1 specifies an alphanumeric item that contains the parameters or a nonnumeric literal that duplicates the parameter format of the FILE control statement.

C.GETEP USING data-name-1

Returns the parameters specified on the execution call control statement in the item indicated by data-name-1. The calling directive and all parameters specified with it are returned to the item. Only one card image can be returned, however; continuation statements are not processed. The data item indicated by data-name-1 must be an alphanumeric item and should be 80 characters in length. If the item is larger than 80 characters, it is blank filled. If it is smaller, the card image is truncated on the right. The UNSTRING statement can be used to separate the parameters for processing.

C.IOST USING file-name-1, data-name-1, data-name-2

Provides the CYBER Record Manager or CDCS error code and the severity of the error. See the discussion of the FILE STATUS clause in section 3 for a description of this routine.

C.LOK USING file-name-1

Issues the CDCS LOCK call for a file. See section 14, Sub-Schema Facility, for a description of this routine.

C.SEEK USING file-name-1

Performs a SEEK operation on the file designated by file-name-1. File-name-1 must be the COBOL file-name of an indexed, direct, or actual-key file. A SEEK operation sometimes provides an overlap between processing and input-output.

C.SORTP USING data-name-1, data-name-2

Regulates the size of the central memory block allocated for sort and merge operations. See section 7, Sort/Merge Facility, for a description of this routine.

C.UNLOK USING file-name-1

Issues the CDCS UNLOK call. See section 14, Sub-Schema Facility, for a discussion of this routine.

C.IOENA

Called to override abort run after error declarative. (See section 5, Format 1 USE.)

EXIT PROGRAM STATEMENT

The EXIT PROGRAM statement returns control from a called program to the calling program; it does not return control to the operating system. The format of the statement is:

```
EXIT PROGRAM.
```

This statement can only appear in a called subprogram. (In a main program, EXIT PROGRAM causes a diagnostic to be issued and execution to continue with the next sentence.) For ANSI usage, EXIT PROGRAM, which transfers control back to the calling program, must be the only statement in a sentence and the only sentence in a paragraph; however, in this implementation it can appear anywhere in a paragraph.

STANDARD CHARACTER SETS A

Control Data operating systems offer the following variations of a basic character set (as shown in table A-1):

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The installation can specify the 63-character set option in either CDC or ASCII mode (see tables A-2 and A-3). In either case, the colon becomes 63_8 and collates properly. In the ALPHABET clause, the mnemonic-names CDC-64 and ASCII-64 refer to CDC-63 and ASCII-63 character sets, respectively.

The set in use at a particular installation is specified when the operating system is installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

The EBCDIC 64-character subset (table A-4) is used for tape or card processing through the CODE-SET clause. (See section 4.)

The UNIVAC 1100 Series character set (table A-5) can be used for tape processing through the CODE-SET clause. (See section 4.)

TABLE A-1. COBOL AND STANDARD CHARACTER SETS

COBOL	Display Code (octal)	CDC			ASCII		
		Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
	00†	: (colon)††	8-2	00	: (colon)††	8-2	072
A	01	A	12-1	61	A	12-1	101
B	02	B	12-2	62	B	12-2	102
C	03	C	12-3	63	C	12-3	103
D	04	D	12-4	64	D	12-4	104
E	05	E	12-5	65	E	12-5	105
F	06	F	12-6	66	F	12-6	106
G	07	G	12-7	67	G	12-7	107
H	10	H	12-8	70	H	12-8	110
I	11	I	12-9	71	I	12-9	111
J	12	J	11-1	41	J	11-1	112
K	13	K	11-2	42	K	11-2	113
L	14	L	11-3	43	L	11-3	114
M	15	M	11-4	44	M	11-4	115
N	16	N	11-5	45	N	11-5	116
O	17	O	11-6	46	O	11-6	117
P	20	P	11-7	47	P	11-7	120
Q	21	Q	11-8	50	Q	11-8	121
R	22	R	11-9	51	R	11-9	122
S	23	S	0-2	22	S	0-2	123
T	24	T	0-3	23	T	0-3	124
U	25	U	0-4	24	U	0-4	125
V	26	V	0-5	25	V	0-5	126
W	27	W	0-6	26	W	0-6	127
X	30	X	0-7	27	X	0-7	130
Y	31	Y	0-8	30	Y	0-8	131
Z	32	Z	0-9	31	Z	0-9	132
0	33	0	0	12	0	0	060
1	34	1	1	01	1	1	061
2	35	2	2	02	2	2	062
3	36	3	3	03	3	3	063
4	37	4	4	04	4	4	064
5	40	5	5	05	5	5	065
6	41	6	6	06	6	6	066
7	42	7	7	07	7	7	067
8	43	8	8	10	8	8	070
9	44	9	9	11	9	9	071
+	45	+	12	60	+	12-8-6	053
-	46	-	11	40	-	11	055
*	47	*	11-8-4	54	*	11-8-4	052
/	50	/	0-1	21	/	0-1	057
(	51	(	0-8-4	34	(	12-8-5	050
)	52	)	12-8-4	74	)	11-8-5	051
$	53	$	11-8-3	53	$	11-8-3	044
=	54	=	8-3	13	=	8-6	075
blank	55	blank	no punch	20	blank	no punch	040
, (comma)	56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
. (period)	57	. (period)	12-8-3	73	. (period)	12-8-3	056
	60	≡	0-8-6	36	#	8-3	043
	61	[	8-7	17	[	12-8-2	133
	62	]	0-8-2	32	]	11-8-2	135
	63	%††	8-6	16	%††	0-8-4	045
" (quote)	64	≠	8-4	14	" (quote)	8-7	042
	65	↱	0-8-5	35	_ (underline)	0-8-5	137
	66	∨	11-0	52	!	12-8-7	041
	67	∧	0-8-7	37	&	12	046
	70	↑	11-8-5	55	' (apostrophe)	8-5	047
	71	↓	11-8-6	56	?	0-8-7	077
<	72	<	12-0	72	<	12-8-4	074
>	73	>	11-8-7	57	>	0-8-6	076
	74	≤	8-5	15	@	8-4	100
	75	≥	12-8-5	75	\	0-8-2	134
	76	¬	12-8-6	76	¬ (circumflex)	11-8-7	136
; (semicolon)	77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end-of-record mark rather than two colons.
††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD	Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	≤	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	∨	66	52
03	03	[	61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	∧	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	≥	75	75	42	52	Q	21	50
11	13	¬	76	76	43	53	R	22	51
12	14	.	57	73	44	54	]	62	32
13	15	)	52	74	45	55	S	23	22
14	16	;	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	-	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(	51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(	51	28	40	50	H	10	48
09	11	)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[	61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75	]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

TABLE A-4. EBCDIC 64-CHARACTER SUBSET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		Graphic	EBCDIC Punch	Display Code	EBCDIC Code
00	00	blank	no punch	55	40
01	01	.	12-8-3	57	4B
02	02	<	12-8-4	72	4C
03	03	(	12-8-5	51	4D
04	04	+	12-8-6	45	4E
05	05	|	12-8-7	66	4F
06	06	&	12	67	50
07	07	$	11-8-3	53	5B
08	10	*	11-8-4	47	5C
09	11	)	11-8-5	52	5D
10	12	;	11-8-6	77	5E
11	13	¬	11-8-7	76	5F
12	14	-	11	46	60
13	15	/	0-1	50	61
14	16	,	0-8-3	56	6B
15	17	%	0-8-4	63	6C
16	20	—	0-8-5	65	6D
17	21	>	0-8-6	73	6E
18	22	?	0-8-7	71	6F
19	23	:	8-2	00	7A
20	24	#	8-3	60	7B
21	25	@	8-4	74	7C
22	26	'	8-5	70	7D
23	27	=	8-6	54	7E
24	30	"	8-7	64	7F
25	31	¢	12-8-2/12-0	61	4A
26	32	A	12-1	01	C1
27	33	B	12-2	02	C2
28	34	C	12-3	03	C3
29	35	D	12-4	04	C4
30	36	E	12-5	05	C5
31	37	F	12-6	06	C6

TABLE A-4. EBCDIC 64-CHARACTER SUBSET COLLATING SEQUENCE (Contd)

Collating Sequence Decimal/Octal		Graphic	EBCDIC Punch	Display Code	EBCDIC Code
32	40	G	12-7	07	C7
33	41	H	12-8	10	C8
34	42	I	12-9	11	C9
35	43	!	11-8-2/11-0	62	5A
36	44	J	11-1	12	D1
37	45	K	11-2	13	D2
38	46	L	11-3	14	D3
39	47	M	11-4	15	D4
40	50	N	11-5	16	D5
41	51	O	11-6	17	D6
42	52	P	11-7	20	D7
43	53	Q	11-8	21	D8
44	54	R	11-9	22	D9
45	55	none	0-8-2	75	E0
46	56	S	0-2	23	E2
47	57	T	0-3	24	E3
48	60	U	0-4	25	E4
49	61	V	0-5	26	E5
50	62	W	0-6	27	E6
51	63	X	0-7	30	E7
52	64	Y	0-8	31	E8
53	65	Z	0-9	32	E9
54	66	0	0	33	F0
55	67	1	1	34	F1
56	70	2	2	35	F2
57	71	3	3	36	F3
58	72	4	4	37	F4
59	73	5	5	40	F5
60	74	6	6	41	F6
61	75	7	7	42	F7
62	76	8	8	43	F8
63	77	9	9	44	F9

TABLE A-5. UNIVAC 1100 SERIES COLLATING SEQUENCE [UNI]

Collating Sequence Decimal/Octal		1108 Graphic	Card Punch	Display Code	CYBER Graphic
00	00	@	8-7	61	[
01	01	[	12-8-5	75	≥
02	02	]	11-8-5	70	↑
03	03	∀	12-8-7	77	;
04	04	Δ	11-8-7	73	>
05	05	blank	no punch	55	blank
06	06	A	12-1	01	A
07	07	B	12-1	02	B
08	10	C	12-3	03	C
09	11	D	12-4	04	D
10	12	E	12-5	05	E
11	13	F	12-6	06	F
12	14	G	12-7	07	G
13	15	H	12-8	10	H
14	16	I	12-9	11	I
15	17	J	11-1	12	J
16	20	K	11-2	13	K
17	21	L	11-3	14	L
18	22	M	11-4	15	M
19	23	N	11-5	16	N
20	24	O	11-6	17	O
21	25	P	11-7	20	P
22	26	Q	11-8	21	Q
23	27	R	11-9	22	R
24	30	S	0-2	23	S
25	31	T	0-3	24	T
26	32	U	0-4	25	U
27	33	V	0-5	26	V
28	34	W	0-6	27	W
29	35	X	0-7	30	X
30	36	Y	0-8	31	Y
31	37	Z	0-9	32	Z

TABLE A-5. UNIVAC 1100 SERIES COLLATING SEQUENCE [UNI] (Contd)

Collating Sequence Decimal/Octal		1108 Graphic	Card Punch	Display Code	CYBER Graphic
32	40	)	12-8-4	52	)
33	41	−	11	46	−
34	42	+	12	45	+
35	43	<	12-8-6	76	¬
36	44	=	8-3	54	=
37	45	>	8-6	63	%
38	46	&	8-2	00	:
39	47	$	11-8-3	53	$
40	50	*	11-8-4	47	*
41	51	(	0-8-4	51	(
42	52	%	0-8-5	65	→
43	53	:	8-5	74	≤
44	54	?	12-0	72	<
45	55	!	11-0	66	∨
46	56	,	0-8-3	56	,
47	58	\	0-8-6	60	≡
48	60	0	0	33	0
49	61	1	1	34	1
50	62	2	2	35	2
51	63	3	3	36	3
52	64	4	4	37	4
53	65	5	5	40	5
54	66	6	6	41	6
55	67	7	7	42	7
56	70	8	8	43	8
57	71	9	9	44	9
58	72	'	8-4	64	≠
59	73	;	11-8-6	71	↓
60	74	/	0-1	50	/
61	75	.	12-8-3	57	.
62	76	□	0-8-7	67	∧
63	77	≢	0-8-2	62	]

DIAGNOSTICS

Diagnostics occur at two distinct times:

During compilation, the compiler itself reports program situations that do not conform to language specifications.

During execution of the compiled program, the execution-time routines of the system report situations that do not allow the program to execute.

Diagnostics are reported in the dayfile and in the error file identified by the compiler call.

Errors in the compiler call itself are reported by these messages:

* aaa * IS FOLLOWED BY AN INVALID SEPARATOR.

The valid separators, subject to correct usage are , .) = and /.

* bbb * IS NOT A VALID PARAMETER.

See Compiler Call in section 12 for a list of valid parameters.

* ccc * APPEARS MORE THAN ONCE.

Each parameter of the compiler call can appear only once.

* ddd * IS NOT A VALID VALUE FOR THE CURRENT PARAMETER.

This option is invalid for this parameter. See Compiler Call in section 12 for the options allowed.

* eee * APPEARS MORE THAN ONCE AS A VALUE OF A PARAMETER.

Each value of a parameter can appear only once.

* fff * CANNOT BE USED ALONG WITH OTHER VALUES.

This parameter cannot have multiple values assigned to it.

* ggg * CANNOT BE USED WITHOUT VALUES.

This parameter, when used, must have values assigned to it, for example, PS.

* hhh * MUST APPEAR WITHOUT VALUES.

This parameter, when used, cannot have values assigned to it, for example, APO.

* iii jjj * PARAMETERS CANNOT USE THE SAME FILE

L and E can share the same file, I and FDL can share the same file, but all other parameters must specify unique file names.

DAYFILE MESSAGES

The dayfile is the summary of a job during its flow through the system. Printed output from every batch job contains a job dayfile. When a program is compiled or executed interactively through a terminal, messages that would otherwise be written to the dayfile are returned to the terminal. Dayfile messages are under system control. They cannot be expanded or suppressed by the program or compiler call.

Detailed diagnostics of program errors are listed in the file OUTPUT or in the file designated by the E parameter of the compiler call. Only the messages that result from system errors or fatal compilation or execution conditions are reported in the dayfile.

Messages written to the dayfile as a result of compilation include status messages and notification of programmer or system errors:

nnnnnnB CM, sss.sss CPS, eeeeeeB ECS

Informative message. The number of central memory words required for compilation is nnnnnn octal. The number of central processor seconds required for compilation is sss.sss. The amount of ECS used for compilation is eeeeee (octal).

nnn T/W AND mmm F ERRORS IN program-id

Informative message. The number of trivial and warning messages generated during compilation is nnn. The number of fatal errors generated is mmm. The name of the program is program-id. Fatal errors must be corrrected before the program can compile and/or execute successfully. An error in the PROGRAM-ID paragraph produces a program name of NO-NAME.

EMPTY SOURCE FILE

Programmer error. The file named as having the source program to be compiled does not have a source program. Possibly the I parameter on the compiler call is in error. If the file is correct, it might be positioned at the end, rather than at the beginning, of the source program.

COBOL 5 COMPILER ABORT

Error message. Programmer error in the compiler call caused the compiler to abort. A following message, such as *K* IS NOT A VALID PARAMETER, might pinpoint the error.

CGABORT or other message beginning with CG

System error during compilation. Follow site-defined procedures for reporting software errors or operational problems.

Messages written to the dayfile as a result of program execution include status messages and notification of programmer or system errors; for example:

SOURCE STATEMENT SYNTAX ERROR

Programmer error in source program.

