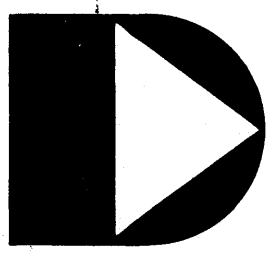
COBOL LANGUAGE SPECIFICATIONS COBOL User's Guide

Version 1

August, 1976

Model Code No. 50233

DATAPOINT CORPORATION



The leader in dispersed data processing ™

COBOL LANGUAGE SPECIFICATIONS
COBOL

User's Guide

Version 1

August, 1976

Model Code No. 50233

DATAPOINT CORPORATION

DATE: October 12, 1976

Errata to: COBOL MANUAL
Version 1
August, 1976
Model Code 50233

Reference: Page 1-1, Section 1.2

Change: CLUASES to CLAUSES

Reference: Page 6-13, Section 6.3

Add: (LIB version 2.1 or later) to end of third paragragh in section.

Reference: Appendix C, page C-1

Change: COBOL <sf>{,<cf>}{,clf}{,rlf}{;options} to

COBOL <sf>{,<cf>}{,rlf}{,clf}{;options}

Change: First sentence in first paragraph of Files section:

From: <sf> is the name of the source file; <cf>, the final command object file; <clf>, a user relocatable library file; <rlf>, the COPY library file.

To: <sf> is the name of source file; <cf>, the final command object file; <rlf>, a user relocatable library file; <clf>, the COPY library file.

9725 DATAPOINT DRIVE SAN ANTONIO, TEXAS 78284 (512) 690-7000

DATAPOINT CORPORATION

DATE: October 12, 1976

Errata to: COBOL MANUAL
Version 1
August, 1976
Model Code 50233

Reference: Page 6-16, Section 6.4.2.

Add: New paragraph following 3rd paragraph:

When the called subprogram is an assembler language program, the addresses of the identifiers specified in the USING list are put together into an argument list in the COBOL program and the subroutine is called with (HL) pointing to this argument list. It is up to the programmer to ensure that the number and type of arguments agree between the calling and called routines.

Reference: Page 5-7, Section 5.3

Add: under Formation and Evaluation Rules after item (5):

(6) If the exponent involved in the exponentiation operation contains no decimal point, all values for the exponent and the base are valid. If the exponent contains a decimal point, all values for the exponent are valid; however, only non-negative values for the base are valid in this context.

9725 DATAPOINT DRIVE SAN ANTONIO, TEXAS 78284 (512) 690-7000

DATAPOINT CORPORATION

DATE: October 12, 1976

Errata to: COBOL MANUAL
Version 1
August, 1976
Model Code 50233

Reference: New Appendix

Add: Appendix F. COBOL System Overlay Functions

Parser Overlays

BA - Parser initializing Overlay
BC - Identification Division Overlay
BE - Environment Division Overlay
BG - Data Division Parse Overlay
BI - Procedure Division Parser Pass 1
BK - Main Table re-organization
BM - Procedure Division Parser Pass 2
BQ - Parser Termination Overlay

Error message generation

BZ/OA/ZA - Error Message Generation

Symbol Table Overlays

DA - Node Linking Phase
DC - Picture Analysis Phase
DE - Size Computation Phase
DG - Procedure Division Name Generation Phase

Merge Overlays

FA - Build Symbol Table Chains
FC - File Attribute Merge
FK - Data Division Attribute Merge
FM - Build I/O table

9725 DATAPOINT DRIVE SAN ANTONIO, TEXAS 78284 (512) 690-7000

Translation Ovelays

KA - Control Translation
KD - Arithmetic Translation
KG - I/O Translation

Index Generation/Optimization Overlay

OE - Subscript/Index Generation and Optimization

PIT Generation Overlays

PA - Parameter PIT Generation
PC - Data Division PIT Generator
PE - Literal Allocation
PG - FDB PIT Generation
PK - Procedure Division PIT Generator
PM - GET PIT Generation
PO - PUT PIT Generation
PP - UPDATE PIT Generation
PW - Generate Prologues & Sort Segments
PZ - Subscript Finalization

Object Program Generation Overlays

WA - OPG initialization
WC - OPG Pass 1
WE - OPG Pass 2
WG - OPG Listing Pass
WI/WJ - Recycle OPG
WM - Generate /SYM file
WO - Generate XREF sort file
WQ - Sort and print XREF
WX - Generate TXC file

Termination Overlays

ZW - Error Messages Lister
ZZ - Termination Phase

interoffice memo
DATAPOINT CORPORATION

DATE: January 24, 1977

Addendum to: Cobol User's Guide
Version 1
Model #50233

Add: Sections 6.6, 6.6.1, 6.7, 6.7.1, 6.7.2, 6.7.3, 6.7.4, and Appendix G.

Section 6.6 DOS Command-line Interface Feature

The DOS Command-line interface feature gives the COBOL programmer access to the DOS command line (MCR$).

Section 6.6.1 The COMMAND-LINE Special Variable

Pre-defined in every COBOL program is a variable named COBOL-LINE which has the characteristics of a 79 character alphanumeric string i.e., PICTURE X(79). This variable does not occupy storage but simply defines the interface to the programmer.

COMMAND-LINE can be used in any place that a regular alphanumeric variable can be used e.g., MOVE, DISPLAY, or ACCEPT, statements. When the COBOL program starts, COMMAND-LINE contains the information entered by the user to the DOS command interpretor. Information moved into COMMAND-LINE will be there after the program completes, unless an abort occurs during execution.

Section 6.7 SPECIAL I/O Feature

The SPECIAL I/O feature allows the COBOL programmer to interface to unsupported I/O devices using the standard I/O statements.

Section 6.7.1 Environment Division Considerations

For files that are maintained on unsupported I/O devices, the device specification in the ENVIRONMENT DIVISION has the form

SELECT <filename> ASSIGN TO
SPECIAL -<rtnname>[-<integer>]

<rtnname> is the name of a SNAP2 program that will perform the operations; <integer> is the number of bytes to be reserved in the File Descriptor Block (FDB) for the file. If <integer> is not specified, no space will be reserved.

Section 6.7.2 Procedure Division Considerations

OPEN, CLOSE, READ, and WRITE statements may be used with SPECIAL files.

Section 6.7.3 I/O Subroutine

Associated with each SPECIAL file is a SNAP2 program to perform the actual operations; more than one file can be associated with a single program. This program is called once for every OPEN, CLOSE, READ, or WRITE operation. The address of the File Descriptor Block is passed in (HL) and a code specifying what operation is to be performed is passed in (A). The file IFDBDEF/TXT, which defines the format of a File Descriptor Block, contains the definitions or these codes.

Section 6.7.4 Reserve Area

In each File Descriptor Block associated with a SPECIAL file is a block of memory called the Reserve Area. The size of this area is specified in the SELECT clause for the file. Initially, the area contains binary zeros; the area is supplied solely for the use of the I/O sub-routine and its contents are not interogated or modified by the COBOL runtime system.

For Appendix G. see printer listing.

PREFACE

Datapoint COBOL adheres to the American National Standard and contains the following modules (See American National Standard X3.23 1968):

NUCLEUS, LEVEL 1
TABLE FACILITIES, LEVEL 2
SEQUENTIAL ACCESS, LEVEL 1
RANDOM ACCESS, LEVEL 1
SORTING FACILITIES, LEVEL 2
SEGMENTATION, LEVEL 1
LIBRARY FACILITIES, LEVEL 2

AND: Selected Features from Level 2 of Nucleus, Sequential Access and Random Access:

NAME QUALIFICATION
FULL CONTINUATION FOR WORDS AND LITERALS
COMPLETE FIGURATIVE CONSTANTS
ARITHMETIC EXPRESSIONS
EXTENDED FILE OPTIONS

AND: Non-ANSI Extensions:

GEDIT FORMAT FILES

CASSETTES

DATABUS AND EBCDIC NUMERIC FORMATS

INDEXED SEQUENTIAL FILE ACCESS

CALLING PRE-COMPILED SUB-PROGRAMS

DEBUGGING FACILITIES

Acknowledgment

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

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

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

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

"The authors and copyright holders of the copyrighted material used herein:

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

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

TABLE OF CONTENTS

Appendix E. RESERVED WORDS

CHAPTER 1. OVERALL LANGUAGE CONSIDERATIONS

1.1 Introduction

This manual contains specifications for the Datapoint subset of the American National Standard COBOL. COBOL, the common business-oriented language, is designed for solving business problems using an English-like programming language. The elementary components of a COBOL program are words and other character strings. These components are combined into entities called entries and sentences. These entries and sentences are then combined into the paragraphs, sections, and divisions of a COBOL program. In this manual we define the format and meaning of each of these entities.

This chapter specifies the general features of the COBOL language, and defines the general concepts used in COBOL.

1.2 Notation Used in Formats and Rules

A General Format is the specific arrangement of the elements of a clause or a statement. A clause or a statement consists of elements. Elements consist of upper case words, lower case words, level numbers, brackets, braces, the alternator, connectives and special characters. Throughout this manual a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. CLAUSES MUST BE WRITTEN IN THE SEQUENCE GIVEN IN THE GENERAL FORMATS. (Clauses that are optional must appear in the sequence shown if they are used.) In certain cases, stated explicitly in the rules associated with a given format, the clauses may appear in sequences other than that shown. Applications, requirements or restrictions are shown as rules.

1.2.1 Words

All underlined upper-case words are called key words and are required when the functions of which they are a part are used. Upper-case words which are not underlined are optional to the user and may or may not be present in the source program. Upper-case words, whether underlined or not, must be spelled correctly.

Lower-case words, in a general format, are generic terms used to represent COBOL words that must be supplied by the user. Except for the list of words following, such lower-case words occuring in a general format are replaced, in an actual program, by a single COBOL word:

a. statement
b. imperative-statement
c. arithmetic-expression
d. character-string
e. comment-entry
f. condition
g. literal

These exceptions represent combinations of COBOL words constructed in accordance with the definitions specified in Section 5.2 of the Procedure Division for statement, and imperative-statement. Condition is defined in Section 5.4 of the Procedure Division. Arithmetic expression is defined in Section 5.3 of the Procedure Division. Definition for the term character-string is given in Section 4.4.8 of the Data Division. Comment-entry is defined in Section 2.1 of the Identification Division. Literal is defined in Section 1.4.9 of this chapter.

Where generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion.

1.2.2 Level Numbers

When specific level-numbers appear in data descriptions entry formats, those specific level-numbers are required when such entries are used in a COBOL program.

1.2.3 Brackets, Braces, and the Alternator

When a portion of a general format is enclosed in brackets, [], that portion may be included or omitted at the user's choice. Braces, {}, enclosing a portion of a general format means a selection of one of the options contained within the braces must be made. In both cases, the alternatives to be selected are separated from each other by the alternator, |. When brackets or braces enclose a portion of a format but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies (see The Ellipsis, below).

1.2.4 The Ellipsis

In text, the ellipsis (...) may show the omission of a portion of a source program. This meaning becomes apparent in context.

In the general format, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

Given ... in a clause or statement format, scanning right to left, determine the] or } immediately to the left of the ...; continue scanning right to left and determine the logically matching [or {; the ... applies to the words between the determined pair of delimiters.

1.2.5 Format Punctuation

The punctuation characters, comma and semicolon, are shown in some formats. However, a semicolon must not appear immediately preceding the first clause of an entry or paragraph. The use of these punctuation characters for each division is as follows:

Identification Division.

Although not expressly shown in the formats within this division, the comma and semicolon may be used within the comment-entry. The paragraph itself must terminate with a

period followed by a space.

Environment Division. Where a comma is shown in the formats, it is optional and may be included or omitted by the user. Where a semicolon is shown in the formats, it is optional and may be included or omitted by the user. The paragraph itself must terminate with a period followed by a space.

Data Division. Where a comma is shown in the formats, it is optional and may be included or omitted by the user. Where a semicolon is shown in the formats, it is optional and may be included or omitted by the user. The paragraph itself must terminate with a period followed by a space.

Procedure Division. A comma used between parentheses for subscripting or indexing is required. Elsewhere in the formats, it may be omitted. A comma must not appear immediately following the first word of a statement. If desired, a semicolon can be used between statements.

Use of Certain Special Characters in Formats:

The characters '+', '-', '>', '<', '=', when appearing in formats, although not underlined, are required when such formats are used.

1.3 Language Concepts

1.3.1 Cobol Character Set

The complete Datapoint COBOL character set consists of the 51 characters listed below:

Character	Meaning
0,1,...,9	digit
A,B,...,Z	letter
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign (equals)
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

1.3.2 Characters Used in Words

A word is a sequence of not more than 30 characters. Each character is selected from the set 'A', 'B', 'C', ..., 'Z', '0', '1', ..., '9', '-', except that the '-' may not appear as the first or last character in a word. A word is delimited by separators.

1.3.3 Punctuation Character

The following characters are used for punctuation:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis

1.3.4 Editing Characters

Editing characters are single characters or fixed two-character combinations belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)

1.3.5 Characters Used in Arithmetic Expressions

The following character combinations are used in arithmetic expressions:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

1.3.6 Characters Used in Relations

The following characters are used in relations:

Character	Meaning
<	less than
=	equal to
>	greater than

1.3.7 Separators

The space and the punctuation characters, when not contained within quotation marks, are separators. Where a space is used, more than one may be used, except for the restrictions set forth in this chapter.

A character-string is delimited on the right by a space, period, right parenthesis, comma, or semicolon.

The use of punctuation characters in connection with character-strings is defined as follows:

(1) A space must follow a period, comma and semicolon when any of these punctuation characters are used to delimit a character-string.
(2) A space must neither immediately follow a left parenthesis nor immediately precede a right parenthesis.

1.4 Character String

A character string is a sequence of contiguous characters which form a literal, a word, a PICTURE character-string, or a NOTE character-string.

1.4.1 Words

A word is a sequence of not more than 30 characters. Each character is selected from the set 'A', 'B', ..., 'Z', '0', '1', ..., '9', '-', except that the '-' may not appear as the first or last character in a word.

1.4.2 Data Name

A data-name is a word that contains at least one alphabetic character and that names an entry in the Data Division, including file description entries.

1.4.3 Condition-Name

A condition-name is a word with at least one alphabetic character, which is assigned to a specific value within the complete set of values that a data item may assume. The data item itself is called a conditional variable. Each condition-name must be unique, or be made unique through qualification. A conditional variable may be used as a qualifier for any of its condition-names. If references to a conditional variable require indexing, subscripting or qualification, then its condition-names also require the same combination of indexing, subscripting or qualification (see Uniqueness of Data Reference, Section 1.5.7).

In addition to being described in the Data Division, condition-names may also be defined in the SPECIAL-NAMES paragraph within the Environment Division, where a condition-name may be given to the ON status or OFF status, or both, of hardware or operating system conditions.