CRM ERROR NUMBER nnn

CRM error nnn detected. Follow site-defined procedures for reporting software errors or operational problems.

COMPILATION DIAGNOSTICS

During source program compilation, the compiler produces a variety of diagnostic messages. Through parameters on the compiler call, the programmer can control the type of messages listed, the location of the messages, and the system reaction to some messages.

Under default parameter selection, diagnostic information appears after the source listing on the file OUTPUT. The diagnostic header (shown in section 12) is followed by a list of error conditions detected by the compiler. A summary appears after the listing of individual diagnostics. When a program compiles without error, no diagnostic information appears.

The total number of errors listed is reported by a message:

n ERRORS LISTED

The EL parameter of the compiler call determines the level of diagnostics listed. Levels are:

T Trivial. Suspicious usage: although the syntax is correct, the usage is questionable.

An example of a trivial diagnostic is error number 1057: ONLY THE FIRST 7 CHARACTERS OF THE PROGRAM NAME ARE USED TO INTERACT WITH THE SYSTEM. Although the language syntax allows a 30-character user-defined word as the program name, the system uses only the first 7 characters.

W Warning. Syntax is incorrect, but compilation of the statement continues.

An example of a warning diagnostic is error number 1026: THIS ELEMENT MAY NOT BEGIN IN THE A AREA. A program with a warning diagnostic often executes successfully.

F Fatal. Unresolved semantic error such that the statement cannot be compiled.

An example of a fatal diagnostic is error number 3080: AN ENTRY WITH LEVEL 02 THROUGH 49 IS ONLY ALLOWED AFTER AN ENTRY WITH LEVEL 01 THROUGH 49 OR LEVEL 88. A program with a fatal error inhibits loading unless the DB=B parameter is used on the compiler call.

C Catastrophic. Compiler error: although the compilation does not usually terminate, a system analyst should be notified.

An example of a catastrophic diagnostic is error number 8020: COMPILER ERROR - INVALID OBJECT TYPE. Such errors should not appear. Any such message should be reported to a system analyst.

A fifth type of diagnostic, which cannot be directly referenced on the EL parameter, is N.

N Usage is allowed by Control Data extensions to the standard language, but not by the ANSI standard itself. The program executes successfully, unless the ANSI=opt parameter specifies that such diagnostics are to be treated as a fatal error.

An example of a nonstandard diagnostic is error number 7284: THE INITIALIZE STATEMENT IS NON-STANDARD. Such statements must be removed if the program is to conform to strict ANSI usage.

N type diagnostics also indicate that usage is supported at the specified FIPS level.

An example of a FIPS diagnostic is error number 3717: FIPS=3 SUPPORTS OCCURS DEPENDING ON.

When the EL parameter specifies a T, W, F, or C option, only diagnostics of the specified level or higher are listed as part of the diagnostic output of compilation. Levels in order of increasing severity are: T, W, F, and C.

The compiler tallies unlisted messages and reports the total as part of the diagnostic summary. Messages are:

nn UNLISTED level ERRORS

A separate message appears for each level: TRIVIAL, WARNING, or FATAL. To obtain a listing of these errors, it is necessary to recompile the program with a different EL parameter.

Nonstandard messages are summarized by:

nn NON-ANSI MESSAGES

This message only appears if the program is compiled with the ANSI parameter specifying that nonstandard usage is to be diagnosed. To obtain a listing of these errors, it is necessary to compile with EL=T.

The compilation-time diagnostics are listed without additional explanation in the COBOL 5 diagnostic handbook.

EXECUTION DIAGNOSTICS

During program execution, several types of messages are produced.

Status messages that require no action on the part of the programmer. An example of such a message is START COBOL SORT.

Fatal error messages indicating errors that must be corrected before the program can execute successfully. The errors might be system errors or programmer errors. System errors should be reported to a system analyst.

When a fatal error causes execution to terminate, several lines of diagnostics usually are produced. These lines identify:

Line containing the last source statement executed.

Program containing the last source statement executed and all programs with active CALL statements, including the line numbers of the CALL statements.

Condition that caused job termination.

Table B-1 presents the execution-time diagnostics in alphabetical order. The table lists the message, its significance, and possible programmer corrective action to eliminate the error.

TABLE B-1. EXECUTION DIAGNOSTICS

Message	Significance	Action
-AT END- NOT ON ROOT AREA	CDCS problem.	Follow site-defined procedures for reporting software errors or operational problems.
AT END WHILE TRYING TO ACCEPT FROM FILE	Partition boundary encountered. Error code is 2022 if ENTER "C.IOST" is used.	Change program logic to detect and process the last file record.
ATTEMPT TO OPEN A LOCKED FILE	Source program error. Error code is 2005 if ENTER "C.IOST" is used.	Remove WITH LOCK of CLOSE or change OPEN statement.
ATTEMPT TO OPEN FILE OPENED BY ACCEPT OR DISPLAY AND PROCESSING DIRECTION DIFFERS	OPEN INPUT on a file used by ACCEPT or OPEN OUTPUT or OPEN I-O or OPEN EXTNED on a file used by DISPLAY. Error code is 2020 if ENTER "C.IOST" is used.	Correct source program.
ATTEMPT TO OPEN FILE WITH SAME NAME AS AN OPEN FILE	Source program error. Error code is 2003 if ENTER "C.IOST" is used.	Change program to reference correct file, file before reopen, or remove file references in subprogram.
CALL CAUSED OVERFLOW, NO ON OVERFLOW SPECIFIED	The program requested could not be loaded.	Specify ON OVERFLOW phrase of CALL statement or increase allowed field length.
CALLED FROM xxxxx AT SOURCE LINE yyyyy	Additional information.	None.
CALLED PROGRAM NOT IN FDL FILE	CALL statement was executed, but program named by literal or identifier was not on FDL file.	Correct FDL file or source program.
CALLS NESTED TOO DEEP IN TAF TASK	More than nine CALL statements executed without EXIT PROGRAM statements in called programs.	Change the task to decrease the number of active calls.
CANCEL FOR ROUTINE WHICH HAS NOT TAKEN EXIT	Source program error.	Check run unit logic. A program cannot be cancelled unless it has executed an EXIT PROGRAM statement.
CANCEL FOR STATIC ROUTINE IGNORED	Program declared as STATIC in FDL file or by default was cancelled; execution not terminated.	None.
CANCELLED PROGRAM NOT IN FDL FILE	CANCEL statement was executed and program named by literal or identifier was not on FDL file.	Correct FDL file or source program.

TABLE B-1. EXECUTION DIAGNOSTICS (Contd)

Message	Significance	Action
COBOL SYSTEM ERROR - IO STACK OVERFLOW	System error.	Follow site-defined procedures for reporting software errors or operational problems.
COBOL SYSTEM ERROR - IO STACK UNDERFLOW	System error.	Follow site-defined procedures for reporting software errors or operational problems.
CRM ERROR NUMBER nnn	Source program or system error nnn.	See appropriate CYBER Record Manager reference manual for error code explanation.
C.DTCMP - PARAMETER NOT ALPHANUMERIC OR LESS THAN 8 CHARACTERS IN LENGTH	The receiving parameter for C.DTCMP is incorrect.	Make parameter alphanumeric and at least 8 characters in length.
C.FILE CALL - COMMA REQUIRED HERE IN CHAR nn	ENTER "C.FILE" passed an incorrect parameter string.	Correct parameter string.
C.FILE CALL - FILE ORG NOT DIRECT AND TRC SPECIFIED	ENTER "C.FILE" passed an incorrect parameter string.	Remove TRC parameter from parameter list or change file organization.
C.FILE CALL - ILLEGAL PARAMETER STARTING IN CHAR nn	ENTER "C.FILE" passed an incorrect parameter string.	Correct parameter string.
C.FILE CALL - LFN NOT LEGAL FILE NAME - NAME IS xxxxxxx	File-name specified in ENTER "C.FILE" is not a legal file-name.	Correct call.
C.FILE CALL - PARAMETER DATA ITEM NOT ALPHANUMERIC	Second parameter of ENTER "C.FILE" is not an alphanumeric call.	Correct call.
C.FILE CALL - = SIGN REQUIRED HERE IN CHAR nn	Equals sign is required in this parameter.	Correct parameter.
C.GETBK GIVEN ILLEGAL BLOCK TYPE	System error.	Follow site-defined procedures for reporting software errors or operational problems.
DEADLOCK CONDITION AND NO DEADLOCK DECLARATIVE	Source program error.	Correct program logic or insert a deadlock declarative.
DELETE FILE ON OPEN FILE - IGNORED	A DELETE FILE was issued on an open file. The statement is ignored and processing continues. Error code is 2021 if ENTER "C.IOST" is used.	Correct program logic.
DELETE OR REWRITE ON SEQ ACCESS FILE WITH NO VALID READ PRECEDING	Source program error. Error code is 2012 if ENTER "C.IOST" is used.	Insert READ before DELETE or REWRITE, or change to ACCESS MODE DYNAMIC.
ECS TRANSFER ERROR	System error.	Follow site-defined procedures for reporting software errors or operational problems.
END COBOL MERGE	Merge status.	None.
END COBOL SORT	Sort status.	None.
ERROR IN C.GETEP CALL - FATAL	Parameter required in USING phrase of ENTER "C.GETEP" statement is not an alphanumeric item or is not specified.	Correct call.
FILE OPENED INPUT OR I-O DOES NOT EXIST	Source program or job error. Error code is 2004 if ENTER "C.IOST" is used.	Make file available before executing program or change OPEN statement.

TABLE B-1. EXECUTION DIAGNOSTICS (Contd)

Message	Significance	Action
GIVING MRL LESS THAN SORT FILE MRL	Maximum record length of GIVING file is less than maximum record length of SD.	Reconcile GIVING file and SD file record length specifications.
GROW REQUEST ON FIXED SIZE BLOCK	System error.	Follow site-defined procedures for reporting software errors or operational problems.
HOME BLOCK COUNT NOT SPECIFIED FOR NEW DA FILE	Number of blocks to preallocate is unknown. Error code is 2006 if ENTER "C.IOST" is used.	Add BLOCK COUNT clause, or HMB to USE clause, or HMB on FILE control statement.
ILLEGAL EXIT FROM PERFORM	In a segmented program, an active PERFORM statement has an execution point beginning within the range of another active PERFORM statement. Control was passed to the wrong EXIT. A GO TO might have passed control out of its PERFORM range and not returned to the normal EXIT.	Correct program logic. All paths within the range of a PERFORM must be to its EXIT.
ILLEGAL RECORD TYPE FOR REWRITE	In sequential files, the record type for a record to be rewritten must be either F (fixed length) or W (control word). Error code is 2015 if ENTER "C.IOST" is used.	Change file description to generate either F or W type records.
ILLEGAL REFERENCE MODIFICATION	Reference modifiers are out of range.	Correct program logic.
INTERNAL FILE NAME = lfn	Accompanying messages reference file lfn, which is the implementor-name of SELECT.	Correct program.
INVALID PARAMETER TO -C.IOST-	An input parameter to C.IOST is not defined as expected. The call to C.IOST is ignored.	Correct input parameter.
INVALID RELATION-NAME	Source program error.	Reconcile relation-name with sub-schema.
JOB ABORTED	Job status.	Correct condition reported with this message.
LOADER ERROR LOADING COBOL SEGMENT	System error.	Follow site-defined procedures for reporting software errors or operational problems.
MRL OR FL SET IN FILE STATEMENT IS LARGER THAN RECORD AREA - TRUNCATED TO RECORD AREA SIZE	Error is in CRM FILE control statement. MRL is set to record area size and program execution continues. Error code is 2024 if ENTER "C.IOST" is used.	Remove FILE control statement.
MULTI-FILE NOT ASSIGNED TO MAGNETIC TAPE	REQUEST control statement is incorrectly specified or is missing. Error code is 2016 if ENTER "C.IOST" is used.	Assign tape with REQUEST control statement specifying MF option.
MULTI-VOLUME VERSION 4 RELATIVE FILE ILLEGAL	File not compatible with COBOL 5 because CLOSE UNIT written to file.	Recreate file without UNIT phrase of CLOSE statement.
NO EXCEPTION PROCESS PROVIDED	Source program omission.	Add USE AFTER STANDARD ERROR declarative or add AT END or INVALID KEY to input or output statement for file.

TABLE B-1. EXECUTION DIAGNOSTICS (Contd)

Message	Significance	Action
OLD RELATIVE FILE EMPTY - NO HEADER	Conflict between specifications and existing file. Error code is 2011 if ENTER "C.IOST" is used.	Reconcile file description and file.
PARAMETER MISMATCH	Number of parameters specified in the USING phrase of the CALL statement is greater than or less than the number specified in the USING phrase of the Procedure Division header.	Specify identical number of parameters in USING phrases.
PRE 446 BINARY CANNOT BE EXECUTED	Attempt was made to execute binary deck created before PSR level 446.	Recompile program.
READ ON FILE OPENED FOR OUTPUT	Source program error. Error code is 2017 if ENTER "C.IOST" is used.	Change logic or open file for INPUT or I-O.
READ . . . NEXT WITH UNDEFINED CURRENT REC POINTER	READ statement cannot be executed because internal record is undefined. Error code is 2013 if ENTER "C.IOST" is used.	Correct prior condition that caused record pointer to be undefined.
RECORD LENGTH ON OLD RELATIVE FILE DIFF THAN PROGRAM	Conflict between specifications and existing file. Error code is 2007 if ENTER "C.IOST" is used.	Reconcile file description and file.
RECORD TYPE NOT PRINTABLE ON AN ADVANCING FILE	Conflict in specifications for file referenced by ADVANCING phrase of WRITE. Error code is 2002 if ENTER "C.IOST" is used.	Change RT parameter of FILE control statement or USE literal clause.
nnnn RECORDS RELEASED	Sort status. nnnn is record count.	None.
nnnn RECORDS RETURNED	Sort or merge status. nnnn is record count.	None.
RECORDING MODE BINARY ON ADVANCING FILE	Conflict in specifications for file referenced by ADVANCING phrase of WRITE. Error code is 2001 if ENTER "C.IOST" is used.	Change to RECORDING MODE DECIMAL or otherwise resolve conflict.
REFERENCE TO LINKAGE SECTION ITEM NOT IN USING LIST	A Linkage Section item is referenced, but is not specified in USING phrase of Procedure Division header.	Correct program.
RELATIVE FILE CREATED WITH PRUF=YES BUT RE-OPENED AS NON PRUF	An OPEN INPUT or I-O or EXTEND was executed on a relative file that was created in a program with PRUF=YES written in the USE literal in the SELECT clause. This program does not have such a USE literal. Error code is 2023 if ENTER "C.IOST" is used.	Put PRUF=YES in the SELECT clause and recompile the program.
RELATIVE FILE HEADER NOT VALID	Conflict between specifications and existing file. Error code is 2010 if ENTER "C.IOST" is used.	Reconcile file description and file.
-RELEASE- RECORD-NAME NOT IN CURRENT SD	Conflict in specifications.	Correct record-name in SD or RELEASE statement.
-RELEASE- USED IMPROPERLY	Illegal RELEASE statement use.	Move RELEASE to within SORT input procedure.
RETURN EXECUTED AFTER AT END DETECTED	A RETURN statement was executed after the previous RETURN in this output procedure took the AT END exit. AT END exit is taken and execution continues until 21 such errors are detected.	Correct program logic.