A condition-name is used in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to the value to which that condition-name is assigned.

1.4.4 Procedure-Name

A procedure-name is a word which is used to name a paragraph or section in the Procedure Division. Procedure-names composed only of the digits '0' through '9' are equivalent if, and only if, they are composed of the same number of digits and have the same value.

1.4.5 Figurative Constants

Certain constants, called figurative constants, have been assigned fixed data-names. These data-names must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The fixed data-names and their meanings are as follows:

ZERO ZEROS ZEROES	Represents the value 0, or one or more of the character 0, depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest value in the collating sequence, namely octal 0377.
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest value in the collating sequence, namely octal 0.
QUOTE QUOTES	Represents one or more occurrences of the quotation mark character. The word QUOTE cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal.
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readibility only.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

(1) When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is equal to the size in characters of the associated data item.
(2) When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, EXAMINE, or STOP statement, the length of the string is one character. The figurative constant ALL literal may not be used with DISPLAY, EXAMINE, or STOP.

A figurative constant can be used any place where a literal

appears in the format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

1.4.6 Special Registers

Special registers are data items generated by the compiler to support the use of certain COBOL features. Seven such special registers may be generated by the Datapoint COBOL compiler.

(1) TALLY. The word TALLY is the name of a special register whose implicit description is that of an integer of five digits, without an operational sign, and whose implicit USAGE is computational. The primary use of the TALLY register is to hold information produced by the EXAMINE statement. The word TALLY may also be used as a data-name wherever an elementary data item of integral value may appear.

(2) LINAGE-COUNTER. The word LINAGE-COUNTER is the name of a special register generated for each FD with a LINAGE clause. The implicit description of a LINAGE-COUNTER is that of an integer without an operational sign whose implicit USAGE is computational and whose size is sufficient to hold the maximum number of lines specified in the LINAGE clause. A LINAGE-COUNTER may not be explicitly modified by the COBOL program.

The value of a LINAGE-COUNTER represents the current line number on a logical page. This value is set to one when the file is opened and reset to one after advancing (to the top of the) page. When a WRITE statement to a file with a LINAGE clause is executed, the value of the LINAGE-COUNTER is incremented either by the value specified in the ADVANCING option or by one if no ADVANCING option is specified.

See The LINAGE Clause, Section 4.1.5, and The WRITE Statement, Section 5.9.5.

(3) DATE. The word DATE is the name of a special register which holds the date. Its implicit description is that of an integer of six digits without an operational sign, and whose implicit USAGE is computational. The value of a date is represented by two digits for the year in the century, two for the month in the year, and two for the day of the month. The value of DATE may be made accessible to a COBOL program by executing the ACCEPT DATE statement. (See The ACCEPT

Statement, Section 5.9.7.)

(4) FILE-NAME. The word FILE-NAME is the name of a special register generated for each FD assigned to a disk file. Its implicit description is that of a group item containing the three elementary items - MAIN-NAME, EXTENSION, and DRIVE-NUMBER. The FILE-NAME of a disk file can be set by a program to specify the exact file to open before it is opened, and can be read after a file is opened for use in headings, name verification, etc. The three components of a FILE-NAME correspond to the DOS components of a name.

(5) MAIN-NAME. The word MAIN-NAME is the name of a special register generated as the first component of a FILE-NAME. Its implicit description is that of an alphanumeric item of eight characters whose implicit USAGE is display. This register holds the main name portion of a DOS disk file name.

(6) EXTENSION. The word EXTENSION is the name of a special register generated as the second component of a FILE-NAME. Its implicit description is that of an alphanumeric item of three characters whose implicit USAGE is display. This register holds the extension portion of a DOS disk file name.

(7) DRIVE-NUMBER. The word DRIVE-NUMBER is the name of a special register generated as the third component of a FILE-NAME. Its implicit description is that of an alphanumeric item of two characters whose implicit USAGE is display. This register holds either two digits for an explicit DOS drive number or two spaces for an implicit DOS drive specification (i.e., all drives).

1.4.7 Mnemonic Names

Mnemonic names are the means of relating system-names with problem-oriented names, and, also, the status of software switches with condition-names. (See Section 3.2.3, The SPECIAL-NAMES Paragraph.)

1.4.8 Reserved Words

A specified list of words which may be used in COBOL Source Programs, but which may not appear in the programs as user-defined words. See Appendix E for a complete list of reserved words.

There are three types of reserved words, as shown below:

(1) Key Words. A key word 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 uppercase and underlined.

    Key Words are of three types:
    a. Verbs such as ADD, READ, ENTER.
    b. Required words, which appear in statement and entry formats.
    c. Words which have a specific functional meaning such as NEGATIVE, SECTION, TALLY, etc.

(2) Optional Words. Within each format, uppercase words that are not underlined are called optional words and may appear at the user's option. The presence or absence of each optional word within a format does not alter the compiler's translation. Misspelling of an optional word, or its replacement by another word of any kind is not allowed.

(3) Connectives. There are three types of connectives:
    a. Qualifier connectives that are used to associate a data-name or a paragraph-name with its qualifier: OF, IN.
    b. Series connectives that link two or more consecutive operands: , (comma).
    c. Logical connectives that are used in the formation of conditions: AND, OR, NOT, AND NOT, OR NOT.

1.4.9 Literals

A literal is a string of characters whose value is implied by an ordered set of characters of which the literal is composed. Every literal belongs to one of two types, numeric or nonnumeric.

Non-numeric Literals

A non-numeric literal is defined as a string of any allowable characters in the ASCII character set, excluding the character quotation mark, bounded by quotation marks. Non-numeric literals

can contain 1 through 120 characters. The value of a non-numeric literal is the string of characters itself, excluding the quotation marks. Any spaces enclosed in the quotation marks are part of the non-numeric literal and, therefore, are part of the value. All non-numeric literals are category alphanumeric (See the PICTURE Clause, Section 4.4.8).

Numeric Literals

A numeric literal is defined as a string of characters chosen from the digits 0 through 9, the plus sign, the minus sign, and the decimal point. Numeric literals of 1 through 18 digits in length are allowed. The rules for formation of numeric literals are as follows:

(1) A literal must contain at least one digit.
(2) A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.
(3) A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric (See the PICTURE Clause, Section 4.4.8).

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a non-numeric literal and it is treated as such by the compiler.

1.4.10 PICTURE Character String

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. The allowable combinations are explained under the PICTURE clause (See Section 4.4.8).

1.4.11 NOTE Character String

A NOTE character-string may consist of any characters from the ASCII character set. This character string is normally terminated with a period followed by a space.

1.5 Concept of Computer Independent Data Description

To make data as computer-independent as possible, the characteristics or properties of the data are described in relation to a Standard Data Format rather than an equipment-oriented format. This Standard Data Format is oriented to general data processing applications and uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the COBOL character set to describe non-numeric data items.

1.5.1 Logical Record and File Concept

The approach taken in defining file information is to distinguish between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

Physical Aspects of a File:

The physical aspects of a file describe the data as it appears on the input or output media and include such features as:

(1) The grouping of logical records within the physical limitations of the file medium.
(2) The means by which the file can be identified.

Conceptual Characteristics of a File:

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a

logical record. A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or a logical record may require more than one physical unit to contain it. There are several source language methods available for describing the relationship of logical records and physical units. Once the relationship has been established, the control of the accessibility of logical records as related to the physical unit is carried out implicitly by the object program. In this manual references to records mean to logical records, unless the term 'physical record' is specifically used.

The concept of logical record is not restricted to file data but is carried over into the definition of working storage and constants. Thus, working storage and constants may be grouped into logical records and defined by a series of data description entries.

Record Concepts

The Record Description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level number followed by a data-name, if required, followed by a series of independent clauses, as required.

1.5.2 Concept of Levels

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a logical record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not

further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. Thus an elementary item may belong to more than one group.

Level Numbers

A system of level numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level numbers for records start at 1 or 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater in value than 49. There are special level numbers 77 and 88 which are exceptions to this rule (see below). Separate entries are written in the source program for each level number used.

A group 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. The level number of an item, either elementary or a group item, immediately following the last elementary item of the previous group, must be that of one of the groups to which the prior elementary item belongs.

Two types of entries exist for which there is no true concept of levels. These are:

(1) Entries that specify noncontiguous Working-Storage items.
(2) Entries that specify condition-names.

Entries that specify noncontiguous Working-Storage items, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level number 77.

Entries that specify condition-names, to be associated with a particular value of a conditional-variable, have been assigned the special level number 88.

1.5.3 Concept of Data Classes

The five categories of data items (see Section 4.4.8, The PICTURE Clause) are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing). Every elementary item belongs to one of the classes and further to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to that group item.

1.5.4 Character Representation and Radix

The value of a numeric item is represented in zoned decimal format. Since this representation is actually combinations of bits, it is commonly called binary-coded decimal. The ASCII binary code (a binary-coded decimal form) is also used to represent characters and symbols that are alphanumeric items. The size of an elementary data item or group item is the number of characters in the Standard Data Format representation of the item.

1.5.5 Algebraic Signs

Algebraic signs are used for two purposes:

(1) To show whether the value of an item involved in an operation is positive or negative; and
(2) To identify the value of an item as positive or negative on an edited report for external use.

Since the method of representing an operational sign is standard, an indication that an operational sign is associated with an item is sufficient.

Editing sign control characters are used to display the sign of an item and are not operational signs. These editing characters are available through the use of the PICTURE clause.

1.5.6 Overall Structure of a COBOL Source Program

A COBOL source program is composed of four divisions: the Identification Division, the Environment Division, the Data Division, and the Procedure Division. Each division is mandatory and must appear in order. Each division has additional structure as shown in the following format:

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

    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER. entry
    OBJECT-COMPUTER. entry
    [SPECIAL-NAMES. entry]
    [INPUT-OUTPUT SECTION.
    FILE-CONTROL. {entry}...
    [I-O-CONTROL. entry]]

    DATA DIVISION.
    [FILE SECTION.
    {file description entry
    {record description entry}...}...]
    [WORKING-STORAGE SECTION.
    [data description entry]...
    [record description entry]...]
    [LINKAGE SECTION.
    [data description entry]...
    [record description entry]...]

    PROCEDURE DIVISION [USING ...].
    {[section-name SECTION [priority].]
    {paragraph-name. {sentence}...}...}...
```

Each division is completely specified in subsequent chapters. In brief, the purpose of each division is as follows:

The Identification Division serves to identify the program, and may include other useful general comments about the program.

The Environment Division is divided into a required Configuration Section and an optional Input-Output Section.

The Configuration Section defines the computer configuration used to compile and to execute the program.

The Input-Output Section defines the assignment of files to devices.

The Data Division is used to define the storage used by the program. It is divided into a File Section where storage associated with files is described, a Working-Storage Section where non-file storage is described, and a Linkage Section where storage defined in other COBOL programs is described (see Section 6.4, The Sub-Program Feature). All sections are optional.

The Procedure Division contains sentences which define the processing performed by the program. Sentences are collected to form paragraphs. Paragraphs may be collected to form sections.

1.5.7 Uniqueness of Data Reference

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because the name can be made unique through a syntactically correct combination of qualifiers, subscripts or indices.

Identifier is the term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts or indices necessary to ensure uniqueness. The general formats for identifiers are:

Format 1

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

Format 2

```
data-name-3 [{OF | IN} data-name-4]...
  [(index-name-1 [{+ | -} integer]
    [, index-name-2 [{+ | -} integer]
      [, index-name-3 [{+ | -} integer]]])]
```

Restrictions on qualification, subscripting and indexing are:

(1) The commas shown in both formats are required.
(2) A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, subscript, or qualifier.
(3) Indexing is not permitted where subscripting is not permitted.
(4) An index may be modified only by the SET and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data without conversion. Such data items are called index data items. (See The USAGE Clause, Section 6.1.1.2.)

The details of the process by which a data-name is made unique through qualification, subscripting, and indexing now follow.

1.5.7.1 Qualification

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because the name exists within a hierarchy of names such that the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and this process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. The name associated with a level indicator is the highest level qualifier available for a data-name. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus level indicator names and section-names must be unique in themselves as they cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and a procedure-name.

Qualification is performed by following a data-name or a paragraph-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent. The general formats for qualification are:

Format 1

```
{data-name-1 | condition-name}
  [{OF | IN} data-name-2]...
```

Format 2

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

The rules for qualification are as follows:

(1) Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.
(2) The same name must not appear at two levels in a hierarchy.
(3) If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except REDEFINES where, by definition, qualification is unnecessary).
(4) A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section.
(5) A data-name cannot be subscripted when it is being used as a qualifier.
(6) A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.

1.5.7.2 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 (see Section 6.1.1.1, The OCCURS Clause).

The subscript can be represented either by a numeric literal that is an integer, or by the special register TALLY, or by a

data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may be qualified but not subscripted.

The subscript may contain a plus sign. The lowest possible subscript value is 1. This value 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, ... The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the terminal space of the table element data-name. The table element data-name appended with a subscript is called a subscripted data name or an identifier. When more than one subscript appears within a pair of parentheses, the subscripts must be separated by commas. A space must follow each comma, but no space may appear between the left parenthesis and leftmost subscript or between the right parenthesis and rightmost subscript. The format is:

```
data-name (subscript [, subscript]...)
```

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE L	C	A	B	COBOL STATEMENT	R	IDENTIFICATION
				DATA DIVISION.		
				WORKING-STORAGE SECTION		
				01 TABLE-A		
				02 ENTRY-A-1 PICTURE XXX OCCURS 4 TIMES.		
				02 ENTRY-A-2 OCCURS 4 TIMES.		
				03 ENTRY-A-3 PICTURE X.		
				03 ENTRY-A-4 PICTURE X.		
				01 TABLE-10.		
				02 STATE-1 OCCURS 10 TIMES.		
				03 YEAR-1 OCCURS 10 TIMES.		
				04 ANIMAL PICTURE 999 OCCURS 3 TIMES.		
				PROCEDURE DIVISION.		
				SINGLE SUBSCRIPTS.		
				MOVE ENTRY-A-1 (1) TO FLD-X.		
				IF ENTRY-A-1 (SUB1) NOT EQUAL '111' GO TO PARA-A.		
				MULTIPLE-SUBSCRIPTS.		
				MOVE 3 TO W-1.		
				MOVE 10 TO W-2.		
				MOVE 10 TO W-3.		
				MOVE ANIMAL (W-3, W-2, W-1) TO FLD-A.		
				IF YEAR-1 (1, 1) EQUAL TO '1234567890' GO TO PARA-C.		
				IF YEAR-1 (W-1, 10) EQUAL TO '9999999999' GO TO PARA-D.		

Figure 1-1. Example of subscript usage.

1.5.7.3 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. An index-name must be initialized by a SET statement before it is used as a table reference (see Section 6.1.2.3, The SET Statement).

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integral numeric literal all enclosed in the parentheses immediately after the terminal space of the data name. The general format for indexing is:

```
data-name (index-name [{+ | -} integer]
  [, index-name [{+ | -} integer]]...)
```

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE	C	A	B	COBOL STATEMENT	IDENTIFICATION
				DATA DIVISION.	
				WORKING-STORAGE SECTION.	
				01 TABLE-1.	
				02 TAB1-REC PICTURE 99 OCCURS 100 TIMES	
				INDEXED BY INDEX-1.	
				01 TABLE-9.	
				02 TABLE-8 OCCURS 10 TIMES INDEXED BY INDEX-A.	
				03 TABLE-7 OCCURS 10 TIMES INDEXED BY INDEX-B.	
				04 TABLE-6 PICTURE 999 OCCURS 3 TIMES INDEXED BY INDEX-C.	
				PROCEDURE DIVISION.	
				SINGLE-INDEX.	
				SET INDEX-1 TO 1.	
				MOVE TAB1-REC (INDEX-1) TO FLD-A.	
				MULTIPLE-INDEX.	
				SET INDEX-A INDEX-B INDEX-C TO 1.	
				IF TABLE-6 (INDEX-A, INDEX-B, INDEX-C) EQUAL TO 299 GO TO XX.	
				IF TABLE-7 (INDEX-A, INDEX-B) EQUAL ZERO GO TO YY.	

Figure 1-2. Example of indexing.

1.6 Reference Format

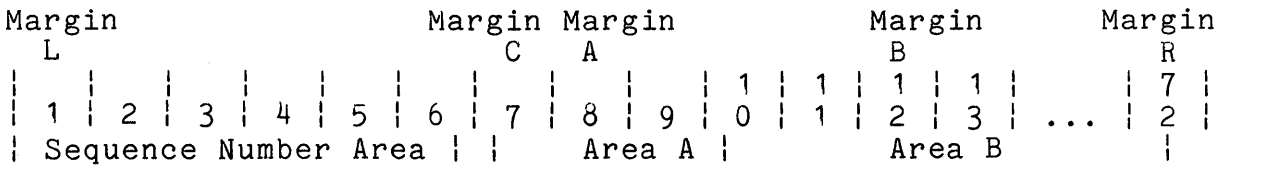
The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input-output medium. The input to the Datapoint COBOL compiler is from a GEDIT-compatible disk file. Each record from the file is treated as if it were 72 characters long. Shorter records will have blanks appended; longer records will be truncated after the 72nd character has been read. In the latter case the additional characters in the record will be treated as commentary and ignored in the processing of the program. Within these definitions, the Datapoint COBOL compiler accepts source programs written in reference format and produces an output listing in reference format.

The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The division of a source program must be ordered as follows: the Identification Division, then the Environment Division, then the Data Division, then the Procedure Division. Each division must be written according to the rules for the reference format.

1.6.1 Reference Format Representation

The reference format for a line is represented as follows:



Margin L designates the first, or left-most, character position of a line.
Margin C designates the seventh character position relative to Margin L.
Margin A designates the eighth character position relative to L.
Margin B designates the twelfth character position relative to L.
Margin R designates the seventy-second, or right-most, character position of a line.

The sequence number area occupies the six character positions beginning at Margin L.

The continuation area occupies one character position beginning at Margin C.

Area A occupies four character positions beginning at Margin A.

Area B occupies sixty-one character positions beginning at Margin B.

Sequence Numbers

A sequence number, consisting of six digits in the sequence number area, may be used to label a source program line.

Continuation of Lines

Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the continuation area of a line indicates that the first nonblank character in Area B of the current line is the successor of the last nonblank character of the preceding line

successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continued line must be blank.

If there is no hyphen in the continuation area of a line, it is assumed that the last character in the preceding line is followed by a space.

Blank Lines

A blank line is one that is blank from Margin C to Margin R, inclusive. A blank line can appear anywhere in the source program, except immediately preceding a continuation line. (See Continuation of Lines, above.)

Comment Lines

Commentary on lines containing an asterisk in the Continuation Area may be inserted anywhere in a COBOL Source Program except immediately preceeding a continuation line. Any combination of characters may be placed in Areas A and B. The line will be listed but will have no further effect on compilation.

Lines containing a slash in the Continuation Area cause a new page to be started in the source listing and then are treated as if the Continuation Area contained an asterisk.

See also Section 5.12.2, The NOTE Statement.

1.6.2 Division, Section, and Paragraph Formats

Division Header

The division header starts in Area A with the division-name, is followed by a space, then the word DIVISION, then a period. After the division header, no text may appear before the following

section or paragraph-header or paragraph-name (that is before the next line without a hyphen in the continuation area and with a nonblank character in Area A), except:

PROCEDURE DIVISION USING ....

Section Header

The name of a section starts in Area A of any line except the first line of a division reference format, is followed by a space, then the word SECTION, then optionally a space followed by a priority number, then a period followed by a space. After the section-header, no text may appear before the following paragraph-header or paragraph-name (that is before the next line without a hyphen in the continuation area and with a nonblank character in Area A), except:

The COPY statement may be present.

A section consists of paragraphs in the Environment and Procedure Divisions, and Data Division entries in the Data Division. Paragraph-names, but not section-names, are permitted in the Identification Division.

Paragraph-Header, Paragraph-Name and Paragraph

A paragraph consists of a paragraph-name followed by one or more sentences, or a paragraph-header followed by one or more entries.

A paragraph-header starts in Area A of any line following the first line of a division or a section. The name of a paragraph starts in Area A of any line following the first line of a division or a section and ends with a period followed by a space.

The first sentence or entry in a paragraph begins in Area B of either the same line as the paragraph-name or paragraph-header or the next nonblank line. Successive sentences or entries either begin in Area B of the same line as the preceding sentence or entry or in Area B of the next nonblank line.

A sentence consists of one or more statements, an entry consists of one or more clauses; all sentences and entries must be followed by a period followed by a space.

When the sentences or entries of a paragraph require more than one line they may be continued as described in Continuation of Lines,

above.

1.6.3 Data Division Entries

Each Data Division entry begins with a level indicator or a level number, followed by a space, followed by the name of a data item, followed by a sequence of independent clauses describing the data item. Each clause, except the last clause of an entry, may be terminated by a semi-colon followed by a space. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level number.

The only level indicators in Datapoint COBOL are FD and SD.

In those Data Division entries that begin with a level indicator, the level indicator begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

Those Data Division entries that begin with level numbers are called data description entries.

In those data description entries that begin with a level number 1 or 77, the level number begins in Area A followed in Area B by its associated record-name or item-name and appropriate descriptive information.

A level number may be one of the following set: 1 through 49, 77, 88. Single digit level-numbers are written either as a single space followed by a digit or as a zero followed by a digit. At least one space must separate a level number from the word following the level number.

Successive data description entries may have the same format as the first or may be indented according to level number. The entries in the output listing are indented only if the input is indented. Indentation does not affect the magnitude of a level number.

When level numbers are to be indented, each new level-number may begin any number of spaces to the right of Margin A. The extent of indentation to the right is determined only by the size of Margin R.

All successive (numerically higher) level numbers need not be indented, rather they may begin in the same position as any preceeding level number.

CHAPTER 2. IDENTIFICATION DIVISION

The Identification Division must be included in every COBOL source program. The Identification Division identifies both the source program and the resultant output listing. In addition, the user may include the date the program is written and such other information as desired under the paragraphs in the General Format shown below.

Fixed paragraph names identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in the order of presentation shown by the General Format below.

2.1 Structure

The following format shows the structure of the Identification Division.

General Format

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

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space. The comment-entry may be any combination of the characters from the ASCII character set organized to conform to sentence and paragraph format. The following text defines the PROGRAM-ID paragraph. While the other paragraphs are not defined by a General Format, each is formed of comment-entries in the same manner.

2.2 The PROGRAM-ID Paragraph

The PROGRAM-ID paragraph gives the name by which a program is identified.

General Format

PROGRAM-ID. program-name.

The program-name must conform to the rules for formation of a procedure-name. The PROGRAM-ID paragraph must contain the name of the program and must be present in every program.

The program-name is used to specify the name of the main object program module, subject to the following conversion. Up to the first seven characters of the program-name are used to name the object module. If any of the characters are '-' (hyphen), they are replaced with '$' for compatibility with the DOS link editor. If the first character is numeric, it is converted as follows:

numeric	conversion
0	J
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE		COBOL STATEMENT	IDENTIFICATION
L	C A	B R	

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE-PROG-NAME.
AUTHOR. NOBODY.
INSTALLATION. DATAPOINT.
DATE-WRITTEN. JULY, 1976.
SECURITY. NONE.
REMARKS. THIS IS AN EXAMPLE OF AN IDENTIFICATION DIVISION FOR
         A COBOL PROGRAM.
```

Figure 2-1. Example of IDENTIFICATION DIVISION.

CHAPTER 3. ENVIRONMENT DIVISION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specifications of the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics and control techniques can be given. The Environment Division must be included in every COBOL source program. The Environment Division is made up of two sections: the Configuration Section and the Input-Output Section.

3.1 Structure

The following is the general outline of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program. The general format is as follows:

General Format

```
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]]
```

3.2 Configuration Section

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run; and the SPECIAL-NAMES paragraph, which relates the system-names used by the compiler to the mnemonic names used in the source-program.

3.2.1 The SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled. Since this is fixed information, this paragraph is only used for documentation.

General Format

```
SOURCE-COMPUTER. computer-name.
```

The computer name in the preceeding format should be DATAPOINT.

3.2.2 The OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer upon which the object program is to execute. The format for this paragraph is as follows:

General Format

```
OBJECT-COMPUTER. computer-name
[, MEMORY SIZE integer CHARACTERS]
[, USING DEBUGGER].
```

The computer-name in the preceeding format should be DATAPOINT. The memory-size clause should specify the number of bytes required by the object program only, and should not include the storage required for the operating system. If the memory-size clause is omitted, the compiler assumes the object program will be executed on the same system that is used to compile the source program.

The USING DEBUGGER clause specifies that the execution-time debugger is to be invoked. This permits the user to stop execution at specified statements and to display or modify the values of data items.

3.2.3 The SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph provides a means of relating system-names to user specified mnemonic-names.

General Format

```
SPECIAL-NAMES.
          [, system-name-1 IS mnemonic-name]...
          [, system-name-2 [IS mnemonic-name ,]
            {ON STATUS IS condition-name-1
              [, OFF STATUS IS condition-name-2]
            | OFF STATUS IS condition-name-2
              [, ON STATUS IS condition-name-1]}]...
            [, QUOTE IS APOSTROPHE]
            [, CURRENCY SIGN IS literal]
            [, DECIMAL-POINT IS COMMA].
```

The SPECIAL-NAMES paragraph is required if mnemonic-names or condition-names, the DECIMAL-POINT clause, the CURRENCY SIGN clause, or the QUOTE clause are used.

The following system-names are legal for system-name-1: CONSOLE, C01, and CSP. Each may be assigned to a mnemonic-name once. The mnemonic-name assigned for CONSOLE may be used in the ACCEPT and DISPLAY statements. The mnemonic-names assigned for C01 (skip to top of page), and CSP (suppress spacing) may be used in the ADVANCING option of a WRITE statement.

System-name-2 may be one of SW-0, SW-1, ..., SW-7. If system-name-2 is used, at least one condition-name must be associated with it. The status of the switch is specified by condition-names and interrogated by testing the condition-names (see Section 5.4.5, Switch-Status Condition).

The clause QUOTE IS APOSTROPHE means that the function of the quote character is assumed by the apostrophe character. If the CURRENCY SIGN clause and the QUOTE clause are both used, the QUOTE clause must precede the CURRENCY SIGN clause.

The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters:

a. digits 0 through 9;
b. alphabetic characters A, B, C, D, P, R, S, V, X, Z; or the space;
c. special characters '*', '+', '-', ',', '.', ';', '(', ')'; or the quote character.

If this clause is not present, only the currency symbol '$' is used in the PICTURE clause.

The clause DECIMAL-POINT IS COMMA means that the function of comma and period are exchanged in the PICTURE clause character-string and in numeric literals.

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE L	C	A	B	COBOL STATEMENT	R	IDENTIFICATION
				ENVIRONMENT DIVISION.		
				CONFIGURATION SECTION.		
				SOURCE-COMPUTER. DATAPOINT-5500		
				OBJECT-COMPUTER. DATAPOINT-5500		
				SPECIAL-NAMES.		
				SW-Ø IS ABBREY-SWITCH		
				ON IS ON-SWITCH		
				OFF IS OFF-SWITCH		
				CURRENCY IS "#".		

Figure 3-1. Example of SPECIAL-NAMES paragraph within ENVIRONMENT DIVISION

3.3 The Input-Output Section

The Input-Output Section deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph which names and associates the files with external media; and the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

3.3.1 The FILE-CONTROL Paragraph

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows particular hardware assignments.

General Format

```
FILE-CONTROL. {SELECT file-name
              ASSIGN TO [integer-1] device-name
                [, device-name-2]...
                [FOR MULTIPLE {REEL | UNIT}]
              [, RESERVE {integer-2 | NO}
                ALTERNATE {AREA | AREAS}]
              [, {FILE-LIMIT IS | FILE-LIMITS ARE}
                {data-name-1 | literal-1} THRU
                {data-name-2 | literal-2}]
              [, ACCESS MODE IS {SEQUENTIAL | RANDOM}]
              [, PROCESSING MODE IS SEQUENTIAL]
              [, ACTUAL KEY IS data-name-3
              | , NOMINAL KEY IS data-name-3]
              [, RECORD KEY IS data-name-4].}...
```

All integers must be unsigned.

The FILE-CONTROL paragraph is required when the Input-Output Section header is present.

3.3.2 The SELECT Clause

Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph following the keyword SELECT. Each selected file must have a File Description or Sort File Description entry in the Data Division.

3.3.3 The ASSIGN Clause

Integer-1 indicates the number of input-output units of a given medium assigned to the file-name. In this system, integer-1, the device-name series, and the MULTIPLE REEL/UNIT clauses are treated as commentary. All files used in the program must be assigned to an input or output medium (device-name). Device-names may be chosen as follows:

Disk Files

The general format for a mass storage device-name is:

DISK-organization{-D-name-ext-drive | -A}

where organization is V for GEDIT-type files, F for fixed-blocked files, I for index-sequential files, and R for files used in RERUN; and where D stands for defined external name, and A for execution-time name assignment. If the D-form is used, a standard DOS file-name: name/ext:drive must be converted into name-ext-drive. As usual the drive may be left unspecified; however the file-name extension must be specified.

The ACCESS MODE and PROCESSING MODE clauses must be given for mass storage files. The treatment of mass storage devices in a sequential access mode is logically equivalent to the treatment of a tape file.