TABLE B-1. EXECUTION DIAGNOSTICS (Contd)

Message	Significance	Action
-RETURN- FILE-NAME NOT CURRENT SD	Conflict in specifications.	Correct file-name in SD or RETURN statement.
-RETURN- USED IMPROPERLY	Illegal RETURN statement use.	Move RETURN to within a SORT or MERGE output procedure.
REWRITE RL DOES NOT EQUAL OLD RL	For sequential files, the length of the record on a REWRITE must be the same as the record to be rewritten. Error code is 2014 if ENTER "C.IOST" is used.	Change record description to match the record being written.
RWCS ERROR IN BODY-NEXT-GROUP ROUTINE	System error.	Follow site-defined procedures for reporting software errors or operational problems.
RWCS ERROR IN BODY-PAGING ROUTINE	System error.	Follow site-defined procedures for reporting software errors or operational problems.
RWCS ERROR IN SPACING ROUTINE	System error.	Follow site-defined procedures for reporting software errors or operational problems.
SET CODE-SET ON ACTIVE FILE	Source program error.	Close file before SET statement.
SORT/MERGE WITHIN SORT/MERGE	Illegal SORT or MERGE statement use.	Remove SORT or MERGE from input procedure or output procedure.
SOURCE STATEMENT SYNTAX ERROR	Source program error.	Correct statement.
S T A R T C O B O L M E R G E	Merge status.	None.
S T A R T C O B O L S O R T	Sort status.	None.
STOP RUN NOT ENCOUNTERED IN PROGRAM	Source program error.	Add STOP RUN as last executable statement.
SUBSCRIPT OUT OF RANGE	Source program error.	Correct subscript or program logic.
THE REPORT FOOTING AND THE LAST PAGE FOOTING OVERLAP SO THE REPORT FOOTING IS SUPPRESSED BY THE RWCS	Report Writer status.	Correct page specifications.
THE REPORT HEADING AND THE FIRST PAGE HEADING OVERLAP SO THE PAGE HEADING IS SUPPRESSED BY THE RWCS	Report Writer status.	Correct page specifications.
THE REPORT IS ALREADY INITIATED SO THIS INITIATE IS IGNORED BY THE RWCS	Report Writer status.	Remove duplicate INITIATE statement or otherwise correct program.
THE REPORT IS NOT INITIATED SO THIS TERMINATE IS IGNORED BY THE RWCS	Report Writer status.	Correct program to initiate report.
THE REPORT WAS NOT INITIATED SO THE RWCS INITIATED IT BEFORE PROCESSING THIS GENERATE	Report Writer status.	Add INITIATE statement or otherwise correct program.

TABLE B-1. EXECUTION DIAGNOSTICS (Contd)

Message	Significance	Action
UNALTERED GO TO EXECUTED	A GO TO statement was executed and never altered.	Delete statement or change logic to set statement via ALTER.
UNIVAC TRANSLATE TABLE -C.XTAB- NOT IN CORE	System error.	Follow site-defined procedures for reporting software errors or operational problems.
UNKNOWN BLOCK OR BAD POINTER ADDR	System error.	Follow site-defined procedures for reporting software errors or operational problems.
USING MRL EXCEEDS SORT FILE MRL	Maximum record length of USING file exceeds maximum record length of SD. Records might be truncated.	Reconcile USING file and SD file record length specificaions.

GLOSSARY C

Access Control -
Protection of data from unauthorized access or modification. Access control is provided by the CDCS privacy module.

Access Control Key -
The value an applications program must supply to CDCS in order to gain access to a particular data base area.

Access Mode -
Manner in which records can be inserted into or retrieved from a file. Can be sequential, random, or dynamic, depending on the ACCESS MODE clause. Access mode, open mode, and file organization affect subsequent operations permitted.

Actual-Key File -
File described by ORGANIZATION IS ACTUAL-KEY clause. For initial actual-key files, the primary key specifies the block and record slot number in which the record is stored. For extended actual-key files, the primary key is a record number that AAM converts to the storage location of the record. Program must perform any key indexing.

Advanced Access Methods (AAM) -
File manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor. See CYBER Record Manager.

Alphabetic Character -
Character belonging to set of letters A through Z and the space.

Alphanumeric Character -
Any character in computer character set defined in appendix A. Most formats allow only characters from the COBOL character set.

Alternate Record Key -
Record key defined by ALTERNATE RECORD KEY clause. Can be used to read, but not write or update, a record in an indexed, direct, or actual-key file. System creates an index that relates alternate record keys to primary keys using the file defined by the second implementor-name of an ASSIGN clause. Job must preserve index file.

ANSI Standard Language -
Language defined in American National Standard X3.23-1974, COBOL, on which this COBOL 5 compiler is based. Control Data extensions to the language are shaded in this manual.

Application -
The programs, sources, destinations, queues, and journals that are defined to MCS as an application through the Application Definition Language (ADL). Typical examples are order entry, inventory control, and accounting.

Application Definition Language (ADL) -
A language that defines and describes application components to MCS.

Area -
Uniquely named data base subdivision that contains data records; a file.

Arithmetic Expression -
Any combination of numeric elementary items, numeric literals, and the figurative constant ZERO connected by arithmetic operators to form an expression that reduces to a single value when it is evaluated during program execution.

Arithmetic Operator -
+ to indicate addition; - to indicate subtraction; * to indicate multiplication; / to indicate division; ** to indicate exponentiation.

Assumed Decimal Point -
Decimal point position that does not involve the existence of an actual character in a data item. Has logical meaning but no physical representation.

At End Condition -
Condition that exists when no further records or data are available for processing during execution of: READ statement, RETURN statement, SEARCH statement when no conditions are satisfied. Sets FILE STATUS data item.

Basic Access Methods (BAM) -
File manager that processes sequential and word addressable file organizations. See CYBER Record Manager.

Beginning-of-Information (BOI) -
CYBER Record Manager defines beginning-of-information as the start of the first user record in a file. System-supplied information, such as an index block or control word, does not affect beginning-of-information. Any label on a tape exists prior to beginning-of-information.

Block -
The term block has several meanings depending on context. On tape, a block is information between interrecord gaps. CYBER Record Manager defines several blocks depending on organization, as shown in table C-1.

TABLE C-1. BLOCK TYPES

Organization	Blocks
Indexed sequential	Data block; index block
Direct access	Home block; overflow block
Actual key	Data block
Sequential	Block type I, C, K, E

Body Group -
In Report Writer, generic name for a report group defined by a TYPE clause with a DETAIL, CONTROL HEADING, or CONTROL FOOTING phrase.

Boolean Data Item -
A data item consisting entirely of the boolean characters zero and one.

Boolean Expression -
An identifier of a boolean data item, a boolean literal, such identifiers and/or literals separated by a boolean operator, two boolean expressions separated by a boolean operator, or a boolean expression enclosed in parentheses.

Boolean Literal -
A literal composed of one or more boolean characters delimited on the left by the separator B" and on the right by the quotation mark separator.

Break Control Item -
In Report Writer, item defined by CONTROL clause. Synonymous with Control Data Item defined by ANSI standard.

Broadcast List -
A symbolic name defined in ADL that references a group of destinations.

Called Program -
Program that is the object of a CALL statement.

Calling Program -
Program that executes a CALL statement.

Capsule -
A relocatable collection of one or more programs bound together in a special format that allows the programs to be loaded and unloaded dynamically from an executing program by the Fast Dynamic Loader facility.

Character-String -
Sequence of contiguous characters that form a COBOL word, a literal, a PICTURE clause character-string, or a comment-entry.

Clause -
Ordered set of COBOL character-strings that make up an entry.

COBOL Character Set -
51-character set listed in table 1-1 in section 1. Defined by ANSI standard.

COBOL Communication Facility (CCF) -
The part of the COBOL language that allows a user to send and receive messages from terminals. CCF allows the COBOL user to interface with the Message Control System (MCS). This interface is allowed only under the NOS operating system.

Collating Sequence -
Sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, and comparing. Defined by default, COLLATING SEQUENCE clause, ALPHABET clause, or SET statement.

Comment-Entry -
Entry in Identification Division that can contain any combination of characters from computer character set.

Comment Line -
Source program line with asterisk or slash in column 7 and any characters from computer character set in area A and area B of that line. Documentary only. / in column 7 ejects page before comment is listed in source listing.

Compilation Time -
Time at which a COBOL source program is translated by the COBOL 5 compiler to an object program that can be loaded and executed. Defined by control statement in batch job or interactive command. Contrast with Execution Time.

Control Break -
In Report Writer, change in value of data item defined by CONTROL clause that is used to control hierarchical structure of a report. Control totals are accumulated and formatted for presentation to the report file when a control break occurs.

Control Data Item -
In Report Writer, synonym for Break Control Item.

Control Word -
A system-supplied word that precedes each W type record in storage.

Currency Symbol -
Character in program character set equivalent to display code value 53; or character defined by CURRENCY SIGN clause.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM which run under the NOS and NOS/BE operating systems and allow a variety of record types, blocking types, and file organizations to be created and accessed. The execution time input/output of COBOL, FORTRAN, Sort/Merge 4, ALGOL, and the DMS-170 products is implemented through CYBER Record Manager. Neither the input/output of the NOS and NOS/BE operating systems themselves nor any of the system utilities such as COPY or SKIPF is implemented through CYBER Record Manager. All CYBER Record Manager file processing requests ultimately pass through the operating system input/output routines.

Data Base -
Collection of information defined by a schema. Sub-schema defines portion of data base that is to be accessed by a COBOL program. Created through Data Description Language; accessed through CYBER Database Control System.

Data-Name -
User-defined word that names a data item described in a Data Description entry. Cannot be subscripted, indexed, qualified, or reference modified unless specifically permitted by a given format.

Deadlock -
A situation that arises in concurrent data base access when two or more applications programs are contending for a resource that is locked, and none of the programs can proceed without that resource.

Debugging Facility -
Capability within COBOL 5 compiler and execution routines that implements the DEBUGGING MODE clause, lines with D in column 7, and debugging declarative sections.

Debugging Line -
Any line with D in column 7 of the source program. Compiled as executable code if DEBUGGING MODE clause is used or DB=DL parameter appears on compiler call; otherwise compiled as comment line.

Delimited Scope Statement -
Any statement that includes an explicit scope terminator.

Delimiter -
Character or sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. Not part of the string of characters that it delimits.

Destination -
A named output device or collection of output devices that can receive messages. The recipient of a transmission from an output queue.

Direct Access File -
In the context of AAM, a direct access file is one of the three file organizations. Implemented according to AAM initial direct or extended direct files. It is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks called home blocks of the file. Synonymous with direct file.

In the context of NOS permanent files, a direct access file is a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

Direct File -
File described by ORGANIZATION IS DIRECT clause. Characterized by preallocated home blocks. Location of record determined by hashing key of record to a home block number. Synonymous with direct access file.

Disable -
The term that indicates deactivating the logical connection between MCS and one or more communications devices under the NOS operating system.

Dynamic Access -
Access mode that allows a nonsequential file on mass storage to be accessed randomly or sequentially depending on the format of the access statement.

Dynamic Program -
Program loaded by Fast Dynamic Loader facility at the time the first CALL statement referencing the program is executed.

Edited Item -
Item whose PICTURE clause contains an editing symbol B 0 + - CR DB Z * $, . or / .

Embedded Key -
A key that is an integral part of a record, as opposed to a key that is defined in the Working-Storage Section of a COBOL program.

Enable -
A term used to indicate activating or reactivating the logical connection between MCS and one or more given communication devices under the NOS operating system.

End-of-Group Indicator (EGI) -
A character defined in the application definition that logically separates a group of several messages from succeeding messages and signals the end of a group of messages to MCS or a COBOL program.

End-of-Information (EOI) -
CYBER Record Manager defines end-of-information in terms of the file organization and file residence. See table C-2.

TABLE C-2. END-OF-INFORMATION BOUNDARIES

File Organization	File Residence	Physical Position
Sequential	Mass storage	After the last user record.
	Labeled tape in SI, I, S, or L format	After the last user record and before any file trailer labels.
	Unlabeled tape in SI or I format	After the last user record and before any file trailer labels.
	Unlabeled tape in S or L format	Undefined.
Word Addressable	Mass storage	After the last word allocated to the file, which might be beyond the last user record.
Indexed Sequential, Actual Key	Mass storage	After the record with the highest key value.
Direct Access	Mass storage	After the last record in the most recently created overflow block or home block with the highest relative address.

End-of-Message Indicator (EMI) -
A conceptual indicator that delimits one message from the next message and notifies MCS or a COBOL program that the end-of-message condition exists.

End-of-Segment Indicator (ESI) -
A conceptual indicator that delimits one segment within a message from the next segment within a message and notifies MCS or a COBOL program that the end-of-segment condition exists.

Entry -
Descriptive set of consecutive clauses terminated by a period and written in Identification, Environment, or Data Division.

Execution Time -
Time at which a compiled source program is executed. Also known as object time. Defined by LGO or EXECUTE control statement or their equivalent in a batch job or interactive command.

Explicit Scope Terminator -
A reserved word that is included in a delimited scope statement and which terminates the scope of a particular conditional statement. Examples are END-IF, END-SEARCH, and END-PERFORM.

Extended -
A term used in conjunction with indexed, direct, and actual-key files to denote a specific type of internal processing by AAM and MIP. Processing is indicated by setting the ORG FIT field to NEW. Contrast with Initial.

Extended Memory -
Core type storage which is physically located outside of the machine. Formerly referred to as Extended Core Storage (ECS) or Large Central Memory (LCM).

External File -
A file that contains the EXTERNAL clause in the FD entry for the file. The file can be described in any program in the run unit, but it exists externally to the program and is shared by all programs that describe it. All External files in a run unit must be described in the main program.

Federal Information Processing Standard (FIPS) -
Standard used by the federal government for internal control or procurement. FIPS PUB 21-1, issued by the National Bureau of Standards, describes a set of leveling specifications and requires all implementors to diagnose items differing from a specified level. Level to be diagnosed is specified through FIPS parameter on compiler call.

File -
Collection of records defined by SELECT clause and described by FD, SD, or RD entry.

File Organization -
Defined by ORGANIZATION clause. Can be sequential, indexed, relative, direct, actual-key, or word-address. Established at time file is created and cannot change as long as file exists. Affects access mode, open mode, and formats of statements that can be used to manipulate file records.

High Order End -
Leftmost character of string of characters; leftmost bit of a string of bits.

Home Block -
Mass storage allocated for a file with direct access organization at the time the file is created.

Implementor-Name -
System-name that refers to a particular feature available in COBOL 5. Particular implementor- names such as CDC-64 and SWITCH-6 are unique to COBOL 5 but are within ANSI standard.

Index -
In the context of AAM, a series of keys and pointers to records associated with the keys. The system creates an index for AAM files which relates alternate record keys to primary keys, using the file defined by the second implementor-name of an ASSIGN clause.

In general context, a computer storage area or register, the content of which represents the identification of a particular element.

Index Data Item -
Data item defined by USAGE IS INDEX clause. Can hold index-names.

Index File -
In the context of AAM, a file that contains a series of keys and pointers to records associated with the keys. The index file, once created, must be preserved by a job. See Index.

Index-Name -
User-defined word that names an index associated with a specific table. Defined by INDEXED phrase of OCCURS clause.