Files on Tape Devices

The system names for tape devices are: TAPE-A (industry-compatible 9-track tape in ASCII code), TAPE-E (9-track tape in EBCDIC code), CASSETTE-1 (rear cassette), and CASSETTE-2 (front cassette). For magnetic tape files, the maximum block size is 1057 bytes. For cassette files, the maximum block length is 250 bytes.

Other Devices

Datapoint COBOL also supports the following input-output devices: SERVO-PRINTER and LOCAL-PRINTER (servo and local printers whose maximum record size is 132 bytes), READER (card reader whose maximum record size is 80 bytes), and CONSOLE (keyboard/display whose maximum record size is 78 bytes).

3.3.4 The RESERVE Clause

The RESERVE clause is treated as commentary.

3.3.5 The FILE LIMIT Clause

The FILE-LIMIT clause is treated as commentary.

3.3.6 The ACCESS MODE Clause

For the ACCESS MODE SEQUENTIAL clause, the mass storage records are obtained or placed sequentially. That is, the next logical record is made available on a READ statement execution or a specific logical record is placed into the file on a WRITE statement execution.

For the ACCESS MODE RANDOM clause, the ACTUAL KEY or NOMINAL KEY entry must be specified. The Mass Storage Control System obtains each record randomly. That is, the specified logical record (located using the KEY data-name contents) is made available from the file on a READ statement execution or is placed in a specific location on the file (using KEY data-name contents) on a WRITE statement execution.

3.3.7 The PROCESSING MODE Clause

For the PROCESSING MODE SEQUENTIAL clause, the mass storage records are processed in the order in which they are accessed.

3.3.8 The ACTUAL KEY Clause

The ACTUAL KEY clause must be specified for fixed organized mass storage files which are accessed randomly. The data item must be described as numeric with no decimal positions. The data item must be properly set before a SEEK, READ, WRITE, or REWRITE statement is executed which uses the KEY value. The value of an ACTUAL KEY is the record number starting from one(1).

3.3.9 The NOMINAL KEY Clause

The NOMINAL KEY clause specifies a search argument for indexed files. The clause is required for index-sequential mass storage files being accessed randomly or being used with a START statement. The data item must be set before a SEEK, START, READ, WRITE, or REWRITE statement is executed which uses the KEY value.

3.3.10 The RECORD KEY Clause

The RECORD KEY clause is required for all index-sequential files. This clause specifies the location in the record for the search key. The RECORD KEY must be contained in the record itself.

3.4 The I-O CONTROL Paragraph

The I-O-CONTROL paragraph specifies the input-output techniques, the points at which rerun is to be established, the memory area which is to be shared by different files.

General Format

```
I-O-CONTROL. [; RERUN ON device-name]
               EVERY integer RECORDS OF file-name-1]...
             [; SAME [SORT | RECORD] AREA
               FOR file-name-2 {, file-name-3}...]...
             [; APPLY CORE-INDEX TO data-name ON file-name-4
               {, file-name-5}...]... .
```

The I-O-CONTROL paragraph is optional.

3.4.1 The RERUN Clause

The RERUN clause specifies where the rerun information is recorded and when the memory dump occurs. Memory dumps are written on a separate rerun disk file, as specified by the device-name given in the RERUN clause.

Rerun points can be established when a number of records (specified by integer) of an input or output file (file-name-1) have been processed.

3.4.2 The SAME AREA Clause

The SAME AREA clause specifies that two or more files are to use the same memory area during processing. The area being shared includes all storage areas (including alternate areas) assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

More than one SAME AREA clause may be included in a program, however, a file-name must not appear in more than one SAME AREA clause.

See Section 6.2.1.2, The I-O-Control Paragraph, for a description of the SAME SORT AREA and SAME RECORD AREA clauses.

3.4.3 The APPLY Clause

The APPLY CORE-INDEX clause is treated as commentary.

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE		COBOL STATEMENT	IDENTIFICATION
	A B		
		INPUT-OUTPUT SECTION.	
		FILE-CONTROL.	
		SELECT PRINT-FILE ASSIGN TO LOCAL-PRINTER.	
		SELECT INPUT-FILE ASSIGN TO DISK-V-DATA-TXT	
		ACCESS MODE IS SEQUENTIAL.	
		SELECT INPUT-FILE-2 ASSIGN TO DISK-1-D-ISAM-IS1	
		ACCESS MODE IS RANDOM	
		NOMINAL KEY IS INPUT-KEY	
		RECORD KEY IS RECORD-NO.	

Figure 3-2. Example of the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

CHAPTER 4. DATA DIVISION

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed falls into three categories:

a. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
b. That which is developed internally and placed into intermediate or working-storage, or placed into specific format for output reporting purposes, and constants defined by the user.
c. That which is developed internally by some other program and is referred to from the current one.

Data Division Organization

The Data Division, which is one of the required divisions in a program, is subdivided into sections. These are the File, Working-Storage, and Linkage Sections.

The File Section defines the contents of data files stored on an external medium. Each file is defined by a file description followed by a record description or a series of record descriptions. The Working-Storage Section describes records and noncontiguous data items which are not part of external data files but are developed and processed internally or data items whose values are assigned in the source program and do not change during the execution of the object program. The Working-Storage Section may specify both logical records and noncontiguous items. The Linkage Section describes records and noncontiguous data items which are defined in some other program, and which are referred to from this program.

Data Division Structure

The skeletal format of the Data Division is as follows:

```
DATA DIVISION.
    [FILE SECTION.
    {file description entry
    {record description entry}...}...]
    [WORKING-STORAGE SECTION.
    [data description entry]...
    [record description entry]...]
    [LINKAGE SECTION.
    [data description entry]...
    [record description entry]...]
```

The Data Division is identified by and must begin with the header, DATA DIVISION. The section header for the File Section is followed by one or more sets of entries composed of a File Description entry, followed by associated Record Description entries. The working-storage header is followed by Data Description entries for non-contiguous items, followed by Record Description entries. The linkage section is structured like the working-storage section. Each of the sections of the Data Division is optional and may be omitted from the source program if not needed.

4.1 File Section

In a COBOL program the File Description entries (FD and SD) represent the highest level of organization in the File Section. The maximum number of File Description entries (FD and SD) is thirty-two (32). The File Section header is followed by a File Description entry consisting of a level indicator (FD or SD), a data-name and a series of independent clauses. These clauses specify the size of the logical and physical records, and values of label items and the names of the data records which comprise the file. The entry itself is terminated with a period. (See Section 6.2.2.1 for SD.)

A Record Description entry consists of a set of Data Description entries which describe the characteristics of a particular record. Each Data Description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A Record Description entry has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in Concept of Levels (Section 1.5.2) while the elements allowed in a Record Description are shown in the Data Description skeleton.

4.1.1 The Complete File Description Entry Skeleton

The File Description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

General Format

```
FD file-name
  [; BLOCK CONTAINS integer-1
    {RECORDS | CHARACTERS}]
  [; DATA {RECORD IS | RECORDS ARE}
    data-name-1 [, data-name-2]...]
  ; LABEL {RECORD IS | RECORDS ARE}
    {STANDARD | OMITTED}
  [; LINAGE IS integer-2 LINES
    [WITH FOOTING AT integer-3]]
  [; RECORD CONTAINS [integer-4 TO]
    integer-5 CHARACTERS]
  [; VALUE OF data-name-3 IS literal-1
    [, data-name-4 IS literal-2]...].
```

The level indicator FD identifies the beginning of a File Description and must precede the file-name. All semicolons are optional in the File Description but the entry must be terminated by a period. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial.

4.1.2 The BLOCK CONTAINS Clause

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format

```
BLOCK CONTAINS integer {CHARACTERS | RECORDS}
```

Integer must be a positive integer.

When the CHARACTERS option is used, the physical record size is specified in terms of the number of characters in Standard Data Format contained within the physical record, regardless of the types of characters used to represent the items within the physical record. In this case integer represents the exact size of the physical record.

4.1.3 The DATA RECORDS Clause

The DATA RECORDS clause serves only as documentation for names of data records with their associated files.

General Format

```
DATA {RECORD IS | RECORDS ARE}
        data-name-1 [, data-name-2]...
```

Both data-name-1 and data-name-2 are the names of data records and must have 01 level numbers.

The presence of more than one data-name indicates that the file contains more than one type of data record. These records need not have the same description. The order in which they are listed is not significant.

Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

4.1.4 The LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present, and if present, identifies the label.

General Format

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

This clause is required in every File Description entry. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned. STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the standard label specifications.

4.1.5 The LINAGE Clause

The LINAGE clause specifies the length of the logical page and the location within the page of the footing area.

General Format

LINAGE IS integer-1 LINES
[WITH FOOTING AT integer-2]

Integer-1 specifies the exact length of the logical page.

When the FOOTING option is used, the footing area begins at the line specified by integer-2. Thus integer-2 must be greater than 0 and less than integer-1.

This clause generates the special register LINAGE-COUNTER. This clause must be specified if the END-OF-PAGE option is used in conjunction with a WRITE statement.

4.1.6 The RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of data records.

General Format

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

Integer-1 and integer-2 must be positive integers.

The size of each data record is completely defined within the Record Description entry, therefore this clause is never required. When present, however, the following notes apply:

(1) Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.
(2) The size is specified in terms of the number of characters in Standard Data Format contained within the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in all variable length items subordinate to the record. This sum may be different from the actual size of the record (see Section 4.4.9, The SYNCHRONIZED Clause, and Section 4.4.11, The USAGE Clause.)

4.1.7 The VALUE OF Clause

The VALUE OF clause particularizes the description of an item in the label records associated with a file, and serves only as documentation.

General Format

```
VALUE OF data-name-1 IS literal-1
  [, data-name-2 IS literal-2]...
```

A figurative constant may be substituted in the format above wherever a literal is specified.

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE		COBOL STATEMENT	IDENTIFICATION
	A B		
		DATA DIVISION.	
		FILE SECTION.	
	FD	PRINT-FILE	
		LABEL RECORD IS STANDARD	
		DATA RECORD IS PRINT-REC DUMMY-REC.	
	01	PRINT-REC PICTURE X(120).	
	01	DUMMY-REC.	
		02 FLD-1 PICTURE X(80).	
		02 FLD-2 PICTURE X(30).	
		02 FLD-3 PICTURE X(10).	
	FD	ISAM-FILE-1	
		LABEL RECORDS STANDARD	
		RECORD CONTAINS 65 CHARACTERS	
		DATA RECORD IS ISAM-REC.	
	01	ISAM-REC.	
		02 RECORD-NO-1 PICTURE 999 USAGE COMP.	
		02 DATA-FIELD PICTURE XXX.	
		02 FILLER PICTURE X(59).	

Figure 4-1. Example of FILE SECTION of the DATA DIVISION.

4.2 Working-Storage Section

The Working-Storage Section is composed of the section header, followed by data description entries of noncontiguous Working-Storage items and record description entries in that order. Each Working-Storage record name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

4.2.1 Noncontiguous Working-Storage

Items in Working-Storage which bear no hierarchic relationship to one another need not be grouped in records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number, 77.

The following data clauses are required in each level 77 data description entry:

a. level-number 77.
b. data-name.
c. The PICTURE clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

4.2.2 Working-Storage Records

Data elements in Working-Storage which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of data description.

4.2.3 Initial Values

The initial value of any item in the Working-Storage Section is specified by using the VALUE clause of the data description.

4.2.4 The Skeletal Format of the Working-Storage Section

```
WORKING-STORAGE SECTION.
    77 data-description entry
         .
         .
         .
    77 data-description entry
    01 data-description entry
        02 data-description entry
         .
         .
         .
    01 data-description entry
        02 data-description entry
            03 data-description entry
```

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE L	C	A	B	COBOL STATEMENT	R	IDENTIFICATION
			WORKING-STORAGE SECTION.			
		77	ONE PICTURE 9 VALUE 1.			
		77	TWO PICTURE 9 VALUE 2.			
		77	COUNT PICTURE 999 VALUE ZERO.			
		01	ALPHA PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ"			
		01	PRTLINE PICTURE X(132) VALUE SPACES.			
		01	HEADER.			
			02 FILLER PICTURE X(50) VALUE SPACES.			
			02 HDR-1 PICTURE X(15) VALUE "DATAPOINT COBOL".			
			02 FILLER PICTURE X(67) VALUE SPACES.			

Figure 4-2. Example of WORKING-STORAGE SECTION of the DATA DIVISION

4.3 Linkage Section

The Linkage Section is used to describe data defined in other COBOL source programs to which reference is made in this program. The Linkage Section has the same structure as the Working-Storage Section, and the same data-description entries may be used except for the VALUE clause. In the Linkage Section the VALUE clause may only be used with condition-names.

4.4 Data Description

4.4.1 The Complete Data Description Entry Skeleton

A data description entry specifies the characteristics of a particular item of data.

Format 1

```
level-number {data-name-1 | FILLER}
  [; REDEFINES data-name-2]
  [; BLANK WHEN ZERO]
  [; {JUSTIFIED | JUST} RIGHT]
  [; OCCURS integer TIMES
    [; INDEXED BY index-name-1
      [, index-name-2]...]]
  [; {PICTURE | PIC} IS character-string]
  [; {SYNCHRONIZED | SYNC} [LEFT | RIGHT]]
  [; [USAGE IS]
    {COMPUTATIONAL | COMP | DISPLAY
      | DATABUS | EBCDIC | INDEX}]
  [; VALUE IS literal].
```

Format 2

```
88 condition-name
  ; VALUE IS literal.
```

All semicolons and commas are optional in the data description but the entry must be terminated by a period.

Format 1 is used for record descriptions in all sections of the Data Division, and for non-contiguous elementary items in the Working Storage and Linkage Sections.

(1) Level-number in Format 1 may be any number from 1-49, or 77.

(2) The clauses may be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

(3) The PICTURE clause must be specified for every elementary item except an index data item, for which this clause is prohibited from being used. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO, must not be specified except at the elementary item level.

(4) The maximum length of any item, group or elementary, is

restricted to 32,767 characters.

DATAPOINT CORPORATION

COBOL Coding Form

PAGE OF

SYSTEM

PROGRAM

PROGRAMMER

DATE

SEQUENCE		COBOL STATEMENT	IDENTIFICATION
L	C A	B R	
	01	CHARACTER-SET PICTURE X(51) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ
-		" 0123456789 +-*/#$,.;()=+".
	01	COMPLETE-01.
		02 COMPLETE-F.
		03 FILLER PICTURE X(90) VALUE SPACES.
		03 FL-EQUAL PICTURE $(3),$$$.99 VALUE " $1,111.11".
		02 COMPLETE-FORMAT
		REDEFINES COMPLETE-F
		JUSTIFIED RIGHT
		PICTURE X(5)
		OCCURS 20 TIMES
		USAGE IS DISPLAY.
		02 MORE-COMPLETE-FORMAT
		BLANK WHEN ZERO
		PICTURE IS 9
		SYNCHRONIZED RIGHT
		DISPLAY
		VALUE IS "5".

Figure 4-3. Example of Format 1 (record descriptions).

Format 2 is used for each condition-name. Each condition-name requires a separate entry with level number 88. Format 2 contains the name of the condition and the value associated with the condition-name. The condition-name entries for a particular conditional-variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level number except the following:

(1) Another condition-name.
(2) A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).
(3) An index data item (see Section 6.1.1.2, The USAGE Clause).

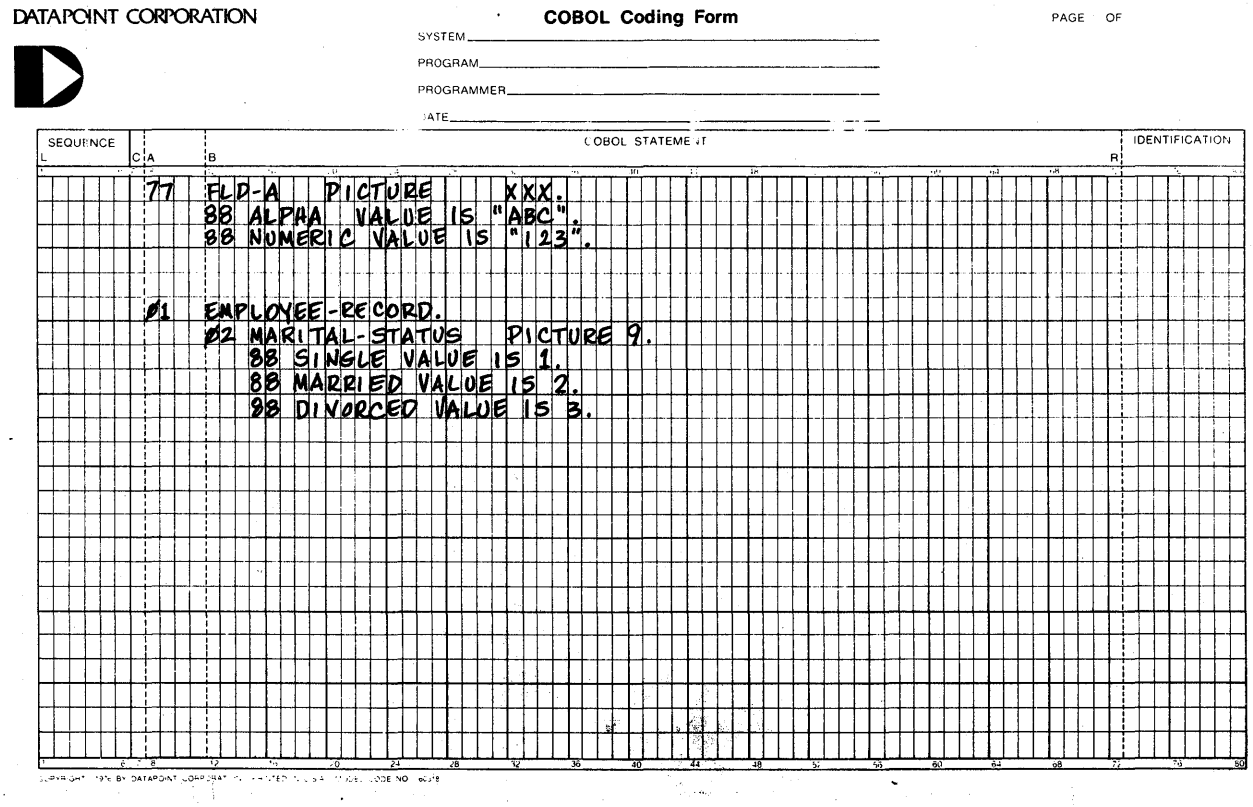


Figure 4-4. Example of Format 2 (condition names).

4.4.2 Level-Number

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

General Format

```
level-number
```

A level-number is required as the first element in each data description entry.

Data description entries subordinate to an FD or SD entry may have level-numbers with the values 01-49, or 88. (See Section 4.1.1 for FD; Section 6.2.2.1 for SD.)

The level-number 01 identifies the first entry in each Record Description. Multiple level 01 entries subordinate to a particular level indicator in the File Section represent implicit redefinitions of the same area.

Special level-numbers have been assigned to certain entries where there is no real concept of level:

(1) Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only in Format 2 of the Data Description Skeleton.
(2) Level-number 77 is assigned to identify noncontiguous Working-Storage and Linkage items.

4.4.3 The data-name or FILLER Clause

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of a logical record that cannot be referred to directly.

General Format

```
{data-name | FILLER}
```

In the File, Working-Storage, or Linkage Sections a data-name or the key word FILLER must be the first word following the level-number in each data description. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to directly.

4.4.4 The REDEFINES Clause

The REDEFINES clause allows the same computer storage area to contain different data items.

General Format

level-number data-name-1; REDEFINES data-name-2

The REDEFINES clause must immediately follow data-name-1. The level-numbers of data-name-1 and data-name-2 must be identical, but must not be 88. This clause must not be used in level 01 entries in the File Section. Implicit redefinition is provided by the DATA RECORDS clause in the File Description entry. (See Section 4.1.3.)

Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered. When the level-number of data-name-1 is other than 01, it must specify a storage area of the same size as data-name-2. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must follow the entries defining the area being redefined, without intervening entries that define new storage areas. Multiple redefinitions of the same storage area must all use the data-name of the entry that originally defined the area.

The data description entry for data-name-2 cannot contain a REDEFINES or an OCCURS clause, nor can data-name-2 be subordinate to an entry which contains a REDEFINES or an OCCURS clause.

The entries giving the new description of the storage area must not contain any VALUE clauses, except in condition-name entries.

4.4.5 The BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

General Format

BLANK WHEN ZERO

The BLANK WHEN ZERO clause can be used only for an

elementary item whose PICTURE is specified as numeric or numeric edited. (See The PICTURE Clause, below.) When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

4.4.6 The JUSTIFIED Clause

The JUSTIFIED clause specifies non-standard positioning of data within a receiving data item.

General Format

{JUSTIFIED | JUST} RIGHT

The standard rules for positioning data within an elementary item are:

(1) If the receiving data item is described as numeric
   a. The data item is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.
   b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its right-most character and is aligned as in a above.

(2) If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where the editing requirements cause replacement of the leading zeroes.

(3) If the receiving data item is alphanumeric (other than numeric edited data item) or alphabetic, the sending data is moved to the receiving character positions and aligned at the left-most character position in the data item with space fill or truncation to the right.

When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the left-most characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the right-most character position in the data item with space fill.

The JUSTIFIED clause can be specified only at the elementary item level. The JUSTIFIED clause cannot be specified for a numeric edited data item or for an item described as numeric.

JUST is an abbreviation for JUSTIFIED.

4.4.7 The PICTURE Clause

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format

{PICTURE | PIC} IS character-string

A PICTURE clause can only be used at the elementary item level.

A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item. The maximum number of symbols allowed in the character-string is 30. The PICTURE clause must be specified for every elementary item except an index data item, for which this clause is prohibited from being used.

PIC is an abbreviation for PICTURE.

Categories of Data

There are five categories of data that can be described with a PICTURE clause: Alphabetic, Numeric, Alphanumeric, Alphanumeric Edited, and Numeric Edited.

(1) To define an item as Alphabetic:

a. Its PICTURE character-string can only contain the symbol 'A';
b. Its contents when represented in Standard Data Format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set; and
c. Its length must be less than 32,768 characters.

(2) To define an item as Numeric:

a. Its PICTURE character-string can only contain the symbols '9', 'P', 'S', 'V', '.', '+', and '-';

b. Its contents when represented in Standard Data Format must be a valid combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9', and the item may contain both an operational sign and an actual decimal point; and
c. Its length must be less than 19 characters.

(3) To define an item as Alphanumeric:

a. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or 9's does not define an Alphanumeric item;
b. Its contents when represented in Standard Data Format are any allowable characters in the ASCII character set; and
c. Its length must be less than 32,768 characters.

(4) To define an item as Alphanumeric Edited:

a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0'; and
   1) The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X'; or
   2) The character-string must contain at least one '0' (zero) and at least one 'A';
b. Its contents when represented in Standard Data Format are any allowable characters in the ASCII character set; and
c. Its length must be less than 32,768 characters.

(6) To define an item as Numeric Edited:

a. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency sign. The allowable combinations are determined from the order of precedence of symbols and the editing rules. The maximum number of digit positions that may be represented in the character-string is 18;
b. The contents of the character positions of these symbols which are allowed to represent a digit in Standard Data Format must be one of the numerals; and
c. Its length must be less than 256 characters.

Item Size

The size of an elementary item, where size means the number of character positions occupied by the elementary item in the Standard Data Format, is determined by the number of allowable symbols that represent character positions.

Repetition of Symbols

An integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '0', '+', '-', or the currency sign indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

Symbols Used in a PICTURE Character-String

The functions of the symbols used to describe an elementary item are explained as follows:

A - Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

B - Each 'B' in the character-string represents a character position into which the space character will be inserted.

P - The 'P' indicates an assumed decimal scaling position and is used to specify the location of an 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. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or items which appear as operands in arithmetic statements. The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description. Since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are left-most PICTURE characters and to the right of 'P's if 'P's are right-most PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the left-most or right-most character within such a PICTURE description.

S - The letter 'S' is used in a character-string to indicate the presence of an operational sign and must be written as the left-most character in the PICTURE. The 'S' is not counted in

determining the size of the elementary item.

V - The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the right-most symbol in the string the 'V' is redundant.

X - Each 'X' in the character-string is used to represent a character position which contains any allowable character from the ASCII character set.

Z - Each 'Z' in a character-string may only be used to represent the left-most leading numeric character positions which will be replaced by a space character when the contents of that character position are zero. Each 'Z' is counted in the size of the item.

9 - Each '9' in the character-string represents a character position which contains a numeral and is counted in the size of the item.

0 - Each '0' (zero) in the character-string represents a character position into which the numeral zero will be inserted. The '0' is counted in the size of the item.

, - (comma) Each ',' in the character-string represents a character position into which the character ',' will be inserted. This character position is counted in the size of the item. The insertion character ',' must not be the last character in the PICTURE character-string.

. - (period) When the character '.' appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will either be inserted in receiving data items, or expected in sending data items. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

+, -, CR, DB - These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed in receiving data items. In sending data items an initial + or - represents an explicit operational sign. In this case + indicates that a plus sign represents positive values and a minus sign represents negative ones, whereas use of - indicates that a space represents positive values and a minus sign represents negative ones. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data-item.

* - Each asterisk, '*', in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each '*' is counted in the size of the item.

$ - (currency symbol) The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol is represented in the character-string by either the dollar sign, '$', or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

NON-EDITING PICTURE CLAUSES

Picture	Value In Memory	Value Used
9(5)	12345	12345
999V99	67890	678.90
S999V99	56789	(-)567.89
X(5)	ABCDE	ABCDE
X(5)	12345	12345
99X99	12.34	12.34
99AAA99	12MAY76	12MAY76
99PPPV	34	34000
VPPP99	12	.00012

Figure 4-5. Example of PICTURE clauses.

Editing Rules

There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

(1) Simple insertion
(2) Special insertion
(3) Fixed insertion
(4) Floating insertion

There are two types of suppression and replacement editing:

(1) Zero suppression and replacement with spaces
(2) Zero suppression and replacement with asterisks

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

Category	Type of Editing
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple Insertion, 0 and B
Numeric Edited	All, subject to rules below

Floating Insertion editing and editing by Zero Suppression and Replacement are mutually exclusive in a PICTURE clause. Only one type of Replacement may be used with Zero Suppression in a PICTURE clause.

Insertion Editing

Simple Insertion Editing. The ',' (comma), 'B' (space) and 0 (zero) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Special Insertion Editing. The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. The result of Special Insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the right-most character positions that are counted in the size of the item. The symbol '+' or '-', when used, must be the left-most or right-most character position to be counted in the size of the item. The currency symbol must be the left-most character position to be counted in the size of the item except that it can be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Edit sign control symbols produce the following results depending upon the value of the data item:

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Floating Insertion Editing. The currency symbol and editing sign symbols '+' or '-' are the insertion characters and they are mutually exclusive as floating insertion character in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the allowable insertion characters to represent the left-most numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters embedded in the string of floating insertion characters or to the immediate right of this string are part of the floating string.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

The result of floating insertion editing depends upon the representation in the PICTURE character-string. If the insertion characters are only to the left of the decimal point the result is a single insertion character that will be placed in the character position immediately preceding the decimal point, or the first nonzero digit in the data represented by the insertion symbol string, whichever is further to the left in the PICTURE character-string.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of fixed

insertion characters being edited into the receiving data item, plus one for the floating character.

Zero Suppression Editing

The suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used, the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero the entire data item will be spaces if the suppression symbol is 'Z' or all '*', except for the actual decimal point, if the suppression symbol is '*'.

When the asterisk is used as the Zero Suppression symbol and the clause BLANK WHEN ZERO also appears in the same entry, the Zero Suppression Editing overrides the function of BLANK WHEN ZERO.

The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence Rules

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

'P', fixed insertion '+', and '-' appear twice. The first occurrence represents their use to the left of the PICTURE's numeric character positions and the second their use to the right of the PICTURE's numeric character positions. 'Z', '*', non-fixed insertion 'cs', '+', and '-' appear twice. The first occurrence represents the use before the decimal point position, the second the use after the decimal point position. At least one of the symbols 'A', 'X', 'Z', '9', or '*', or at least two of the symbols '+', '-', or 'cs' must be present in a PICTURE string. In the cases where the preceding Editing Rules and the following chart conflict, the stated rules have precedence.

EDITING EXAMPLES

Send Area		Receive Area	
Picture	Memory	Picture	Edited Field
9(5)	12345	$ZZ,ZZ9.99	$12,345.00
999V99	23456	$ZZ,ZZ9.99	$ 234.56
999V99	00034	$ZZ,ZZ9.99	$ 0.34
999V99	00005	$ZZ,ZZZ.99	$ .05
9(5)	00000	$ZZ,ZZZ.ZZ	
9(5)	00000	$ZZ,ZZZ.99	$ .00
9(5)	12345	$**,**9.99	$12,345.00
9(5)	00000	$**,***.**	
9(5)	23456	$$$,$$9.99	$23,456.00
999V99	34567	$$$,$$9.99	$345.67
9(5)	00000	$$$,$$9.99	$0.00
9(5)	00000	$$$,$$$.ZZ	
V99999	67543	$$$,$$9.99	$0.67
S9(5)	12345	-ZZZZ9.99	-12345.00
S9(5)	12345	-ZZZZ9.99	12345.00
S9(5)	12345	+ZZZZ9.99	+12345.00
S9(5)	12345	+ZZZZ9.99	-12345.00
S9(5)	12345	$$$$$$.99CR	$12345.00CR
S9(5)	12345	$$$$$$.99CR	$12345.00
AAA	MNP	ABABA	M N P
XXXX	RSPQ	XBBXXBBX	R SP Q
X(5)	BCDEF	XX0X0X0X	BC0D0E0F
9(5)	34567	99B999B000	34 567 000
9(5)	00000	ZZ99.99	00.00
S9(5)	12345	ZZ,ZZZ.ZZ+	12,345.00-

Figure 4-6. Examples of Editing.