Indexed File -
File described by ORGANIZATION IS INDEXED clause. Implemented according to AAM initial indexed or extended indexed files. Characterized by records always being in physical and logical order by prime key values. Synonymous with indexed sequential file.

Initial -
A term used in conjunction with indexed, direct, and actual key files to denote a specific type of internal processing by AAM and MIP. Processing is indicated by setting the ORG FIT field to OLD. Contrast with Extended.

Input File -
File referenced in OPEN statement with INPUT phrase after OPEN statement execution and before CLOSE statement execution.

Input Procedure -
In Sort/Merge, set of statements defined by INPUT PROCEDURE phrase of SORT statement. Executes prior to physical sort. Alternative to USING phrase. Must include RELEASE statement for each record to be sorted.

Integer -
Numeric literal or numeric data item that does not include any character positions to the right of the assumed decimal point. Must not be signed or have a zero value unless explicitly allowed for a given format.

Invalid Key Condition -
Condition that exists during execution when a specific value of the key associated with an indexed, direct, actual-key, or relative file is not valid for the access being attempted. Sets FILE STATUS data item.

Invitation list -
A symbolic name defined in ADL that refers to a group of sources.

Job Step -
The execution of a control statement.

Journal -
A file that receives copied messages as they are transferred into or out of queues. The journal provides a record of message transfer activities for recovery purposes.

Key Item -
Data item that defines the location, size, and characteristics of a key for a relative, indexed, direct, actual-key, or word-address file. During execution, contents of item specifies information the system can use to locate record in file. Also, data item that serves to identify the ordering of data.

Key of Reference -
Either the primary key or alternate record key currently being used to access record in an indexed, direct, or actual-key file.

Keyword -
Reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name -
System-name that specifies a particular programming language. COMPASS and FORTRAN-X are the only language-names allowed in the ENTER statement.

Level -
Value of a level-number that indicates either the hierarchical position of a data item or the special properties of a Data Description entry. Value can be 01 through 49, 66, 77, or 88.

For system-logical-records, an octal number 0 through 17 in the system-supplied 48-bit marker that terminates a short or zero-length PRU.

Library -
In COBOL, source of text to be used during COPY statement processing. In operating system context, file produced by EDITLIB utility containing executable code in format required by system loader.

Library-Name -
User-defined word that names a file containing a random Update program library that is to be used by the compiler during compilation of a source program containing COPY statements. Also used to specify the library from which a dynamic subprogram is to be loaded.

Literal -
Character-string whose value is implied by the ordered set of characters that make up the string. See Nonnumeric Literal and Numeric Literal.

Logical Record -
In COBOL, equivalent to a record.

Under NOS, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. Equivalent to a system-logical-record under NOS/BE.

Low Order End -
Rightmost character of a string of characters; rightmost bit of a string of bits.

Mass Storage -
Disk pack or other rotating mass storage device. Not a magnetic tape.

Mass Storage File -
File assigned by control statements to a disk or disk pack. Can have any organization.

Message -
Data associated with an EMI or an EGI.

Message Control System (MCS) -
The subsystem that provides a generalized method of queuing, routing, and journaling of messages that pass between on-line COBOL programs and telecommunications equipment in the CYBER Network. It can only be used under the NOS operating system.

Message Count -
The count of the number of complete messages that exists in the designated message queue.

Message Indicators -
The conceptual indicators that notify MCS or an application program that the specified condition exists. The indicators are EGI (end-of-group indicator), EMI (End-of-Message Indicator), and ESI (end-of-segment indicator). An EGI indicator implies the presence of an ESI and an EMI. An EMI implies the presence of an ESI.

Message Segment -
Data that forms a logical subdivision of a message, normally associated with an ESI.

Mnemonic-Name -
User-defined word associated with an implementor-name in the SPECIAL-NAMES paragraph of the Environment Division.

Mode-Name -
System-name that refers to a particular method of data representation on a magnetic tape. DECIMAL and BINARY in the RECORDING MODE clause are the only two mode-names.

Multiple File Set -
Tape reel or reels with more than one individually labeled file.

Native Character Set -
Character set associated with system. Defined by installation when COBOL 5 is installed.

Native Collating Sequence -
Collating sequence associated with native character set. Defined by installation. Can be overridden by COLLATING SEQUENCE clause or SET statement.

Noise Record -
Number of characters the tape drivers discard as being extraneous noise rather than a valid record. Value depends on installation settings.

Nonnumeric Literal -
Literal bounded by quotation marks. Can include any character in computer character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character -
Digit 0 through 9.

Numeric Literal -
Literal composed of one or more numeric characters. Can contain decimal point, algebraic sign, or both. Decimal point must not be the rightmost character. Algebraic sign must be leftmost character.

Object Time -
See Execution Time.

Open Mode -
File state that allows records to be either written to or read from a file. Defined by OPEN statement phrase. Affects subsequent statements that can be used to access file.

Operand -
Data referenced by any lower-case word or words that appear in a statement or entry format.

Output Procedure -
In Sort/Merge, set of statements defined by OUTPUT PROCEDURE phrase of SORT or MERGE statement. Executes after physical sort or merge. Alternative to GIVING phrase. Must include RETURN statement for each record to be retrieved from sort or merge.

Overflow Block -
Mass storage the system adds to a file with direct access organization when records cannot be accommodated in the home block.

Paragraph -
In Procedure Division, paragraph-name followed by separator period and zero, one, or more entries. In Identification Division and Environment Division, paragraph header followed by zero, one, or more entries.

Paragraph Header -
In Identification Division and Environment Division, specific predefined reserved words followed by separator period.

Partition -
Defined by BAM as a division within a file with sequential organization. Generally, a partition contains several records or sections. Implementation of a partition boundary is affected by file structure and residence. See table C-3.

Notice that in a file with W type records, a short PRU of level 0 terminates both a section and a partition.

Permanent File -
Feature of operating system. When the job requests permanent file status, the file remains on mass storage when the job terminates so that it can be accessed by other jobs in the future. Requested by control statements outside source program.

Phrase -
Optional portion of a clause or statement.

Primary Key -
Key defined by RECORD KEY IS clause for indexed, direct, or actual-key file. Determines physical placement of a record. File creation and updating is only by key. Synonymous with prime key. Contrast with Alternate Record Key.

TABLE C-3. PARTITION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Boundary
PRU device	W	I	A short PRU of level 0 containing a one-word deleted record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary.
	W	C	A short PRU of level 0 containing a control word with a flag indicating a partition boundary.
	D,F,R, T,U,Z	C	A short PRU of level 0 followed by a zero-length PRU of level 17_8.
	S	-	A zero-length PRU of level number 17_8.
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary.
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with a flag indicating a partition boundary.
	D,F,T, R,U,Z	C,K,E	A tapemark.
	S	-	A tapemark.
Any other tape format	-	-	Undefined.

Printable Item -
In Report Writer, printable line is defined by a LINE NUMBER clause.

Privacy Module -
See Access Control.

Procedure -
In Procedure Division, paragraph or group of logically successive paragraphs or a section or group of logically successive sections.

Procedure-Name -
In Procedure Division, user-defined word that names a paragraph or section.

PRU -
Under NOS and NOS/BE, the amount of information transmitted by a single physical operation of a specified device. The size of a PRU depends on the device. See table C-4. A PRU which is not full of user data is called a short PRU; a PRU that has a level terminator but no user data is called a zero-length PRU.

TABLE C-4. PRU SIZES

Device	Size in Number of 60-Bit Words
Mass storage (NOS and NOS/BE only).	64
Tape in SI format with coded data (NOS/BE only).	128
Tape in SI format with binary data.	512
Tape in I format (NOS only).	512
Tape in any other format.	Undefined.

PRU Device -
Under NOS and NOS/BE, a mass storage device or a tape in SI or I format, so called because records on these devices are written in PRUs.

Pseudo-File-Name -
User-defined word that names a file residing on a tape referenced in a MULTIPLE TAPE FILE clause for which no FD entry is specified. File cannot be accessed in program.

Pseudo-Text -
Source program text consisting of character-strings, comment lines, and/or separators bounded by, but not including, pseudo-text delimiters.

Pseudo-Text Delimiter -
Two contiguous equal sign (=) characters used to delimit pseudo-text.

Qualification -
Method of uniquely referencing user-defined words or particular special registers. Performed by following word or register with OF or IN and appropriate qualifier.

Qualifier -
Word used to uniquely reference user-defined word or special register. Can be data-name, file-name, section-name, library-name, or report-name.

Queue -
A storage area for messages. A logical collection of messages stored in an area before being transmitted or delivered to a specified destination.

Queue (compound) -
A queue that has subqueues; a hierarchial structure.

Queue (input) -
A queue that accumulates messages acquired from external sources.

Queue (interprogram) -
A queue that accumulates messages sent by one program and destined for another program.

Queue (output) -
A queue that accumulates messages to be delivered to external destinations.

Queue (response) -
A queue that stores the MCS response to a message sent by a COBOL program.

Queue (simple) -
A queue that does not have subqueues. Output queues and interprogram queues are simple queues. Input queues can be either simple or compound queues.

Random Access -
Access mode that allows a nonsequential file on mass storage to be accessed by key value. Contrast with Sequential Access.

Random File -
In the context of CYBER Record Manager, a file with word addressable, indexed sequential, direct access, or actual key organization in which individual records can be accessed by the values of their keys. Contrast with Sequential File.

In the context of the NOS or NOS/BE operating systems, a file with the random bit set in the file information table in which individual records are accessed by their relative PRU numbers.

Realm -
File that is described by the user in a sub-schema. A named collection of data base records subject to access by a COBOL program.

Record -
In COBOL, unit of a file. Defined by level 01 Data Description entry.

CYBER Record Manager defines a record as a group of related characters. A record or a portion thereof is the smallest collection of information passed between CYBER Record Manager and a user program. Eight different record types exist, as defined by the RT field of the file information table.

Other parts of the operating systems and their products might have additional or different definitions of records.

Record Area -
Memory area used to process record. Named by record-name in level 01 Data Description entry.

Record Description Entry -
Set of Data Description entries that furnish information about the physical structure and identification of a record.

Record Key -
For indexed, direct, and actual-key files, primary key or alternate record key. For relative files, the key item defined by the RELATIVE KEY clause. For word-address files, key item defined by WORD-ADDRESS KEY clause.

Record Type -
The term record type can have one of several meanings, depending on the context. CYBER Record Manager defines eight record types established by an RT field in the file information table. Tables output by the loader are classified as record types such as text, relocatable, or absolute, depending on the first few words of the tables.

Reference Modification -
A method of referencing a data item at a position other than the first character position by specifying its leftmost character position and length.

Relation -
The logical structure formed by the joining of records based on common identifiers.

Relation Condition -
Type of conditional expression which compares the values of two operands to determine alternative paths of action.

Relative File -
File described by ORGANIZATION IS RELATIVE clause. Characterized by fixed length records with key values equivalent to ordinal positions of records in file.

Repeating Group -
A group data item which is described with an OCCURS clause or a group data item subordinate to a data item which is described with an OCCURS clause.

Report File -
In Report Writer, file whose FD entry contains a REPORT clause. Details of report are specified in RD entry in Report Section.

Reserved Word -
COBOL word specified in the list of words of appendix D which can be used in a COBOL source program, but which must not appear in program as user-defined words or system-names.

Root File -
The file that ranks lowest in a relation; its record occurrences are pictured at the root of a tree in a hierarchical tree structure.

Run Unit -
Main program and any subprogram it calls plus execution routines needed to execute the main and subprograms.

Schema -
A detailed description of the internal structure of a data base. A schema is not specified in the COBOL environment.

Section -
In COBOL, group of one or more paragraphs introduced by section header; predefined in Environment and Data Divisions, user-defined in Procedure Division.

Defined by BAM as a division within a file with sequential organization. Generally, a section contains more than one record and is a division within a partition of a file. A section terminates with a physical representation of a section boundary. See table C-5.

The NOS and NOS/BE operating systems equate a section with a system-logical-record of level 0 through 16_8.

Sentence -
One or more consecutive statements terminated by a separator period.

Sequential Access -
Access mode that allows a sequential or nonsequential file to be accessed by record position. Contrast with Random Access.

Sequential File -
File described by ORGANIZATION IS SEQUENTIAL clause. Characterized by access to records only by position of record in file. Can reside on magnetic tape or mass storage. Contrast with Random File.

Short PRU -
A PRU that does not contain as much user data as the PRU can hold and that is terminated by a system terminator with a level number.

Under NOS, a short PRU defines EOR.

Under NOS/BE, a short PRU defines the end of a system-logical-record. In the CYBER Record Manager context, a short PRU can have several interpretations depending on the record and blocking types.

Source -
A named physical device from which messages can be received under MCS.

Source Program -
Syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. Also known as program.

Sparse Key -
An alternate key that is used infrequently. Only those alternate key values of interest are included in the index file. The ALTERNATE RECORD KEY WITH DUPLICATES clause is used to specify keys of interest.

Statement -
Syntactically valid combination of words and symbols written in Procedure Division.

Static Program -
Program that is loaded with the base module. Contrast with Dynamic Program.

Subqueue -
A lower level queue in a queue hierarchy.

TABLE C-5. SECTION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Representation
PRU device	W	I	A deleted one-word record pointing back to the last I block boundary followed by a control word with flags indicating a section boundary. At least the control word is in a short PRU of level 0.
	W	C	A control word with flags indicating a section boundary. The control word is in a short PRU of level 0.
	D,F,R, T,U,Z	C	A short PRU with a level less than 17_8.
	S	-	Undefined.
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with flags indicating a section boundary.
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with flags indicating a section boundary.
	D,F,R, T,U,Z	C,K,E	Undefined.
	S	-	Undefined.
Any other tape format	-	-	Undefined.

Sub-Schema -
Description of part of a data base that is to be accessed by COBOL program. Created through Data Description Language; accessed through CYBER Database Control System interfaces. Items can be used in a program, but the description of the items is in the sub-schema rather than the program itself.

Sum Counter -
In Report Writer, signed numeric data item established by SUM clause. Contains result of summing operations.

Switch -
Software convention that can have the status OFF or ON. SWITCH-1 through SWITCH-126 can be defined in SPECIAL-NAMES paragraph. First six switches are external and can be manipulated by SWITCH control statement, terminal user, operator, or SET statement. Switches 7 through 126 are internal and can be manipulated by the SET statement.

Symbolic Name -
A user-defined name representing a source or destination; references queues, terminals, and programs. It is used in the Communication Section, Data Division of the COBOL program, and defined in the application definition.

System-Logical-Record -
Under NOS/BE, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. These records can be transferred between devices without loss of structure.

Equivalent to a logical record under NOS.

Equivalent to a CYBER Record Manager S type record.

Table -
Set of repeated items of data that is defined by OCCURS clause.

Text-Word -
A character or a sequence of contiguous characters in a COBOL library, source program or in pseudo-text, including separators (except for spaces, and pseudo-text delimiters) and literals, and excluding characters in comment lines.

Update -
Product running under operating system that allows a program library to be created in a special format which can be used by COPY statement processing. Described in Update reference manual.

User-Defined Word -
Word supplied by programmer to satisfy format of clause or statement. 1 through 30 characters A through Z, 0 through 9, or hyphen; hyphen cannot be first or last. Different types of words might have further restrictions for uniqueness, length, or characters.

Variable-Occurrence Data Item -
Data item described with OCCURS clause that has a DEPENDING ON phrase.

W Type Record -
One of the eight record types supported by CYBER Record Manager. Such records appear in storage preceded by a system-supplied control word. The existence of the control word allows files with sequential organization to have both partition and section boundaries.

Word-Address File -
File described by ORGANIZATION IS WORD-ADDRESS clause. Characterized by records identified by a key that indicates the relative word within mass storage file at which the record begins.

Zero-Byte Terminator -
12 bits of zero in the low order position of a word that marks the end of the line to be displayed at a terminal or printed on a line printer. The image of cards input through the card reader or terminal also has such a terminator.

Zero-Length PRU -
A PRU that contains system information, but no user data. Under CYBER Record Manager, a zero-length PRU of level 17 is a partition boundary. Under NOS, a zero-length PRU defines EOF.

RESERVED WORDS

ACCEPT
ACCESS
ACTUAL-KEY
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABET
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
APOSTROPHE
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BITS
BLANK
BLOCK
BOOLEAN
BOOLEAN-AND
BOOLEAN-EXOR
BOOLEAN-OR
BOTTOM
BY
CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMON-STORAGE
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION

CONTAINS
CONTINUE
CONTROL
CONTROLS
CONVERSION
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEADLOCK
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-NUMERIC-CONTENTS
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DIRECT
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EGI
ELSE
EMI
ENABLE
END
END-IF
END-OF-PAGE
END-PERFORM
END-SEARCH
ENDING
ENTER
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
ESI
EVERY
EXCEEDS
EXCEPTION
EXIT
EXTEND

EXTERNAL
FALSE
FD
FILE
FILE-CONTROL
FILES
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HASHED-VALUE
HASHING
HEADING
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE

MULTIPLE
MULTIPLY
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PURGE
QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
REALMS
RECEIVE
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE
REPLACING
REPORT
REPORTING

REPORTS	SEQUENTIAL	SYMBOLIC	UNTIL
RERUN	SET	SYNC	UP
RESERVE	SIGN	SYNCHRONIZED	UPON
RESET	SIZE	TABLE	USAGE
RETURN	SORT	TALLYING	USE
REVERSED	SORT-MERGE	TAPE	USING
REWIND	SOURCE	TERMINAL	VALUE
REWRITE	SOURCE-COMPUTER	TERMINATE	VALUES
RF	SPACE	TEST	VARYING
RH	SPACES	TEXT	WHEN
RIGHT	SPECIAL-NAMES	THAN	WITH
ROUNDED	STANDARD	THEN	WORD-ADDRESS
RUN	STANDARD-1	THROUGH	WORDS
SAME	START	THRU	WORKING-STORAGE
SD	STATUS	TIME	WRITE
SEARCH	STOP	TIMES	ZERO
SECONDARY-STORAGE	STRING	TO	ZEROES
SECTION	SUB-SCHEMA	TOP	ZEROS
SECURITY	SUB-QUEUE-1	TRACE--ON	+
SEGMENT	SUB-QUEUE-2	TRACE--OFF	-
SEGMENT-LIMIT	SUB-QUEUE-3	TRAILING	*
SELECT	SUBTRACT	TRUE	/
SEND	SUM	TYPE	**
SENTENCE	SUPERVISOR	UNEQUAL	>
SEPARATE	SUPPRESS	UNIT	<
SEQUENCE	SUSPEND	UNSTRING	=

LANGUAGE SUMMARY E

A summary of the language formats appears in this appendix. Detailed information for each format is referenced by page number. The following elements are alphabetized in one list:

- Division structure, by division name
- Section structure, by section name
- Paragraph header and contents, by paragraph name
- FD entry
- SD entry
- RD entry
- CD entry
- Data description entry
- Report group description entry
- Communication description entry
- Data Division clauses, by clause name
- Procedure Division statements, by statement name
- Condition type, by type
- Qualification
- Subscripting
- Indexing

Page

ACCEPT Statement Format 1 5-4

$$\underline{\text{ACCEPT}}\ \text{identifier}\ \ [\underline{\text{FROM}}\ \text{mnemonic-name}]$$

ACCEPT Statement Format 2 5-4

$$\underline{\text{ACCEPT}}\ \text{identifier}\ \underline{\text{FROM}}\left\{\begin{array}{l}\underline{\text{DATE}}\\ \underline{\text{DAY}}\\ \underline{\text{DAY-OF-WEEK}}\\ \underline{\text{TIME}}\end{array}\right\}$$

ACCEPT Statement Format 3 5-5

$$\underline{\text{ACCEPT}}\ \text{cd-name MESSAGE}\ \underline{\text{COUNT}}$$

ADD Statement Format 1 5-5

$$\underline{\text{ADD}}\left\{\begin{array}{l}\text{literal-1}\\ \text{identifier-1}\end{array}\right\}\left[\begin{array}{l}\text{, literal-2}\\ \text{, identifier-2}\end{array}\right]\ldots\ \underline{\text{TO}}\ \text{identifier-m}\ \ [\underline{\text{ROUNDED}}]\ \left[\text{, identifier-n}\ [\underline{\text{ROUNDED}}]\right]\ldots$$

$$[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

ADD Statement Format 2 5-6

$$\underline{\text{ADD}}\left\{\begin{array}{l}\text{literal-1}\\ \text{identifier-1}\end{array}\right\}\left\{\begin{array}{l}\text{, literal-2}\\ \text{, identifier-2}\end{array}\right\}\left[\begin{array}{l}\text{, literal-3}\\ \text{, identifier-3}\end{array}\right]\ldots\ \underline{\text{GIVING}}\ \text{identifier-m}\ \ [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-n}\ \ [\underline{\text{ROUNDED}}]\right]\ldots\ \ [;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

Page

ADD Statement Format 3 5-6

```
    { CORRESPONDING }
ADD {               }  identifier-1 TO identifier-2  [ROUNDED] [, identifier-3  [ROUNDED] . . .]
    { CORR          }

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

ALTER Statement 5-6

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

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

AUTHOR Paragraph of Identification Division 2-1

```
AUTHOR.  [comment-entry] . . .
```

BLANK WHEN ZERO Clause 4-12

```
BLANK WHEN ZERO
```

BLOCK CONTAINS Clause 4-3

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

CALL Statement 15-4

```
     { identifier }
CALL {            } [USING data-name-1   [, data-name-2] . . .]   [; ON OVERFLOW imperative-statement]
     { literal    }
```

CANCEL Statement 15-5

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

CD Entry in Communication Section

Format 1 4-25

```
CD  cd-name;              FOR     [INITIAL]     INPUT
  ┌                                                            ┐
  │  [[ ;  SYMBOLIC QUEUE IS data-name-1 ]                       │
  │        [ ; SYMBOLIC SUB-QUEUE-1        IS     data-name-2 ]  │
  │        [ ; SYMBOLIC SUB-QUEUE-2        IS     data-name-3 ]  │
  │        [ ; SYMBOLIC SUB-QUEUE-3        IS     data-name-4 ]  │
  │        [ ; MESSAGE DATE                IS     data-name-5 ]  │
  │        [ ; MESSAGE TIME                IS     data-name-6 ]  │
  │        [ ; SYMBOLIC SOURCE             IS     data-name-7 ]  │
  │        [ ; TEXT LENGTH                 IS     data-name-8 ]  │
  │        [ ; END KEY                     IS     data-name-9 ]  │
  │        [ ; STATUS KEY                  IS     data-name-10 ] │
  │        [ ; MESSAGE COUNT               IS     data-name-11 ]]│
  │ [data-name-1, data-name-2, . . . . . . data-name-11]         │
  └                                                            ┘
```

Format 2 4-29

```
CD    cd-name;       FOR      OUTPUT
  [ ;  DESTINATION  COUNT          IS          data-name-1 ]
  [ ;  TEXT LENGTH                 IS          data-name-2 ]
  [ ;  STATUS KEY                  IS          data-name-3 ]
       [ ;  DESTINATION TABLE OCCURS     integer-2         TIMES
         [ ;  INDEXED      BY     index-name-1    [ , index-name-2 ]   . . . ]]
  [ ;  ERROR KEY                              IS          data-name-4 ]
  [ ;  SYMBOLIC DESTINATION                   IS          data-name-5 ]
```

Class Condition 1-13

$$\text{identifier IS [NOT]} \left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{array}\right\}$$

Page

CLOSE Statement Format 1 5-7

```
                   ┌ ⎧REEL⎫ ┌WITH NO REWIND┐ ┐ ┌              ┌ ⎧REEL⎫ ┌WITH NO REWIND┐ ┐ ┐
                   │ ⎩UNIT⎭ └FOR REMOVAL   ┘ │ │              │ ⎩UNIT⎭ └FOR REMOVAL   ┘ │ │
CLOSE file-name-1  │                         │ │,file-name-2  │                         │ │
                   │        ⎧NO REWIND⎫      │ │              │        ⎧NO REWIND⎫      │ │
                   └ WITH   ⎩LOCK     ⎭      ┘ │              └ WITH   ⎩LOCK     ⎭      ┘ │
                                               └                                          ┘ ...
```

CLOSE Statement Format 2 14-3

```
CLOSE relation-name [WITH LOCK] . . .
```

CODE Clause 6-3

```
CODE literal
```

CODE-SET Clause 4-4

```
CODE-SET IS alphabet-name
```

COLUMN NUMBER Clause 6-6

```
COLUMN NUMBER IS integer
```

Common-Storage Section of Data Division 15-1

```
COMMON-STORAGE SECTION.

 [data-description-entry] . . .
```

Communication Section of Data Division 4-3

```
COMMUNICATION SECTION.
    [communication-description-entry [record-description entry] . . .] . . .
```

COMPUTE Statement Format 1 5-8

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] . . .

    ⎧FROM  ⎫
    ⎨ =    ⎬ arithmetic-expression  [; ON SIZE ERROR imperative-statement]
    ⎩EQUALS⎭
```

COMPUTE Statement Format 2 5-8

```
COMPUTE {,identifier-3} . . .

    ⎧FROM  ⎫
    ⎨ =    ⎬ boolean-expression
    ⎩EQUALS⎭
```

Page

Condition-Name Condition 1-13

```
condition-name
```

Configuration Section of Environment Division 3-1

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

CONTINUE Statement 5-8

```
CONTINUE
```

CONTROL Clause 6-3

```
{CONTROL IS    }  {data-name-1  [, data-name-2] . . .                  }
{CONTROLS ARE  }  {FINAL  [, data-name-1  [, data-name-2] . . .]       }
```

COPY Statement 9-1

```
COPY text-name  [{OF} library-name]
                [{IN}             ]

[              { {==pseudo-text-1==}      {==pseudo-text-2==} }       ]
[  REPLACING   {,{identifier-1     }  BY  {identifier-2     } } . . . ].
[              { {literal-1        }      {literal-2        } }       ]
[              { {word-1           }      {word-2           } }       ]
```

Data Description Entry Format 1 4-11

```
level-number  [{data-name}]  [; REDEFINES data-name-2]
              [{FILLER   }]
```

```
[; BLANK WHEN ZERO]
```
4-12

```
[ ; {JUSTIFIED}  RIGHT ]
[   {JUST     }        ]
```
4-13

```
[  / OCCURS integer-1 TIMES                                                \ ]
[ |                                                                         | ]
[ |     [{ASCENDING }  KEY IS data-name-1 [, 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] . . .]                   /  ]
```
4-13

Page

```
[ ; { PICTURE }  IS character-string ]                        4-14
    { PIC     }

[ ;[SIGN IS] { LEADING  } [SEPARATE CHARACTER] ]              4-21
             { TRAILING }

[ ; { SYNCHRONIZED }  [ LEFT  ] ]                             4-21
    { SYNC         }  [ RIGHT ]

[                  { COMPUTATIONAL   } ]                      4-22
[                  { COMP            } ]
[                  { COMPUTATIONAL-1 } ]
[                  { COMP-1          } ]
[ ;[USAGE IS]      { COMPUTATIONAL-2 } ]
[                  { COMP-2          } ]
[                  { COMPUTATIONAL-4 } ]
[                  { COMP-4          } ]
[                  { DISPLAY         } ]
[                  { INDEX           } ]

[;VALUE IS literal].                                          4-24
```

Data Description Entry Format 2 — 4-12

```
66 data-name-1; RENAMES data-name-2 [ { THRU    } data-name-3 ].     4-20
                                      { THROUGH }
```

Data Description Entry Format 3 — 4-12

```
88 condition-name; { VALUE IS   } literal-1 [ { THRU    } literal-2 ]     4-24
                   { VALUES ARE }             { THROUGH }

    [ , literal-3 [ { THRU    } literal-4 ] ] . . . .
                    { THROUGH }
```

Data Division — 4-1

```
DATA DIVISION.
[File Section]
[Common-Storage Section]
[Working-Storage Section]
[Secondary-Storage Section]
[Linkage Section]
[Communication Section]
[Report Section]
```

DATA RECORDS Clause — 4-5

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

Page

DATE-COMPILED Paragraph of Identification Division 2-1

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

DATE-WRITTEN Paragraph of Identification Division 2-1

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

Declaratives Portion of Procedure Division 5-1

```
DECLARATIVES.
{ section-name  SECTION [segment-number].  declarative-sentence.

[ paragraph-name.  [sentence] . . . ] . . . } . . .
END DECLARATIVES.
```

DELETE Statement 5-8

```
DELETE  {file-name  RECORD [; INVALID KEY imperative-statement]}
        {FILE  {file-name}  . . .                              }
```

DISABLE Statement 5-9

```
DISABLE {INPUT      [TERMINAL]}  cd-name WITH KEY  {identifier-1}
        {OUTPUT               }                    {literal-1   }
```

DISPLAY Statement 5-9

```
DISPLAY {literal-1   }  [, literal-2   ]
        {identifier-1}  [, identifier-2] . . .  [UPON mnemonic-name]  [WITH NO ADVANCING]
```

DIVIDE Statement Format 1 5-11

```
DIVIDE {identifier-1}  INTO identifier-2 [ROUNDED]  [, identifier-3 [ROUNDED]] . . .
       {literal-1   }

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

DIVIDE Statement Format 2 5-11

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

    [, identifier-4 [ROUNDED]] . . . [; ON SIZE ERROR imperative-statement]
```

DIVIDE Statement Format 3 5-11

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

    [, identifier-4 [ROUNDED]] . . . [; ON SIZE ERROR imperative-statement]
```

Page

DIVIDE Statement Format 4 5-11

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

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

DIVIDE Statement Format 5 5-12

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

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

ENABLE Statement 5-12

```
ENABLE {INPUT    [TERMINAL]} cd-name WITH KEY {identifier-1}
       {OUTPUT             }                  {literal-1   }
```

ENTER Statement 15-5

```
                                        [       {data-name-1        } [, data-name-2        ]     ]
ENTER [COMPASS  ] routine-name          [ USING {file-name-1        } [, file-name-2        ]     ]
      [FORTRAN-X]                       [       {procedure-name-1   } [, procedure-name-2   ]     ]
                                        [       {literal-1          } [, literal-2          ] ... ]
```

Environment Division 3-1

```
ENVIRONMENT DIVISION.