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

4.4.8 The SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundary of the computer memory.

General Format

{SYNCHRONIZED | SYNC} [LEFT | RIGHT]

This clause may only appear with an elementary item.

SYNC is an abbreviation for SYNCHRONIZED.

The intent of the SYNCHRONIZED clause is to increase the processing efficiency for data items by aligning them on natural machine boundaries. There are no special boundaries for the

Datapoint processors and therefore the SYNCHRONIZED clause has no effect on a COBOL program which uses it.

The length of a data item is not changed by use of the SYNCHRONIZED clause.

4.4.9 The USAGE Clause

The USAGE clause specifies the format of a data item.

General Format

```
[USAGE IS]
  {COMPUTATIONAL | COMP | DISPLAY
  | DATABUS | EBCDIC | INDEX}
```

The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

This clause specifies the manner in which a data item is represented in the storage of the processor. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect type of character representation of the item.

If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is assumed to be DISPLAY.

The PICTURE of a COMPUTATIONAL item can contain only 9's, the operational sign character 'S', the implied decimal point character 'V', and one or more P's.

COMP is an abbreviation for COMPUTATIONAL.

A COMPUTATIONAL item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group itself is not COMPUTATIONAL (cannot be used in computations).

The USAGE IS DISPLAY clause indicates that the format of the data is a Standard Data Format.

For compatibility with the Databus and RPG-II language processors, USAGE IS DATABUS and USAGE IS EBCDIC may be used.

The PICTURE of a DATABUS item can contain only -'s and the decimal point character '.'. A DATABUS item represents a numeric value to be used in computations and must be numeric. If a group item is described as DATABUS, the elementary items in the group are DATABUS, but the group itself is not DATABUS.

The PICTURE of an EBCDIC item can contain only 9's, the operational sign character 'S', the implied decimal point character 'V', and one or more P's. An EBCDIC item represents a numeric value to be used in computations and must be numeric. If a group item is described as EBCDIC, the elementary items in the group are EBCDIC, but the group itself is not EBCDIC.

The meaning of USAGE IS INDEX is discussed in Section 6.1.1.2, The USAGE Clause.

Representations for Numeric Items

There are four forms of numeric representation which may be used in Datapoint COBOL - COMPUTATIONAL, DISPLAY, DATABUS, and EBCDIC. The COMPUTATIONAL form is directly used in arithmetic operations, while the others are converted to/from COMPUTATIONAL as required. A COMPUTATIONAL number of length N is represented by N-1 digits, followed by either a digit (0-9) or one of the letters P-Y. The letters are used to represented the signed digits, -0, -1, ..., -9. If the number has no operational sign or is non-negative, ordinary digits are used. In COMPUTATIONAL representation the location of the decimal point is implicit.

An EBCDIC number uses the RPG sign convention and leading zeroes may be represented as blanks. The characters, {, A-I, are used to represent the positively signed digits (+0, +1, ..., +9) in this format and are equivalent to the unsigned digits (0, ..., 9). The characters, }, J-R, are used to represent the negatively signed digits (-0, -1, ..., -9).

A DISPLAY number follows the same conventions as EBCDIC.

A DATABUS number is written with a floating leading minus sign if negative, and with no explicit sign character if non-negative. The decimal point for non-integral numbers must be explicitly represented. Initial zeros may be replaced with blanks.

4.4.10 The VALUE Clause

The VALUE clause defines the initial value of working-storage items.

General Format

VALUE IS literal

General Rules

The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. If the category of an elementary item is specified as numeric or alphabetic, it does not contradict the alphanumeric category of group items. The following rules apply:

(1) If the category of the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a working-storage item, the literal is aligned according to the alignment rules except that the literal must not have a value which would require truncation of non-zero digits.
(2) If the category of the item is Alphabetic or Alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned according to the alignment rules (see The JUSTIFIED Clause, above) except that the number of characters in the literal must not exceed the size of the item.
(3) All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause; for example, for PICTURE PPP99, the literal must be within the range .00000 through .00099.
(4) The function of the BLANK WHEN ZERO clause or any editing characters in a PICTURE clause have no effect on initialization of the item. The VALUE clause is the only clause that may (depending on its usage) provide initialization. Editing characters are included however, in determining the size of the item.

A figurative constant may be substituted for literal in the format.

Condition-Name Rules

In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two

clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

Data Description Entries Other Than Condition-Names

Rules governing the use of the VALUE clause differ with the respective section of the Data Division:

(1) In the File and Linkage Sections, the VALUE clause may be used only in condition-name entries.
(2) In the Working-Storage Section, the VALUE clause may be used in condition-name entries, and it may also be used to specify the initial value of any data item. It causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value is unpredictable.

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

The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

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

The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

CHAPTER 5. PROCEDURE DIVISION

The Procedure Division must be included in every COBOL source program. The Procedure Division contains procedures.

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified), or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

A section consists of a section header followed by one or more successive paragraphs. A section ends immediately before the next section-name or at the end of the Procedure Division.

A paragraph consists of a paragraph-name followed by one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term 'identifier' is defined as the word or words necessary to make unique reference to a data item.

Execution begins with the first statement of the Procedure Division. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

5.1 Procedure Division Structure

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION
          [USING identifier-1 [, identifier-2]...].
```

The Procedure Division must conform to one of the following formats:

```
 Format 1
PROCEDURE DIVISION
  [USING identifier-1 [, identifier-2]...].
{section-name SECTION [priority-number].
{paragraph-name. {sentence}...}...}...

 Format 2
PROCEDURE DIVISION
  [USING identifier-1 [, identifier-2]...]
{paragraph-name. {sentence}...}...
```

5.2 Statements and Sentences

There are three types of statements: imperative statements, conditional statements, and compiler directing statements.

There are three types of sentences: imperative sentences, conditional sentences, and compiler directing sentences.

5.2.1 Conditional Statements and Sentences

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this value. A conditional statement is one of the following:

```
IF

ADD                 (ON SIZE ERROR)
COMPUTE             (ON SIZE ERROR)
SUBTRACT            (ON SIZE ERROR)
MULTIPLY            (ON SIZE ERROR)
DIVIDE              (ON SIZE ERROR)
```

GO TO	(DEPENDING ON)
READ	(AT END)
WRITE	(AT END-OF-PAGE)
START	(INVALID KEY)
READ	(INVALID KEY)
WRITE	(INVALID KEY)
REWRITE	(INVALID KEY)
PERFORM	(UNTIL or VARYING)

In the above list, the options enclosed in parenthesis cause statements which are normally imperative to become conditional statements. A discussion of these options is included in the description of the imperative form of the statement.

A conditional sentence is a conditional statement optionally preceded by an imperative statement terminated by a period followed by a space.

5.2.2 Imperative Statements and Sentences

An imperative statement indicates a specific action to be taken by the object program.

An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement. An imperative statement may consist of a sequence of imperative statements each possibly separated from the next by a separator. Imperative statements are one of the following types of statement:

Arithmetic Statements
- ADD
- COMPUTE
- DIVIDE
- MULTIPLY
- SUBTRACT

Procedure Branching Statements
    GO TO
    ALTER
    PERFORM
    STOP
    EXIT

Data-Manipulating Statements
    MOVE
    EXAMINE

Input-Output Statements
    OPEN
    SEEK
    START
    READ
    WRITE
    REWRITE
    ACCEPT
    DISPLAY
    CLOSE

Table-Manipulating Statements
    SET

Program Linkage Statements
    CALL
    EXIT [PROGRAM]

Whenever an imperative-statement appears in the General Format of statements, the imperative statement refers to that sequence of consecutive imperative statements which is ended by a period or an ELSE associated with a previous IF verb. An imperative sentence is an imperative statement terminated by a period followed by a space.

5.2.3 Compiler Directing Statements and Sentences

A compiler directing statement is one of the fcllowing:

ENTER
NOTE
COPY

A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space.

5.3 Arithmetic Expressions

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by an arithmetic operator, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Arithmetic negation can be expressed by using a unary '-'. The permissable combinations of identifiers, literals, and arithmetic operators are given in Table 1.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

Arithmetic Operators

There are five binary arithmetic operators and one unary arithmetic operator that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space; however a unary operator must not be preceded by a space when it follows a left parenthesis.

Binary Arithmetic Operators	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operator	Meaning
-	The effect of multiplication by numeric literal -1.

Formation and Evaluation Rules

(1) Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and, within a nest of parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of operations is implied:

Unary -
**
* and /
+ and -

(2) When the order of a sequence of consecutive operations on the same hierarchical level is not completely specified by parentheses, the order of operations is from left to right.

(3) The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in Table 1, where:

a. The letter 'P' represents a permissable pair of symbols.
b. The character '-' represents an invalid pair.
c. Variable represents an identifier or literal.

First Symbol \ Second Symbol	Variable	* / ** + -	Unary -	(	)
Variable	-	P	-	-	P
* / ** + -	P	-	P	P	-
Unary -	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Table 1. Combination of Symbols in Arithmetic Expressions

(4) An arithmetic expression may only begin with the symbols '(', '-', or a variable and may only end with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

(5) Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. See, for example, the ADD statement, Section 5.6.5.

5.4 Conditions

A condition causes the object program to select between alternate paths of control depending upon the truth value of a test. Conditions are used in IF and PERFORM statements. A condition is one of the following:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
NOT condition
condition AND condition
condition OR condition

Any condition may be enclosed in parentheses. The truth value of a parenthesized condition is determined from the evaluation of the truth values of its constituents. A parenthesized condition is a condition in the sense of the last three items of the preceding list.

The construction

NOT condition

where condition is one of the first six types of conditions listed above, is not permitted if the condition itself contains NOT.

Conditions may be combined by logical operators. The logical operators must be preceded by a space and followed by a space. The meaning of the logical operators is as follows:

Logical Operator	Meaning
OR	Logical Inclusive Or
AND	Logical Conjunction
NOT	Logical Negation

Table 2 indicates the relationships between the logical operators and conditions, A and B. Table 3 indicates the way in which

conditions and logical operators may be combined.

Condition A	Condition B	Condition and Value A AND B	A OR B	NOT A
True	True	True	True	False
False	True	False	True	True
True	False	False	True	False
False	False	False	False	True

Table 2. Relationship of Conditions, Logical Operators, and Truth Values

First Symbol \ Second Symbol	Condition	OR	AND	NOT	(	)
Condition	-	P	P	-	-	P
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P	-	-	-	P	-
(	P	-	-	P	P	-
)	-	P	P	-	-	P

Table 3. Combinations of Conditions and Logical Operators

5.4.1 Relation Condition

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, or an arithmetic expression. Comparison of two numeric operands is permitted regardless of the format as specified in individual USAGE clauses. However, for all other comparisons the operands must have the same usage.

The general format for a relation condition is as follows:

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

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2, literal-2, or arithmetic-expression-2) is called the object of the condition.

The subject and the object may not both be literals.

The relational operators specify the type of comparison to be made in a relation condition. The relational operators must be preceded by a space and followed by a space. The meaning of the relational operators is as follows:

Meaning	Relational Operator
Greater than or	IS [NOT] GREATER THAN
not greater than	IS [NOT] >
Less than or	IS [NOT] LESS THAN
not less than	IS [NOT] <
Equal to or	IS [NOT] EQUAL TO
not equal to	IS [NOT] =

Comparison of Numeric Operands

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

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

Comparison of Nonnumeric Operands

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to the ASCII collating sequence.

The size of an operand is the total number of characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same, implicitly or explicitly.

There are two cases to consider: operands of equal size and operands of unequal size.

(1) Operands of Equal Size. If the operands are of equal size, characters in corresponding character positions of the two operands are compared starting from the high-order end through the low-order end.

    If all pairs of characters compare equally through the last

pair, the operands are considered equal when the low-order end is reached.

The first pair of unequal characters to be encountered is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

(2) Operands of Unequal Size. If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

5.4.2 Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero. The general format for a sign condition is as follows:

```
{identifier | arithmetic-expression}
          IS [NOT] {POSITIVE | NEGATIVE | ZERO}
```

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

5.4.3 Class Condition

The class condition determines whether the operand is numeric, that is, consists entirely of the characters 0, 1, 2, 3, ..., 9, with or without an operational sign, or alphabetic, that is consists entirely of the characters A, B, C, ..., Z, space. The general format for the class condition is as follows:

```
identifier IS [NOT] {NUMERIC | ALPHABETIC}
```

The usage of the operand being tested must be described, implicitly or explicitly, as DISPLAY.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic. If the record description of the item being tested does not contain an operational sign the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

The ALPHABETIC test cannot be used with an item whose record

description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

5.4.4 Condition Name Condition

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to the value associated with a condition-name. The general format for a condition-name condition is as follows:

condition-name

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if the value corresponding to the condition-name equals the value of its associated conditional variable.

5.4.5 Switch Status Conditon

A switch-status condition determines the on or off status of a software switch. The system-name and its associated ON or OFF value must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

condition-name

The result of the test is true if the switch value corresponds to the setting specified by the condition-name.

5.4.6 Evaluation Rules for Conditions

The evaluation rules for conditions are analogous to those given for arithmetic expressions (see Formation and Evaluation Rules in Section 5.3) except that the following hierarchy applies:

arithmetic expression
- all relational operators
- NOT
- AND
- OR

5.5 Conditioal Statements

5.5.1 The IF Statement

The IF statement causes a condition (see the preceding section, Conditions) to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format