[Configuration Section]

[Input-Output Section]
```

EXIT Statement 5-12

```
EXIT [PROGRAM].
```

EXTERNAL Clause 4-5

```
EXTERNAL
```

FD Entry in File Section 4-3

```
FD file-name

    [; BLOCK CONTAINS [integer-1 TO] integer-2 [{RECORDS   }]]          4-3
                                               [{CHARACTERS}]

    [; CODE-SET IS alphabet-name]                                       4-4

    [; DATA {RECORD IS  } data-name-1  [, data-name-2] . . .]           4-5
            {RECORDS ARE}

    [; EXTERNAL]                                                        4-5
```

Page

```
; LABEL { RECORDS ARE } { STANDARD }                                              4-5
        { RECORD IS   } { OMITTED  }

   [ ; VALUE OF implementor-name-1 IS { data-name-1 }
                                      { literal-1   }

       [ , implementor-name-2 IS { data-name-2 } ] . . . ]
                                 { literal-2   }

[ ; LINAGE IS { data-name-1 } LINES [ , WITH FOOTING AT { data-name-2 } ]          4-7
              { integer-1   }                           { integer-2   }

  [ , LINES AT TOP { data-name-3 } ] [ , LINES AT BOTTOM { data-name-4 } ] ]
                   { integer-3   }                       { integer-4   }

[ { ; RECORD CONTAINS [integer-1 TO]                                      } ]      4-7
  {   integer-2 CHARACTERS   [DEPENDING ON data-name]                     }
  { ; RECORD IS VARYING IN SIZE  [[FROM integer-1]                        }
  {   [TO integer-2] CHARACTERS]  [DEPENDING ON data-name]                 }

[ ; RECORDING MODE IS { DECIMAL } ]                                                4-10
                      { BINARY  }

[ ; { REPORT IS   } report-name-1   [, report-name-2] . . . ]                      6-2
    { REPORTS ARE }

[record-description-entry] . . .                                                   4-10
```

File-Control Entry Format 1 (Sequential File Organization) 3-7

```
SELECT [OPTIONAL] file-name                                                        3-16

    ASSIGN TO implementor-name-1   [, implementor-name-2] . . .                    3-12

    [; ORGANIZATION IS SEQUENTIAL]                                                 3-15

    [; ACCESS MODE IS SEQUENTIAL]                                                  3-10

    [; FILE STATUS IS data-name]                                                   3-13

    [ ; RESERVE integer [ AREA  ] ]                                                3-16
                        [ AREAS ]

    [; USE literal]                                                                3-16
```

```
                                                                              Page
File-Control Entry Format 2 (Relative File Organization)                      3-7

    SELECT file-name                                                           3-16

        ASSIGN TO implementor-name-1 [, implementor-name-2] . . .              3-12

        ; ORGANIZATION IS RELATIVE                                             3-15

        [                     { SEQUENTIAL [,RELATIVE KEY IS data-name]   } ]
        [ ; ACCESS MODE IS    {                                           } ]  3-10
        [                     { { RANDOM  } , RELATIVE KEY IS data-name   } ]
        [                     { { DYNAMIC }                               } ]

        [; FILE STATUS IS data-name]                                           3-13

        [ ; RESERVE integer [AREA ] ]                                          3-16
        [                   [AREAS] ]

        [; USE literal].                                                       3-16


File-Control Entry Format 3 (Indexed File Organization)                        3-7

    SELECT file-name                                                           3-16

        ASSIGN TO implementor-name-1 [, implementor-name-2] . . .              3-12

        ; ORGANIZATION IS INDEXED                                              3-15

        ; RECORD KEY IS data-name                                              3-15

        [                     { SEQUENTIAL } ]
        [ ; ACCESS MODE IS    { RANDOM     } ]                                 3-10
        [                     { DYNAMIC    } ]

        [ ;ALTERNATE RECORD KEY IS data-name-1 [WITH DUPLICATES [ASCENDING]]  ] 3-10
        [                                                                     ]
        [   [ { OMITTED }                                                 ]   ]
        [   [ { USE     }  WHEN data-name-2 CONTAINS CHARACTER FROM literal]   ]
        [                                                                     ]
        [   [ OMITTED WHEN KEY IS { SPACES } ]                                ]
        [   [                     { ZEROS  } ]                                ]

        [; FILE STATUS IS data-name]                                           3-13

        [; USE literal].                                                       3-16
```

```
                                                                          Page
File-Control Entry Format 4 (Direct File Organization)                    3-8

  SELECT file-name                                                        3-16

     ASSIGN TO implementor-name-1 [, implementor-name-2] . . .            3-12

     ; ORGANIZATION IS DIRECT                                             3-15

                     { integer   }
     ; BLOCK COUNT IS {           }                                       3-13
                     { data-name }

     ; RECORD KEY IS data-name                                            3-15

     [                   { SEQUENTIAL } ]
     [ ; ACCESS MODE IS  { RANDOM     } ]                                 3-10
     [                   { DYNAMIC    } ]

     [ ; ALTERNATE RECORD KEY IS data-name-1 [WITH DUPLICATES [ASCENDING]] ]   3-10
     [                                                                      ]
     [   [ { OMITTED }                                                ]     ]
     [   [ { USE     } WHEN data-name-2 CONTAINS CHARACTER FROM literal ]   ]
     [                                                                      ]
     [   [                          { SPACES } ]                            ]
     [   [ OMITTED WHEN KEY IS      { ZEROS  } ]                            ]

     [; FILE STATUS IS data-name]                                         3-13

     [; USE literal].                                                     3-16


File-Control Entry Format 5 (Actual-Key File Organization)                3-8

  SELECT file-name                                                        3-16

     ASSIGN TO implementor-name-1 [, implementor-name-2] . . .            3-12

     ; ORGANIZATION IS ACTUAL-KEY                                         3-15

     ; RECORD KEY IS data-name                                            3-15

     [                   { SEQUENTIAL } ]
     [ ; ACCESS MODE IS  { RANDOM     } ]                                 3-10
     [                   { DYNAMIC    } ]

     [ ; ALTERNATE RECORD KEY IS data-name-1 [WITH DUPLICATES [ASCENDING]] ]   3-10
     [                                                                      ]
     [   [ { OMITTED }                                                ]     ]
     [   [ { USE     } WHEN data-name-2 CONTAINS CHARACTER FROM literal ]   ]
     [                                                                      ]
     [   [                          { SPACES } ]                            ]
     [   [ OMITTED WHEN KEY IS      { ZEROS  } ]                            ]

     [; FILE STATUS IS data-name]                                         3-13

     [; USE literal].                                                     3-16
```

Page

File-Control Entry Format 6 (Word-Address File Organization) 3-9

```
SELECT file-name                                                3-16
    ASSIGN TO implementor-name-1  [, implementor-name-2] . . .  3-12
    ; ORGANIZATION IS WORD-ADDRESS                              3-15
    ; WORD-ADDRESS KEY IS data-name
    [; ACCESS MODE IS {SEQUENTIAL}]                             3-10
    [                 {RANDOM    }]
    [                 {DYNAMIC   }]
    [ ; FILE STATUS IS data-name]                               3-13
    [; RESERVE integer [AREA ]]                                 3-16
    [                  [AREAS]]
    [ ; USE literal].                                           3-16
```

FILE-CONTROL Paragraph in Input-Output Section 3-5

```
FILE-CONTROL.  [file-control-entry] . . .
```

File Section of Data Division 4-1

```
FILE SECTION.
[FD entry] . . .
[SD entry] . . .
```

GENERATE Statement 6-11

```
GENERATE {data-name  }
         {report-name}
```

GO TO Statement Format 1 5-13

```
GO TO [ procedure-name-1]
```

GO TO Statement Format 2 5-13

```
GO TO procedure-name-1  [, procedure-name-2] . . . , procedure-name-n DEPENDING ON identifier
```

GROUP INDICATE Clause 6-6

```
GROUP INDICATE
```

Page

Identification Division 2-1

```
IDENTIFICATION DIVISION.

PROGRAM-ID paragraph

[AUTHOR paragraph]

[INSTALLATION paragraph]

[DATE-WRITTEN paragraph]

[DATE-COMPILED paragraph]

[SECURITY paragraph]
```

IF Statement 5-13

```
IF condition; THEN { [, statement-1] . . . }  { ; ELSE  [statement-2] . . .  [; END-IF] }
                   {  NEXT SENTENCE        }  { ; ELSE NEXT SENTENCE                   }
                                              { ; END-IF                               }
```

Indexing 1-18

```
{ data-name      }  ( { index-name-1  [{±}  literal-2] }
{ condition-name }    { literal-1                      }

  [ , { index-name-2  [{±}  literal-4] }  [ , { index-name-3  [{±}  literal-6] } ] ] )
  [   { literal-3                      }  [   { literal-5                      } ] ]
```

INITIALIZE Statement 5-14

```
INITIALIZE identifier-1  [, identifier-2] . . .

  [             { ALPHABETIC           }                              ]
  [             { ALPHANUMERIC         }                              ]
  [ REPLACING   { NUMERIC              }  DATA BY { identifier-3 }    ]
  [             { ALPHANUMERIC-EDITED  }          { literal      }    ]
  [             { NUMERIC-EDITED       }                              ]
  [             { BOOLEAN              }                              ]
```

INITIATE Statement 6-11

```
INITIATE report-name-1  [, report-name-2] . . .
```

Input-Output Section Header in Environment Division 3-5

```
INPUT-OUTPUT SECTION.

[FILE-CONTROL paragraph]

[I-O-CONTROL paragraph]
```

Page

INSPECT Statement Format 1 5-16

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{TALLYING}} \left\{ \text{,identifier-2}\ \underline{\text{FOR}} \left\{ \text{,} \left\{ \begin{array}{l} \begin{Bmatrix}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\end{Bmatrix} \begin{Bmatrix}\text{literal-1}\\ \text{identifier-3}\end{Bmatrix} \\ \underline{\text{CHARACTERS}} \end{array} \right\} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-2}\\ \text{identifier-4}\end{Bmatrix} \right] \right\} \dots \right\} \dots$$

INSPECT Statement Format 2 5-16

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{REPLACING}} \left\{ \begin{array}{l} \underline{\text{CHARACTERS}}\ \underline{\text{BY}} \begin{Bmatrix}\text{literal-4}\\ \text{identifier-6}\end{Bmatrix} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-5}\\ \text{identifier-7}\end{Bmatrix} \right] \\ \left\{ \text{,} \begin{Bmatrix}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{FIRST}}\end{Bmatrix} \left\{ \text{,} \begin{Bmatrix}\text{literal-3}\\ \text{identifier-5}\end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix}\text{literal-4}\\ \text{identifier-6}\end{Bmatrix} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-5}\\ \text{identifier-7}\end{Bmatrix} \right] \right\} \dots \right\} \dots \end{array} \right\}$$

INSPECT Statement Format 3 5-16

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{TALLYING}} \left\{ \text{,identifier-2}\ \underline{\text{FOR}} \left\{ \text{,} \left\{ \begin{array}{l} \begin{Bmatrix}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\end{Bmatrix} \begin{Bmatrix}\text{literal-1}\\ \text{identifier-3}\end{Bmatrix} \\ \underline{\text{CHARACTERS}} \end{array} \right\} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-2}\\ \text{identifier-4}\end{Bmatrix} \right] \right\} \dots \right\} \dots$$

$$\begin{bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{bmatrix} \underline{\text{REPLACING}} \left\{ \begin{array}{l} \underline{\text{CHARACTERS}}\ \underline{\text{BY}} \begin{Bmatrix}\text{literal-4}\\ \text{identifier-6}\end{Bmatrix} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-5}\\ \text{identifier-7}\end{Bmatrix} \right] \\ \left\{ \text{,} \begin{Bmatrix}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{FIRST}}\end{Bmatrix} \left\{ \text{,} \begin{Bmatrix}\text{literal-3}\\ \text{identifier-5}\end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix}\text{literal-4}\\ \text{identifier-6}\end{Bmatrix} \left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix} \text{INITIAL} \begin{Bmatrix}\text{literal-5}\\ \text{identifier-7}\end{Bmatrix} \right] \right\} \dots \right\} \dots \end{array} \right\}$$

Page

INSTALLATION Paragraph of Identification Division 2-1

```
INSTALLATION. [comment-entry] . . .
```

I-O CONTROL Paragraph of Input-Output Section 3-17

```
I-O-CONTROL.

[; APPLY input-output-technique ON file-name-1 [, file-name-2] . . .]           3-17

[; MULTIPLE FILE TAPE CONTAINS { file-name-1        } [POSITION integer-1]   ]   3-17
[                              { pseudo-file-name-1 }                        ]
[                                                                            ]
[   [, { file-name-2        } [POSITION integer-2]] . . .                    ] . . .
[      { pseudo-file-name-2 }                                                ]

[        [{ RECORD     }]                                              ]
[; SAME  [{ SORT       }] AREA FOR file-name-1 {, file-name-2} . . .  ] . . .   3-19
[        [{ SORT-MERGE }]                                              ]

[                                           { { [END OF] { REEL } }                    } ]
[; RERUN [ON { file-name-1         }] EVERY { {          { UNIT } } OF file-name-2     } ] . . .   3-18
[            { implementor-name    }        { { integer-1 RECORDS }                    } ]
[                                           { condition-name                           } ]
```

JUSTIFIED Clause 4-13

```
{ JUSTIFIED } RIGHT
{ JUST      }
```

LABEL RECORDS Clause 4-5

```
LABEL { RECORDS ARE } { STANDARD }
      { RECORD IS   } { OMITTED  }

    [; VALUE OF implementor-name-1 IS { data-name-1 }
    [                                 { literal-1   }

        [, implementor-name-2 IS { data-name-2 } ] . . . ]
                                 { literal-2   }
```

LINAGE Clause 4-7

```
LINAGE IS { data-name-1 } LINES [, WITH FOOTING AT { data-name-2 } ]
          { integer-1   }                          { integer-2   }

    [, LINES AT TOP { data-name-3 } ]  [, LINES AT BOTTOM { data-name-4 } ]
                    { integer-3   }                       { integer-4   }
```

Page

LINE NUMBER Clause 6-7

```
               { integer-1 [ON NEXT PAGE] }
LINE NUMBER IS {                          }
               { PLUS integer-2           }
```

Linkage Section of Data Division 15-1

```
LINKAGE SECTION.

[data description entry] . . .
```

MERGE Statement 7-4

```
                     { DESCENDING }
MERGE file-name-1 ON {            } KEY data-name-1 [, data-name-2] . . .
                     { ASCENDING  }

    [    { DESCENDING }                                 ]
    [ ON {            } KEY data-name-3 [, data-name-4] . . . ] . . .
    [    { ASCENDING  }                                 ]

[COLLATING SEQUENCE IS alphabet-name]

USING file-name-2, file-name-3 [, file-name-4] . . .

{ OUTPUT PROCEDURE IS section-name-1 [ { THRU    } section-name-2 ] }
{                                    [ { THROUGH }                ] }
{ GIVING file-name-5                                                }
```

MOVE Statement Format 1 5-18

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

MOVE Statement Format 2 5-18.1

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

MULTIPLY Statement Format 1 5-19

```
         { identifier-1 }
MULTIPLY {              } BY identifier-2 [ROUNDED] [, identifier-3 [ROUNDED]] . . .
         { literal-1    }

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

MULTIPLY Statement Format 2 5-19

```
         { identifier-1 }    { identifier-2 }
MULTIPLY {              } BY {              } GIVING identifer-3 [ROUNDED]
         { literal-1    }    { literal-2    }

    [, identifier-4 [ROUNDED]] . . . [; ON SIZE ERROR imperative-statement]
```

Page

NEXT GROUP Clause 6-7

```
                 ⎧ integer-1       ⎫
NEXT GROUP IS    ⎨ PLUS integer-2  ⎬
                 ⎩ NEXT PAGE       ⎭
```

OBJECT-COMPUTER Paragraph of Configuration Section 3-1

```
OBJECT-COMPUTER.  ⎡ computer-name                                             ⎤
                  ⎢                                                           ⎥
                  ⎢ [, PROGRAM COLLATING SEQUENCE IS alphabet-name]           ⎥
                  ⎢                                                           ⎥
                  ⎣ [, SEGMENT-LIMIT IS segment-number]                       ⎦
```

OCCURS Clause Format 1 4-13

```
OCCURS integer-1  TIMES  ⎡⎧ ASCENDING  ⎫ KEY IS data-name-1 [, data-name-2] . . .⎤ . . .
                         ⎣⎩ DESCENDING ⎭                                         ⎦

    [INDEXED BY index-name-1 [, index-name-2] . . .]
```

OCCURS Clause Format 2 4-14

```
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] . . .]
```

OPEN Statement Format 1 5-19

```
       ⎧ INPUT file-name-1 ⎡REVERSED        ⎤ ⎡, file-name-2 ⎡REVERSED        ⎤⎤ . . . ⎫
       ⎪                   ⎣WITH NO REWIND  ⎦ ⎣              ⎣WITH NO REWIND  ⎦⎦       ⎪
OPEN   ⎨ OUTPUT file-name-3 [WITH NO REWIND] [, file-name-4 [WITH NO REWIND]] . . .    ⎬ . . .
       ⎪ I-O file-name-5 [, file-name-6] . . .                                          ⎪
       ⎩ EXTEND file-name-7 [, file-name-8] . . .                                       ⎭
```

Page

OPEN Statement Format 2 14-3

```
OPEN  { INPUT relation-name [WITH NO REWIND] . . . }
      { I-O relation-name                          } . . .
```

PAGE Clause 6-3

```
PAGE  [LIMIT IS  ]  integer-1  [LINE ]  [, HEADING integer-2]
      [LIMITS ARE]             [LINES]

     [, FIRST DETAIL integer-3]  [, LAST DETAIL integer-4]

     [, FOOTING integer-5]
```

PERFORM Statement Format 1 5-22

```
PERFORM [procedure-name-1  [{ THRU    }  procedure-name-2]] [; imperative-statement ; END-PERFORM]
                           [{ THROUGH }                 ]
```

PERFORM Statement Format 2 5-22

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

     [; imperative-statement ; END-PERFORM]
```

PERFORM Statement Format 3 5-22

```
PERFORM [procedure-name-1  [{ THRU    }  procedure-name-2]] [; WITH TEST { BEFORE }]
                           [{ THROUGH }                 ]   [            { AFTER  }]

     UNTIL condition-1  [; imperative-statement ; END-PERFORM]
```

PERFORM Statement Format 4 5-22

```
PERFORM [procedure-name-1  [{ THRU    }  procedure-name-2]]  [; WITH TEST { BEFORE }]
                           [{ THROUGH }                 ]    [            { AFTER  }]

     VARYING { identifier-1 }  FROM { identifier-2 }  BY { identifier-3 }  UNTIL condition-1
             { index-name-1 }       { index-name-2 }     { literal-2    }
                                    { literal-1    }

      [AFTER { identifier-4 }  FROM { identifier-5 }  BY { identifier-6 }  UNTIL condition-2] . . .
      [      { index-name-3 }       { index-name-4 }     { literal-4    }                   ]
      [                             { literal-3    }                                        ]

          [; imperative-statement ; END-PERFORM]
```

Page

PICTURE Clause 4-14

```
{ PICTURE }
{ PIC     } IS character-string
```

Procedure Division Format 1 5-1

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

Procedure Division Format 2 5-1

```
PROCEDURE DIVISION [USING data-name-1  [, data-name-2] . . .] .
{ paragraph-name.  [sentence] . . .} . . .
```

PURGE Statement 5-23

```
PURGE cd-name
```

PROGRAM-ID Paragraph of Identification Division 2-1

```
PROGRAM-ID.  program-name.
```

Qualification Format 1 1-16

```
                      ( { { IN }  data-name-2 } . . . [ { IN }  file-name ] )
{ data-name-1    }    (   { OF }                        { OF }  cd-name     )
{ condition-name }    (                                                     )
                      ( [ { IN }  file-name ]                               )
                      (   { OF }  cd-name                                   )
```

Qualification Format 2 1-16

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

Page

Qualification Format 3 1-16

```
              ⎡⎧ IN ⎫                 ⎤
text-name     ⎣⎩ OF ⎭   library-name  ⎦
```

Qualification Format 4 1-16

```
                ⎡⎧ IN ⎫            ⎤
LINAGE-COUNTER  ⎣⎩ OF ⎭  file-name ⎦
```

Qualification Format 5 1-16

```
⎧ PAGE-COUNTER ⎫ ⎡⎧ IN ⎫              ⎤
⎩ LINE-COUNTER ⎭ ⎣⎩ OF ⎭  report-name ⎦
```

Qualification Format 6 1-16

```
             ⎧⎡⎧ IN ⎫              ⎤       ⎡⎧ IN ⎫              ⎤⎫
             ⎪⎣⎩ OF ⎭  data-name-4 ⎦ . . . ⎣⎩ OF ⎭  report-name ⎦⎪
data-name-3  ⎨                                                   ⎬
             ⎪⎡⎧ IN ⎫              ⎤                              ⎪
             ⎩⎣⎩ OF ⎭  report-name ⎦                              ⎭
```

RD Entry in Report Section 6-2

```
RD report-name

   [; CODE literal]

   ⎡ ;⎧ CONTROL IS   ⎫ ⎧ data-name-1  [, data-name-2] . . .              ⎫⎤
   ⎣  ⎩ CONTROLS ARE ⎭ ⎩ FINAL  [, data-name-1  [, data-name-2] . . .]  ⎭⎦

   ⎡        ⎡ LIMIT IS   ⎤            ⎡ LINE  ⎤                        ⎤
   ⎢; PAGE  ⎣ LIMITS ARE ⎦  integer-1 ⎣ LINES ⎦  [, HEADING integer-2] ⎥   6-3
   ⎢                                                                   ⎥
   ⎢   [, FIRST DETAIL integer-3]  [, LAST DETAIL integer-4]           ⎥
   ⎢                                                                   ⎥
   ⎣   [, FOOTING integer-5]                                           ⎦

   { report-group-description entry } . . .                                6-4
```

READ Statement Format 1 5-24

```
READ file-name [NEXT] RECORD [INTO identifier]  [; AT END imperative-statement]
```

READ Statement Format 2 5-25

```
READ file-name RECORD  [INTO identifier]  [; KEY IS data-name]  [; INVALID KEY imperative-statement]
```

READ Statement Format 3 14-2

```
READ relation-name [NEXT] RECORD [; AT END imperative-statement]
```

Page

READ Statement Format 4 14-2

```
READ relation-name RECORD [; KEY IS data-name] [; INVALID KEY imperative-statement]
```

RECEIVE Statement 5-25

```
RECEIVE cd-name { MESSAGE }  INTO identifier-1  [; NO DATA imperative-statement]
                { SEGMENT }
```

RECORD Clause Format 1 4-7

```
RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS [; DEPENDING ON data-name]
```

RECORD Clause Format 2 4-7

```
RECORD IS VARYING IN SIZE [[FROM integer-1] [TO integer-2] CHARACTERS] [DEPENDING ON data-name]
```

RECORDING MODE Clause 4-10

```
RECORDING MODE IS { DECIMAL }
                  { BINARY  }
```

REDEFINES Clause 4-19

```
REDEFINES data-name-2
```

Reference Modification 1-18

```
data-name (leftmost-character-position : { length } )
                                         { END    }
```

Relation Condition Format 1 1-13

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

Relation Condition Format 2 1-13

```
                     { IS [NOT] EQUAL TO }
boolean expression-1 { IS [NOT] =        } boolean expression-2
                     { IS UNEQUAL TO     }
                     { EQUALS            }
```

RELEASE Statement 7-5

```
RELEASE record-name [FROM identifier]
```

Page

RENAMES Clause 4-20

```
RENAMES data-name-2  [ { THRU    }  data-name-3 ]
                       { THROUGH }
```

REPLACE Statement Format 1 9-2

```
REPLACE { , ==pseudo-text-1==  BY  ==pseudo-text-2== } . . .
```

REPLACE Statement Format 2 9-3

```
REPLACE OFF
```

REPORT Clause 6-2

```
{ REPORT IS    }  report-name-1 [, report-name-2] . . .
{ REPORTS ARE  }
```

Report Group Description Entry Format 1 6-4

```
01 [data-name-1]

  [ ; LINE NUMBER IS { integer-1 [ON NEXT PAGE] } ]                6-7
                     { PLUS integer-2           }

  [ ; NEXT GROUP IS  { integer-1      } ]                          6-7
                     { PLUS integer-2 }
                     { NEXT PAGE      }

                ( { REPORT HEADING  }                          )
                ( { RH              }                          )
                ( { PAGE HEADING    }                          )
                ( { PH              }                          )
                ( { CONTROL HEADING }   { data-name-1 }        )
                ( { CH              }   { FINAL       }        )
   ; TYPE IS    ( { DETAIL }                                   )   6-9
                ( { DE     }                                   )
                ( { CONTROL FOOTING }   { data-name-2 }        )
                ( { CF              }   { FINAL       }        )
                ( { PAGE FOOTING    }                          )
                ( { PF              }                          )
                ( { REPORT FOOTING  }                          )
                ( { RF              }                          )

  [ ;[USAGE IS] DISPLAY ] .                                        6-11
```

Report Group Description Entry Format 2 6-5

```
level-number [data-name]

  [ ; LINE NUMBER IS { integer-1 [ON NEXT PAGE] } ]                6-7
                     { PLUS integer-2           }

  [ ; [USAGE IS] DISPLAY ] .                                       6-11
```

Page

Report Group Description Entry Format 3 6-5

```
level-number [data-name]

    [; BLANK WHEN ZERO]                                                    6-6

    [; COLUMN NUMBER IS integer]                                           6-6

    [; GROUP INDICATE]                                                     6-6

    [; {JUSTIFIED} RIGHT]                                                  4-13
       {JUST     }

    [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]                          6-7
                      {PLUS integer-2          }

    ; {PICTURE}  IS character-string                                       4-14
      {PIC    }

    {; SOURCE IS identifier                                          }     6-8
    {; VALUE IS literal                                              }     6-11
    {; {SUM identifier-1 [, identifier-2] . . . [UPON data-name-1  [, data-name-2] . . . ]} . . . }  6-8
    {   [RESET ON {data-name-3}]                                     }
    {             {FINAL      }                                      }

    [; [USAGE IS]  DISPLAY]                                                6-11
```

Report Section of Data Division 6-2

```
REPORT SECTION.
{RD entry}
[report group description entry] . . .
```

RETURN Statement 7-6

```
RETURN file-name RECORD [INTO identifier] ; AT END imperative-statement
```

REWRITE Statement 5-26

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

Page

SD Entry of File Section 7-1

```
SD file-name

  [          (CONTAINS [integer-1 TO] integer-2 CHARACTERS )]
  [ ; RECORD )IS VARYING IN SIZE [[FROM integer-3]         (]   4-7
  [          )[TO integer-4] CHARACTERS]                   (]
  [          ([DEPENDING ON data-name-1]                   )]

  [ ; DATA {RECORD IS  } data-name-1 [, data-name-2] . . . ] .   4-5
  [        {RECORDS ARE}                                   ]

  [record-description-entry] . . .
```

SEARCH Statement Format 1 5-28

```
SEARCH identifier-1 [VARYING {index-name-1}] [; AT END imperative-statement-1]
                    [        {identifier-2}]

   ; WHEN condition-1 {imperative-statement-2} [; WHEN condition-2 {imperative-statement-3}] . . .
                      {NEXT SENTENCE         } [                   {NEXT SENTENCE         }]
    [; END-SEARCH]
```

SEARCH Statement Format 2 5-28

```
SEARCH ALL identifier-1 [; AT END imperative-statement-1]

          ( data-name-1  (EQUALS      ) ( identifier-3             ) )
   ; WHEN (              (IS EQUAL TO ) ( literal-2                ) )
          (              (IS =        ) ( arithmetic-expression-1  ) )
          (                                                          )
          ( condition-name-1                                         )

   [      ( data-name-2  (EQUALS      ) ( identifier-4             ) )]
   [ AND  (              (IS EQUAL TO ) ( literal-3                ) )] . . .
   [      (              (IS =        ) ( arithmetic-expression-2  ) )]
   [      ( condition-name-2                                         )]

   {imperative-statement-2} [; END-SEARCH]
   {NEXT SENTENCE         }
```

Secondary-Storage Section of Data Division 4-2

```
SECONDARY-STORAGE SECTION.
[data description entry] . . .
```

SECURITY Paragraph of Identification Division 2-1

```
SECURITY. [comment-entry] . . .
```

SEND Statement Format 1 5-29

```
SEND cd-name FROM identifier-1
```

Page

SEND Statement Format 2 5-29

```
                                        ( WITH identifier-2 )
SEND cd-name [FROM identifier-1]       <  WITH ESI           >
                                        ( WITH EMI           )
                                        ( WITH EGI           )
    [                                                                   ]
    [ { BEFORE }              ( { identifier-3 }      [LINE ] )         ]
    [ { AFTER  }  ADVANCING  <  { integer      }      [LINES] >         ]
    [                         ( { mnemonic-name }             )         ]
    [                         ( { PAGE          }             )         ]
```

SET Statement Format 1 5-31

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

SET Statement Format 2 5-31

```
                                            { UP BY   }  { identifier-4 }
SET index-name-4  [, index-name-5] . . .    { DOWN BY }  { integer-2    }
```

SET Statement Format 3 5-32

```
      ( { SORT       }                      )
      ( { MERGE      }  COLLATING SEQUENCE  )
      ( { SORT-MERGE }                      )
SET  <  { PROGRAM    }                       >  TO alphabet-name
      (                                     )
      (  CODE-SET FOR { file-name-1  [, file-name-2] . . . } )
      (               { ALL FILES                          } )
```

SET Statement Format 4 5-32

```
SET { mnemonic-name-1  [, mnemonic-name-2] . . . TO { ON  } } . . .
                                                    { OFF }
```

SET Statement Format 5 5-32

```
SET condition-name TO TRUE
```

SIGN Clause 4-21

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

Sign Condition 1-15

```
                                     { POSITIVE }
arithmetic-expression IS [NOT]      { NEGATIVE }
                                     { ZERO     }
```

Page

SORT Statement 7-2

```
SORT file-name-1 ON {DESCENDING} KEY data-name-1 [, data-name-2] . . .
                    {ASCENDING }

   [ON {DESCENDING} KEY data-name-3 [, data-name-4] . . .] . . .
       {ASCENDING }

  [WITH DUPLICATES IN ORDER]

  [COLLATING SEQUENCE IS alphabet-name]

  {INPUT PROCEDURE IS section-name-1 [{THRU   } section-name-2]}
  {                                   {THROUGH}                }
  {USING file-name-2 [, file-name-3] . . .                      }

  {OUTPUT PROCEDURE IS section-name-3 [{THRU   } section-name-4]}
  {                                    {THROUGH}                }
  {GIVING file-name-4                                            }
```

SOURCE Clause 6-8

```
SOURCE IS identifier
```

SOURCE-COMPUTER Paragraph of Configuration Section 3-1

```
SOURCE-COMPUTER.  [computer-name

                    [, WITH DEBUGGING MODE]                 ]          10-5
```

SPECIAL-NAMES Paragraph of Configuration Section 3-2

```
SPECIAL-NAMES.