```
IF condition; {statement-1 | NEXT SENTENCE};
  {ELSE statement-2 | ELSE NEXT SENTENCE}
```

Statement-1 and statement-2 represent either an imperative statement or an IF statement, and either may be followed by an imperative statement or an IF statement. The phrase ELSE NEXT SENTENCE may be omitted if it immediately precedes the terminal period of the sentence.

When an IF statement is executed, the following action is taken:

(1) If the condition is true, the statements immediately following the condition (represented by statement-1) are executed, control then passes implicitly to the next sentence.
(2) If the condition is false, either the statements following ELSE are executed or, if the ELSE clause is omitted the next sentence is executed.

When an IF statement is executed and the NEXT SENTENCE option is present, control passes explicitly to the next sentence depending on the truth value of the condition and the placement of the NEXT SENTENCE clause in the statement.

Statement-1 and statement-2 may contain an IF statement. In this case the IF statement is said to be nested. IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceeding IF that has not been already paired with an ELSE.

When control is transferred to the next sentence, either implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM statement.

5.6 Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation. The maximum size of each operand is eighteen (18) decimal digits.

In the statement descriptions that follow, several options appear frequently: the GIVING option, the ROUNDED option, and the SIZE ERROR option. In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

5.6.1 The GIVING Option

If the GIVING option is used, the value of the arithmetic statement is moved after computation to the identifier specified after the word GIVING. Thus this resultant identifier may be a numeric edited data item, as well as being a numeric item.

5.6.2 The ROUNDED Option

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant-identifier are represented by the character P in the picture for that resultant-identifier, rounding or truncation occurs relative to the right-most integer position for which storage is allocated.

5.6.3 The SIZE ERROR Option

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant-identifer, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

(1) If the SIZE ERROR option is not specified and a size error condition occurs, the value of the resultant-identifier affected will be unpredictable.

(2) If the SIZE ERROR option is specified and a size error condition occurs, then the value of the resultant-identifier affected by the size error is not altered. After completion of the execution of this operation, the imperative-statement in the SIZE ERROR option is executed.

5.6.4 Overlapping Operands

When a sending and a receiving item in an Arithmetic Statement or an EXAMINE or MOVE statement share a part of their storage areas, the result of the execution of such a statement is undefined.

5.6.5 The ADD Statement

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

Format 1

```
ADD {identifier-1 ¦ literal-1}
  [, identifier-2 ¦ , literal-2]...
  TO identifier-m [ROUNDED]
  [; ON SIZE ERROR imperative-statement]
```

Format 2

```
ADD {identifier-1 | literal-1} ,
  {identifier-2 | literal-2}
  [, identifier-3 | , literal-3]...
  GIVING identifier-m [ROUNDED]
  [; ON SIZE ERROR imperative-statement]
```

In Formats 1 and 2 each identifier must refer to an elementary numeric item, except that the identifier appearing to the right of the word GIVING may refer to a data item that contains editing symbols.

Each literal must be a numeric literal.

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

See Section 5.6.2, The ROUNDED Option; Section 5.6.3, The SIZE ERROR Option; and Section 1.4.6, Special Register; for a description of these functions.

If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value in identifier-m, and the result is stored in the resultant identifier-m.

If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m, the resultant-identifier.

5.6.6 The COMPUTE Statement

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

General Format

```
COMPUTE identifier-1 [ROUNDED] =
  {identifier-2 | literal | arithmetic-expression}
  [; ON SIZE ERROR imperative-statement]
```

The identifier-2 and literal options provide a method for setting the value of identifier-1, equal to the value of identifier-2 or literal-1. Literal must be a numeric literal.

Identifier-2 must refer to an elementary numeric item. Identifier-1 may contain editing symbols.

The arithmetic expression option permits the use of any meaningful combination of identifiers, numeric literals, and arithmetic operators, parenthesized as required. The COMPUTE statement allows the user to combine arithmetic operations without the restriction on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

The maximum size of each operand is eighteen decimal digits. See The ROUNDED Option and The SIZE ERROR Option above, and Special Register (Section 1.4.6).

5.6.7 The DIVIDE Statement

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

Format 1

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

Format 2

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

Format 3

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

Format 4

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

Format 5

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

When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient.

When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3.

When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3.

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

Each identifier must refer to a numeric elementary item, except in Formats 2 through 5, where the identifier that appears only to the right of the word GIVING may refer to a data item that contains editing symbols. Each literal must be a numeric literal. The maximum size of each operand is eighteen (18) decimal digits. See also The ROUNDED Option and The SIZE ERROR Option above, and Special Register (Section 1.4.6) for a description of these functions.

5.6.8 The MULTIPLY Statement

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

Format 1

```
MULTIPLY {identifier-1 | literal-1}
  BY identifier-2 [ROUNDED]
  [; ON SIZE ERROR imperative-statement]
```

Format 2

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

When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product.

When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3.

Each identifier must refer to a numeric elementary item, except in Format 2, where the identifier that appears to the right of the word GIVING may refer to a data item that contains editing symbols. Each literal must be a numeric literal. The maximum size of each operand is eighteen (18) decimal digits. See also The ROUNDED Option and The SIZE ERROR Option above, and Special Register (Section 1.4.6).

5.6.9 The SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more numeric data items from an item, and sets the value of an item equal to the result.

Format 1

```
SUBTRACT {literal-1 | identifier-1}
  [, literal-2 | , identifier-2]...
  FROM identifier-m [ROUNDED]
  [; ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT {literal-1 | identifier-1}
  [, literal-2 | , identifier-2]...
  FROM {literal-m | identifier-m}
  GIVING identifier-n [ROUNDED]
  [; ON SIZE ERROR imperative-statement]
```

In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m and the difference is stored as the new value of identifier-m.

In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n.

Each identifier must refer to a numeric elementary item except in Format 2, where the identifier that appears to the right of the word GIVING may refer to a data item that contains editing symbols. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from superimposing all operands, excluding the identifier that follows the word GIVING, aligned on their decimal points, must not contain more than eighteen digits. See The ROUNDED Option and The SIZE ERROR Option above, and Special Register (Section 1.4.6).

5.7 Procedure Branching Statements

5.7.1 The GO TO Statement

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

Format 1

```
GO TO [procedure-name-1]
```

Format 2

```
GO TO procedure-name-1 [, procedure-name-2]...
  , procedure-name-n DEPENDING ON identifier
```

Whenever a GO TO statement, represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been altered by an ALTER statement.

If procedure-name-1 is not specified in Format 1, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.

When, in Format 1, the GO TO statement is referred to by an ALTER statement the following rules apply:

(1) The GO TO statement must have a paragraph-name.
(2) The GO TO statement must be the only statement in the

paragraph.

If a GO TO statement represented by Format 1 appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative statements.

A GO TO statement represented by Format 2 causes control to be transferred to one of the specified procedures named procedure-name-1, procedure-name-2, etc., depending onthe value of the identifier being 1, 2, ..., n. If the value of identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then the GO TO statement has no effect.

Each procedure-name is the name of a paragraph or section in the Procedure Division of the program. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

5.7.2 The ALTER Statement

The ALTER statement modifies a predetermined sequence of operations.

General Format