     [, implementor-name IS mnemonic-name] . . .                                    3-2

    ┌                            ┌ STANDARD-1                                        ┐ ┐
    │                            │ NATIVE                                            │ │
    │                            │ CDC-64                                            │ │
    │                            │ ASCII-64                                          │ │
    │                            │ EBCDIC                                            │ │
    │                            │ UNI                                               │ │
    │                            │                                                   │ │
    │                            │          ┌ { THRU    }  literal-2          ┐      │ │
    │ , ALPHABET alphabet-name IS┤ literal-1│ { THROUGH }                     │      ├ │ . . .   3-3
    │                            │          └ ALSO literal-3 [, ALSO literal-4] . . .┘ │ │
    │                            │                                                   │ │
    │                            │ ┌          ┌ { THRU    }  literal-6          ┐ ┐   │ │
    │                            │ │ , literal-5 { THROUGH }                    │ │ . . . │
    │                            │ └          └ ALSO literal-7 [, ALSO literal-8] . . .┘ ┘ │ │
    └                            └                                                   ┘ ┘

     [, CURRENCY SIGN IS literal]                                                    3-4

     [, DECIMAL-POINT IS COMMA]                                                      3-4

     [ , { QUOTE IS   }  APOSTROPHE ]                                                3-4
     [   { QUOTES ARE }             ]

     [ , SIGN CONTROL IS { LEADING  }  [SEPARATE CHARACTER] ]                        3-5
     [                   { TRAILING }                       ]

     [, SUB-SCHEMA IS sub-schema-name]                                               14-1

               ┌ IS mnemonic-name [ ON STATUS IS condition-name-1  ]               ┐
               │                  [ OFF STATUS IS condition-name-2 ]               │
     SWITCH-n  ┤                                                                   ├  3-5
               │ ON STATUS IS condition-name-1  [, OFF STATUS IS condition-name-2] │
               └ OFF STATUS IS condition-name-2  [, ON STATUS IS condition-name-1] ┘
```

START Statement Format 1 5-33

```
                         ┌     ┌ IS EQUAL TO         ┐           ┐
                         │     │ EQUALS              │           │
                         │     │ IS =                │           │
     START file-name     │ KEY ┤ EXCEEDS             ├ data-name │
                         │     │ IS GREATER THAN     │           │
                         │     │ IS >                │           │
                         │     │ IS NOT LESS THAN    │           │
                         └     └ IS NOT <            ┘           ┘

          [; INVALID KEY imperative-statement]
```

Page

START Statement Format 2 14-4

```
                       ┌      ⎧ IS EQUAL TO        ⎫           ┐
                       │      ⎪ EQUALS             ⎪           │
                       │      ⎪ IS =               ⎪           │
START relation-name    │ KEY ⎨ EXCEEDS            ⎬ data-name │
                       │      ⎪ IS GREATER THAN    ⎪           │
                       │      ⎪ IS >               ⎪           │
                       │      ⎪ IS NOT LESS THAN   ⎪           │
                       └      ⎩ IS NOT <           ⎭           ┘

     [; INVALID KEY imperative-statement]
```

STOP Statement 5-34

```
STOP ⎧ RUN     ⎫
     ⎩ literal ⎭
```

STRING Statement 5-34

```
STRING ⎧ identifier-1 ⎫ ⎡, identifier-2⎤ . . . DELIMITED BY ⎧ identifier-3 ⎫
       ⎩ literal-1    ⎭ ⎣, literal-2   ⎦                     ⎨ literal-3    ⎬
                                                             ⎩ SIZE         ⎭

   ┌  ⎧ identifier-4 ⎫ ⎡, identifier-5⎤ . . . DELIMITED BY ⎧ identifier-6 ⎫ ┐
   │ ,⎩ literal-4    ⎭ ⎣, literal-5   ⎦                     ⎨ literal-6    ⎬ │ . . .
   └                                                        ⎩ SIZE         ⎭ ┘

   INTO identifier-7  [WITH POINTER identifier-8]  [; ON OVERFLOW imperative-statement]
```

Subscripting 1-17

```
⎧ data-name      ⎫ (subscript-1 [, subscript-2  [, subscript-3] . . . [, subscript] ] )
⎩ condition-name ⎭
```

SUBTRACT Statement Format 1 5-36

```
SUBTRACT ⎧ literal-1    ⎫ ⎡, literal-2    ⎤ . . . FROM identifier-m  [ROUNDED]
         ⎩ identifier-1 ⎭ ⎣, identifier-2 ⎦

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

SUBTRACT Statement Format 2 5-36

```
SUBTRACT ⎧ literal-1    ⎫ ⎡, literal-2    ⎤ . . . FROM ⎧ literal-m    ⎫
         ⎩ identifier-1 ⎭ ⎣, identifier-2 ⎦           ⎩ identifier-m ⎭

   GIVING identifier-n  [ROUNDED] [, identifier-o  [ROUNDED] ] . . .

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

Page

SUBTRACT Statement Format 3 — 5-36

```
SUBTRACT { CORRESPONDING }  identifier-1 FROM identifier-2 [ROUNDED]
         { CORR          }

        [, identifier-3 [ROUNDED]] . . .  [; ON SIZE ERROR imperative-statement]
```

SUM Clause — 6-8

```
{ SUM identifier-1 [, identifier-2] . . . [UPON data-name-1 [, data-name-2] . . . ] } . . .

   [ RESET ON { data-name-3 } ]
   [          { FINAL       } ]
```

SUPPRESS Statement — 6-12

```
SUPPRESS PRINTING
```

SYNCHRONIZED Clause — 4-21

```
{ SYNCHRONIZED }  [ LEFT  ]
{ SYNC         }  [ RIGHT ]
```

TERMINATE Statement — 6-12

```
TERMINATE report-name-1 [, report-name-2] . . .
```

TYPE Clause — 6-9

```
              ( { REPORT HEADING }                                 )
              ( { RH             }                                 )
              ( { PAGE HEADING }                                   )
              ( { PH           }                                   )
              ( { CONTROL HEADING }    { data-name-1 }             )
              ( { CH              }    { FINAL       }             )
TYPE IS       < { DETAIL }                                         >
              ( { DE     }                                         )
              ( { CONTROL FOOTING }    { data-name-2 }             )
              ( { CF              }    { FINAL       }             )
              ( { PAGE FOOTING }                                   )
              ( { PF           }                                   )
              ( { REPORT FOOTING }                                 )
              ( { RF             }                                 )
```

UNSTRING Statement — 5-36

```
UNSTRING identifier-1

[ DELIMITED BY [ALL] { identifier-2 }  [, OR [ALL]  { identifier-3 } ] . . . ]
[                    { literal-1    }  [            { literal-2    } ]       ]

INTO identifier-4 [, DELIMITER IN identifier-5]  [, COUNT IN identifier-6]

    [, identifier-7 [, DELIMITER IN identifier-8] [, COUNT IN identifier-9] ] . . .

  [WITH POINTER identifier-10]  [TALLYING IN identifier-11]

[; ON OVERFLOW imperative-statement]
```

Page

USAGE Clause 4-22

```
              ( COMPUTATIONAL   )
              ( COMP            )
              ( COMPUTATIONAL-1 )
              ( COMP-1          )
[USAGE IS]   <  COMPUTATIONAL-2  >
              ( COMP-2          )
              ( COMPUTATIONAL-4 )
              ( COMP-4          )
              ( DISPLAY         )
              ( INDEX           )
```

USE Statement Format 1 5-38

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

USE Statement Format 2 6-12

```
USE BEFORE REPORTING identifier.
```

USE Statement Format 3 10-5

```
USE FOR DEBUGGING ON

{ [ALL REFERENCES OF] identifier-1 }  [ [ALL REFERENCES OF] identifier-2 ]
{ procedure-name-1                 }  [ procedure-name-2                 ]
{ file-name-1                      }  [ file-name-2                      ] . . . .
{ cd-name-1                        }  [ cd-name-2                        ]
{ ALL PROCEDURES                   }  [ ALL PROCEDURES                   ]
```

USE Statement Format 4 5-38

```
USE FOR HASHING ON file-name-1 [, file-name-2] . . . .
```

USE Statement Format 5 14-5

```
                             [     { INPUT     } ]
USE FOR ACCESS CONTROL       [ ON  { I-O       } ]
                             [     { INPUT I-O } ]
                             [     { I-O INPUT } ]

; KEY IS data-name [FOR { realm-name-1 [, realm-name-2] . . . } ] .
                        { REALMS                              }
```

USE Statement Format 6 14-5

```
USE FOR DEADLOCK ON { realm-name-1 [, realm-name-2] . . . } .
                    { REALMS                              }
```

Page

VALUE Clause Format 1 4-24

```
VALUE IS literal
```

VALUE Clause Format 2 4-24

```
{VALUE IS  }            [{THRU   }           ]   [             [{THRU   }           ]]
{VALUES ARE}  literal-1 [{THROUGH}  literal-2]   [ , literal-3 [{THROUGH}  literal-4]]  . . .
```

Working-Storage Section of Data Division 4-2

```
WORKING-STORAGE SECTION.

[data description entry] . . .
```

WRITE Statement Format 1 5-39

```
WRITE record-name [FROM identifier-1]

    [                            ( {identifier-2}  [LINE ] ) ]
    [ {BEFORE}                   ( {integer     }  [LINES] ) ]
    [ {AFTER }    ADVANCING      {                          } ]
    [                            ( {mnemonic-name}          ) ]
    [                            ( {PAGE         }          ) ]

    [ ; AT {END-OF-PAGE}   imperative-statement ]
    [      {EOP        }                        ]
```

WRITE Statement Format 2 5-40

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

FUTURE SYSTEM MIGRATION GUIDELINES F

This appendix contains programming practices recommended by CDC for users of the software described in this manual. When possible, application programs based on this software should be designed and coded in conformance with these recommendations.

Two forms of guidelines are given. The general guidelines minimize application program dependence on the specific characteristics of a hardware system. The feature use guidelines ensure the easiest migration of an application program to future hardware or software systems.

GENERAL GUIDELINES

Good programming techniques always include the following practices to avoid hardware dependency:

- Programs should avoid hardcoded constants. Manipulation of data should never depend on the occurrence of a type of data in a fixed multiple such as 6, 10, or 60.
- Programs should not manipulate data based on the binary representation of that data. Characters should be manipulated as characters, rather than as octal display-coded values or as 6-bit binary digits. Numbers should be manipulated as numeric data of a known type, rather than as binary patterns within a central memory word.
- Programs should not identify or classify information based on the location of a specific value within a specific set of central memory word bits.
- COMPASS should be avoided in application programs. COMPASS and other machine-dependent languages can complicate migration to future hardware or software systems. Migration is restricted by continued use of COMPASS for stand-alone programs, by COMPASS subroutines embedded in programs using higher-level languages, and by COMPASS owncode routines in CDC standard products. COMPASS should only be used to create part or all of an application program when the function cannot be performed in a higher-level language or when execution efficiency is more important than any other consideration.

FEATURE USE GUIDELINES

The recommendations in the remainder of this appendix ensure the easiest migration of an application program for use on future hardware or software systems. These recommendations are based on known or anticipated changes in the hardware or software system, or comply with proposed new industry standards or proposed changes to existing industry standards.

COBOL 5

COBOL 5 offers its users choices among features that perform the same function. The following paragraphs indicate preferred usage.

Comment Entries

Comment entries AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, or SECURITY should not be used. This information can be documented with a normal COBOL comment, using an * in column 7. This usage complies with a proposed revision to the ANSI standard.

SUPERVISOR, MEMORY, and ASSIGN OBJECT-PROJECT Clauses

The SUPERVISOR, MEMORY, and ASSIGN OBJECT-PROGRAM clauses should not be used. These clauses are currently used only for documentation and the information in them can be specified with a comment. This usage complies with a proposed revision to the ANSI standard.

Collating Sequence

Whenever possible, the ASCII collating sequence should be specified.

File Organizations

For advanced access methods, indexed or direct with or without alternate keys should be used; actual-key should not be used. For word-address files, a method that can be easily modified to byte addresses should be used; the REPLACE statement can be used to modify the code at a later date. No restrictions are imposed on sequential file usage.

Record Types

Specification of a particular record type with the USE clause should be avoided. This decision can be made by the COBOL compiler based on the characteristics of the records. If, however, the program would produce a default record type of D or T (for example, the record contains an OCCURS DEPENDING ON clause), the record type should be forced to W by specifying RT=W.

RECORD Clause

The VARYING phrase within the RECORD clause should be used if records are to be variable length. This usage complies with a proposed revision to the ANSI standard.

RECORDING MODE Clause

The RECORDING MODE clause must not be used; usage of this clause should not be necessary.

Level 77 Items

Level 77 items should not be used. Elementary 01 items can be used instead. This usage complies with a proposed revision to the ANSI standard.

REDEFINES and RENAMES Clauses

REDEFINES, RENAMES, or group operations involving synchronized or COMP-n items should be avoided, particularly where usage is based on a knowledge of the internal representation of data, such as floating-point layout or the number of characters per word.

Group Items

Avoid operations such as reference modification, string, and unstring on group items containing noncharacter data or synchronized items. Such usage is dependent on the characteristics of the hardware system.

ALTER Statement

The ALTER statement should not be used. In general, use of the ALTER statement does not conform to good coding practices. If its use is necesary, the GO TO DEPENDING ON statement can be used instead. This usage complies with a proposed revision to the ANSI standard.

Segment Numbers

Segment numbers 50 through 99 should not be used; in the future, segment numbers will be limited to 0 through 49. This should be adequate for virtually all applications. Programs should no longer need the initial-state capability of numbers 50 through 99 because this is only useful with the ALTER statement. This usage complies with a proposed revision to the ANSI standard.

OPEN REVERSED Statement

The REVERSED phrase of the OPEN statement should not be used. Usage has always led to inefficient processing, and the capability generally should not be required. This recommendation complies with a proposed revision to the ANSI standard.

SORT Statement

Specify the WITH DUPLICATES IN ORDER phrase for those sorts that expect it. Do not rely on this as the default. This usage complies with a proposed revision to the ANSI standard.

STRING and UNSTRING Statements

One of the operands in a STRING or UNSTRING statement must not be used as a subscript in the same statement. This is poor coding practice and generally can be avoided. This recommendation complies with a proposed revision to the ANSI standard.

INDEX

CONTROL DATA CORPORATION

COMMENT SHEET

MANUAL TITLE: COBOL Version 5 Reference Manual

PUBLICATION NO.: 60497100 **REVISION:** G

NAME: ______________________________

COMPANY: ______________________________

STREET ADDRESS: ______________________________

CITY: ____________ **STATE:** ____________ **ZIP CODE:** ____________

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

FOLD FOLD

TAPE TAPE