```
ALTER procedure-name-1 TO
    [PROCEED TO] procedure-name-2
  [, procedure-name-3 TO
    [PROCEED TO] procedure-name-4]...
```

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

Each procedure-name-1, procedure-name-3, ... is the name of a paragraph that contains only one sentence consisting of a GO TO statement without the DEPENDING option. Each procedure-name-2, procedure-name-4, ... is the name of a paragraph or section in the Procedure Division.

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. (See Restrictions on Program Flow, Section 6.5.3.)

All other uses of the ALTER statement are valid.

5.7.3 The PERFORM Statement

The PERFORM statement is used to depart from the normal sequence of execution in order to execute one or more procedures either a specified number of times or until a specified condition is satisfied and to return control to the normal sequence.

Format 1

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

Format 2

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

Format 3

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

Format 4

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

Each procedure-name is the name of a section or paragraph in the Procedure Division. Each identifier represents a numeric elementary item described in the Data Division. In Format 2, and Format 4 with the AFTER option, each identifier represents a numeric item with no positions to the right of the assumed decimal point. Each literal represents a numeric literal. The word THROUGH may be substituted for THRU.

When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. An

automatic return to the statement following the PERFORM statement is established as follows:

(1) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.
(2) If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.
(3) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.
(4) If procedure-name-2 is specified and it is a section-name, then the return is after the last sentence of the last paragraph in this section.

There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more direct paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all these paths must lead.

If control passes to these procedures by means other than a PERFORM statement, control passes through the last statement of the procedure to the following statement as if no PERFORM statement mentioned these procedures.

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit.

Format 1 is the basic PERFORM statement. A procedure referred to by this type of PERFORM statement is executed once and then control passes to the statement following the PERFORM statement.

Format 2 is the TIMES option. When the TIMES option is used the procedures are performed the number of times specified by the initial value of identifier-1 or integer-1, for that execution.

When the PERFORM statement is executed, the value of integer-1 must be positive. If the initial value of identifier-1 is negative or zero, control passes immediately to the statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transfered to the statement following the PERFORM statement.

During execution of the PERFORM statement, reference to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

Format 3 is the UNTIL option. The specified procedures are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition is true at the time that the PERFORM statement is encountered, the specified procedures are not executed.

Format 4 is the VARYING option. This option is used to augment the value of one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion every reference to identifier as the object of the VARYING and FROM (starting value) phrases also refers to index-names. When index-names are used, the FROM and BY clauses have the same effect as in a SET statement. (See The SET Statement, Section 6.1.2.3.)

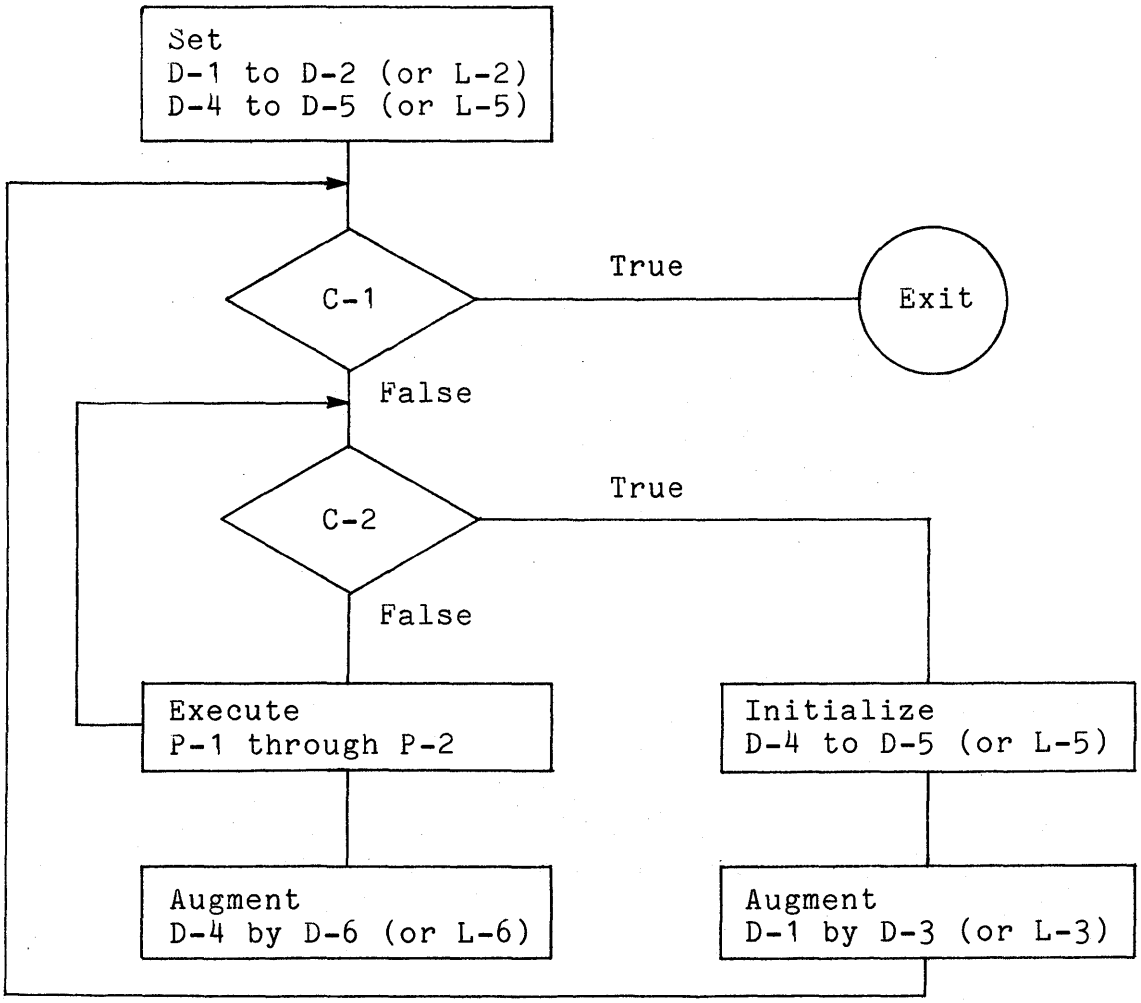
In Format 4, when one identifier is varied, identifier-1 is set equal to its starting value, identifier-2 or literal-2, when commencing the PERFORM statement; if the condition is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-1 is augmented by the specified increment or decrement, identifier-3, and condition-1 is evaluated again. The cycle continues until this expression is true; at which point, control passes to the statement following the PERFORM statement. If the condition is true at the beginning of execution of the PERFORM, control passes directly to the statement following the PERFORM statement.

In Format 4, when two identifiers are varied, identifier-1 and identifier-4 are set to their initial values, identifier-2 and identifier-5, respectively. During execution, these initial values must be positive. When commencing the PERFORM statement, condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, after which identifier-4 is

augmented by identifier-6 and condition-2 is evaluated again. This cycle of execution and augmentation continues until condition-2 is true. When this condition is true, identifier-4 is set to its initial value, identifier-5, identifier-1 is augmented by identifier-3 and condition-1 is reevaluated; identifier-3 and identifier-6 must not be zero; the PERFORM statement is completed if condition-1 is true; if not, these cycles continue until condition-1 is true.

During execution of the PERFORM statement, reference to index-names or identifiers of the FROM clause has no effect in altering the numbers of times the procedures are to be executed. Changing a value of the index-names or identifiers of the VARYING clause or identifiers of the BY clause, however, will change the number of times procedures are executed.
The following flow chart illustrates the logic of the PERFORM statement when two identifiers are varied.

Let:
Each D-i represent an identifier.
Each L-i represent a literal.
Each C-i represent a condition.
Each P-i represent a procedure-name.



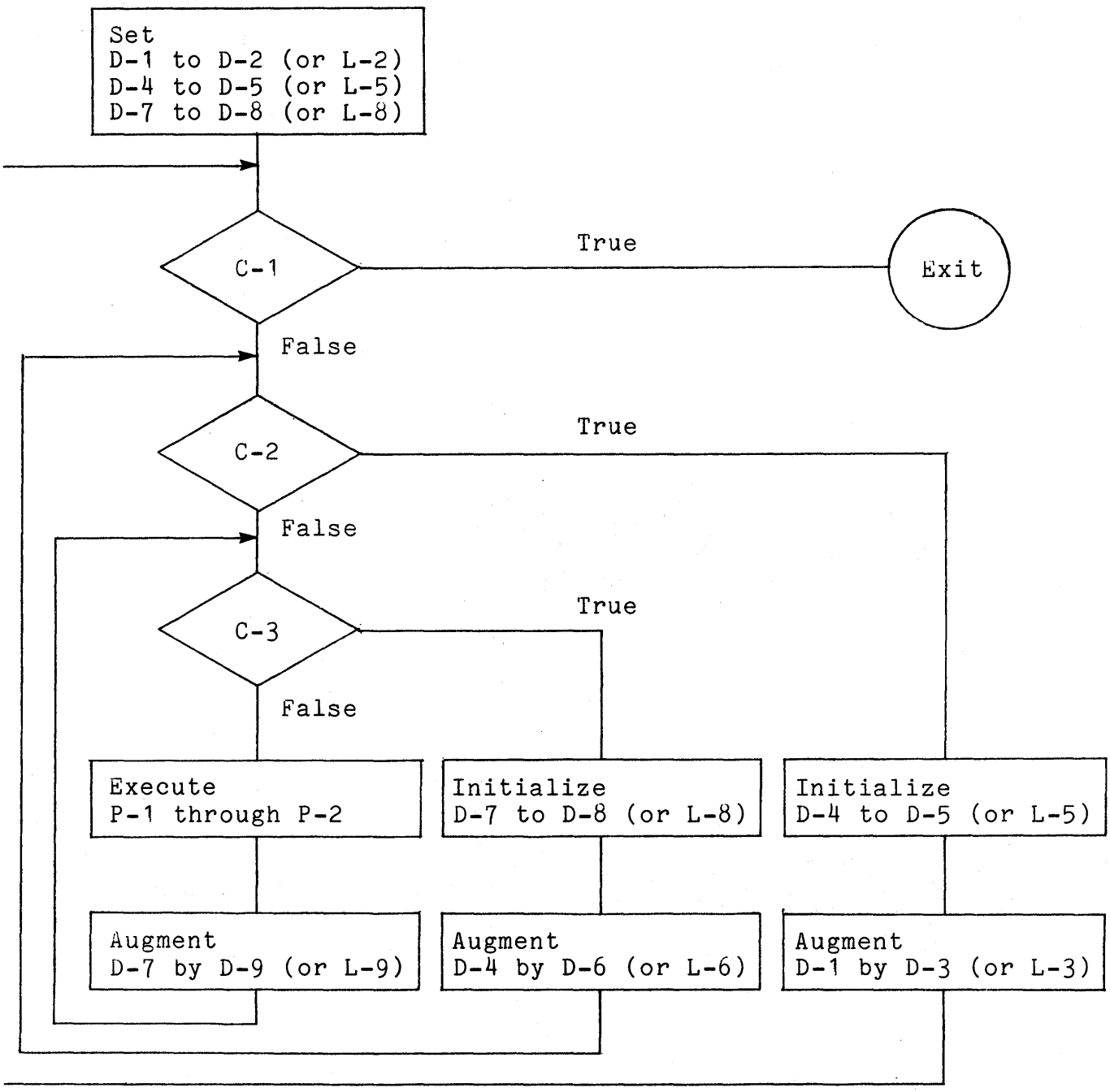
At the termination of the PERFORM statement identifier-4 contains its initial value, while identifier-1 has a value that exceeds the last used setting by one increment or decrement, as the case may be, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-1 and identifier-4 contain their initial values.

For three identifiers the mechanism is the same as for two

identifiers except that identifier-7 goes through a complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

The following flow chart illustrates the logic of the PERFORM statement when three identifiers are varied. Let:

Each D-i represent an identifier.
Each L-i represent a literal.
Each C-i represent a condition.
Each P-i represent a procedure-name.


Set
D-1 to D-2 (or L-2)
D-4 to D-5 (or L-5)
D-7 to D-8 (or L-8)
C-1
True
Exit
False
C-2
True
False
C-3
True
False
Execute
P-1 through P-2
Initialize
D-7 to D-8 (or L-8)
Initialize
D-4 to D-5 (or L-5)
Augment
D-7 by D-9 (or L-9)
Augment
D-4 by D-6 (or L-6)
Augment
D-1 by D-3 (or L-3)

After the completion of Format 4, identifier-4 and identifier-7 contain their initial values, while identifier-1 has a value that exceeds its last used setting by one increment or decrement, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-1, identifier-4, and identifier-7 all contain their initial values.

A PERFORM statement that appears in a section whose priority is less than the 50, can have within its range only the following:

(1) Sections each of which has a priority number less than 50, or
(2) Sections wholly contained in a single segment whose priority number is greater than 49. (See Restrictions on Program Flow, Section 6.5.3.)

A PERFORM statement that appears in a section whose priority number is equal to or greater than the 50, can have within its range only the following:

(1) Sections each of which has the same priority number as that containing the PERFORM statement, or
(2) Sections with a priority number that is less than the segment limit. (See Restrictions on Program Flow, Section 6.5.3.)

When a procedure-name in a segment with a priority number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority number, the segment referred to is made available in its initial state for each execution of the PERFORM statement. (See Restrictions on Program Flow, Section 6.5.3.)

5.7.4 The STOP Statement

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format

STOP {literal | RUN}

If the RUN option is used, then the object program makes a normal exit to DOS, the Disk Operating System. Any files open at this time will not be closed.

If a STOP statement with the RUN option appears in an imperative sentence, then it must appear as the only or last statement in a sequence of imperative-statements.

If the literal option is used, the literal is displayed on the console display and the operator is queried as to whether to continue execution or not. If he responds with the character 'N', the effect is the same as that of a STOP RUN statement. If he responds with the character 'Y' the object program continues with the execution of the next statement in sequence. The literal may be numeric or nonnumeric or may be any figurative constant, except ALL literal.

5.7.5 The EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

General Format

EXIT [PROGRAM].

It is sometimes necessary to transfer control to the end point of a series of procedures. This is normally done by transferring control to the next paragraph or section, but in some cases this does not have the required effect. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a PERFORM statement. The EXIT statement is provided to enable a procedure-name to be associated with such a point.

If control reaches an EXIT paragraph and no associated PERFORM statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

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

The PROGRAM option is used with the Sub-Program Feature. See The EXIT PROGRAM Statement in Section 6.4.3.

5.8 Data Manipulation Statements

5.8.1 The MOVE Statement

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

General Format

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

Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.

The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, etc. The notes referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, etc., is evaluated immediately before the data is moved to the respective data item.

An index data item cannot appear as an operand of a MOVE statement. (See The USAGE Clause, Section 6.1.1.2.)

Any move in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move between these categories:

(1a) The figurative constant SPACE, or a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

(1b) A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic item.

(1c) A numeric literal, or a numeric data item whose implicit decimal point is not immediately to the right of the least significant digit, must not be moved to an alphanumeric or alphanumeric edited data item.
(1d) All other elementary moves are legal and are performed according to the rules given in General Rule (2).

Any necessary conversion of data from one form of internal representation to another takes place during the legal elementary moves, along with any editing specified for the receiving data item. The following rules apply to legal elementary moves:

(2a) When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause (see The JUSTIFIED Clause, Section 4.4.6). If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.
(2b) When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero-filling takes place, except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, the excess digits are truncated. When a data item described as alphanumeric is the sending item, it is moved as though it was described as an unsigned numeric integer item. If the sending item contains any nonnumeric characters, the results are undefined.
(2c) When a receiving field is described as alphabetic and the sending data item contains any nonalphabetic characters, the results are undefined.

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

Data in the following chart represents the 'Legality of the MOVE/General Rule Reference'. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

Category of Sending Data Item	Category of Receiving Data Item		
	ALPHABETIC	ALPHANUMERIC EDITED, ALPHANUMERIC	NUMERIC INTEGER, NUMERIC NON-INTEGER, NUMERIC EDITED
ALPHABETIC	Yes/2a	Yes/2a	No/1a
ALPHANUMERIC	Yes/2c	Yes/2a	Yes/2b
ALPHANUMERIC EDITED	Yes/2c	Yes/2a	No/1a
NUMERIC INTEGER	No/1b	Yes/2a	Yes/2b
NUMERIC NON-INTEGER	No/1b	No/1c	Yes/2b
NUMERIC EDITED	No/1b	Yes/2a	No/1a

5.8.2 The EXAMINE Statement

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

General Format

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

The description of the identifier must be such that usage is DISPLAY (explicitly or implicitly).

Each literal must consist of a single character belonging to a class consistent with that of identifier; in addition, each literal may be any figurative constant, except ALL literal.

Examination proceeds as follows:

(1) For nonnumeric data items, examination starts at the left-most character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.
(2) If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the left-most character (excluding the sign) and proceeds to the right. Each character except the sign is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE statement.

The TALLYING option creates an integral count which replaces the value of a special register called TALLY (see Special Register, Section 1.4.6). The count represents the number of:

(1) Occurrences of literal-1 when the ALL option is used.
(2) Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.
(3) Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

When either of the REPLACING options is used the replacement rules are as follows, subject to the preceeding paragraph:

(1) When the ALL option is used, then literal-2 or literal-4 is substituted for each occurrence of literal-1 or literal-3.
(2) When the LEADING option is used, the substitution of literal-2 or literal-4 terminates as soon as a character other than literal-1 or literal-3 is encountered or the right-hand boundary of the data item is reached.
(3) When the UNTIL FIRST option is used, the substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 is encountered or the right-hand boundary of the data item is reached.
(4) When the FIRST option is used the first occurrence of literal-1 or literal-3 is replaced by literal-2 or literal-4.

5.9 Input-Output Statements

5.9.1 The OPEN Statement

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

General Format

```
OPEN [INPUT file-name [, file-name]...]
  [, OUTPUT file-name [, file-name]...]
  [, I-O file-name [, file-name]...]
```

Each of the choices (INPUT, OUTPUT, I-O) can be specified only once in a given OPEN statement. The I-O option pertains only to mass storage files.

The OPEN statement must be applied to all files, except sort files

(see The Sort Facility, Section 6.2). The OPEN statement for a file must be executed prior to the first READ, WRITE, START, or SEEK for that file. A second OPEN for a file cannot be executed prior to the execution of a CLOSE statement for that file. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record. If the external medium for the file permits rewinding, execution of the OPEN statement causes the file to be positioned at its beginning.

The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

When processing mass storage files for which the access mode is sequential, the OPEN statement supplies the initial address of the first record to be accessed.

5.9.2 The START Statement

The START statement is used to sequentially process an index-sequential file from a specified key.

General Format

START file-name; INVALID KEY imperative-statement

The value of the NOMINAL KEY data item must be set before the execution of the START statement. After a START statement has been executed, the next record of the file to be processed by a READ or WRITE statement will be the record whose RECORD KEY matches the NOMINAL KEY value. If no record was found, the imperative statement following the INVALID KEY phrase will be executed.

5.9.3 The SEEK Statement

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

General Format

SEEK file-name RECORD

A SEEK statement pertains only to fixed-format mass storage files in the random access mode and may be executed prior to the execution of each READ and WRITE statement.

Two SEEK statements for the same mass storage file may logically follow each other.

In Datapoint COBOL the SEEK statement has no effect on the execution of the object program.

5.9.4 The READ Statement

For sequential file processing, the READ statement makes available the next logical record from an input file and allows performance of a specified imperative statement when end of file is detected. For random file processing, the READ statement makes available a specific record from an imput file and allows performance of a specified imperative statement if the contents of the associated ACTUAL KEY or NOMINAL KEY data item are found to be invalid.

General Format

```
READ file-name RECORD [INTO identifier];
  {AT END | INVALID KEY} imperative-statement
```

An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file.

The AT END option is only used for non-mass storage files and for mass storage files in the sequential access mode. The INVALID KEY option is only used for mass storage files in the random access mode.

When a file consists of more than one type of logical record, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

If after reading the final logical record of a file in sequential access mode, another READ statement is initiated for that file, that final logical record is no longer available in its record area; the READ statement is then completed by the execution of the AT END phrase. After the AT END condition has been recognized for a file, a READ statement for that file must not be given without

prior execution of a CLOSE statement and an OPEN statement for that file.

For random file processing the READ statement implicitly performs the functions of the SEEK statement for a specific mass storage file, regardless of whether a SEEK statement is executed for the specified record of this file prior to the execution of the READ statement for that specified record. If such files are accessed for a specified mass storage record and the contents of the associated ACTUAL KEY or NOMINAL KEY data item are invalid, the INVALID KEY phrase is executed.

If the INTO option is specified, the current record is moved from the input area to the area specified by identifier according to the rule for the MOVE statement. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item. When the INTO option is used, the record being read is available in both the data area associated with identifier and the input record area.

The INTO option may only be used when the input file contains records of one type. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

5.9.5 The WRITE Statement

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of a printer. For random processing of mass storage files, the WRITE statement allows the performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

Format 1

```
WRITE record-name [FROM identifier-1]
  [{BEFORE | AFTER} ADVANCING
    {{identifier-2 | integer} [LINE | LINES]
    | mnemonic-name | PAGE}]
  [; AT {END-OF-PAGE | EOP}
    imperative-statement]
```

Format 2

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

EOP is an abbreviation for END-OF-PAGE.

An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

The logical record released by the execution of the WRITE statement is no longer available. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

If the FROM option is specified the data is moved from the area specified by identifier-1 to the output area, according to the rules specified for the MOVE statement. After execution of the WRITE statement is completed, the information in identifier-1 is available, even though that in record-name is not. Record-name and identifier-1 must not refer to the same storage area.

The ADVANCING option allows control of the vertical positioning of each record on the printed page. If the ADVANCING phrase is not used, the record will be written after the file has been advanced one line. If the ADVANCING option is used, the automatic single space is overridden.

(1) If identifier-2 is specified, the printer page is advanced the number of lines equal to the current value associated with identifier-2.
(2) If integer is specified, the printer page is advanced the number of lines equal to the value of integer.
(3) If mnemonic-name is specified, the printer page is advanced no spaces if the mnemonic-name is defined to be CSP, or to the top of the form if the mnemonic-name is defined to be C01.
(4) If the BEFORE option is used, the record is printed before the printer page is advanced according to the preceding rules.
(5) If the AFTER option is used, the record is printed after the printer page is advanced according to the preceding rules.

When identifier-2 or integer is used in the ADVANCING option, it must be the name of a numeric elementary item described without any positions to the right of the assumed decimal point. When the mnemonic-name option is used, the name is identified with a particular feature specified above. The mnemonic-name is defined

in the Special-Names paragraph of the Environment Division, Section 3.2.3.

If a LINAGE clause is associated with the file, the value of the associated LINAGE-COUNTER is updated to correspond with the new position on the page.

The END-OF-PAGE option allows special processing to be performed if a WRITE statement positions a file to its footing area. If after execution of a WRITE statement, the file is positioned in the footing area, the imperative-statement following the AT END-OF-PAGE phrase is executed. However if the WRITE statement advances the file past the footing area, the imperative-statement will not be executed.

The END-OF-PAGE option can only be used with files which have a LINAGE clause specified in their FD.

Format 2 is used for processing mass storage files. For mass storage files in the sequential access mode, the imperative statement in the INVALID KEY clause is executed when the end of the file area is reached and an attempt is made to execute a WRITE statement for that file.

For mass storage files in the random access mode, the WRITE statement implicitly performs a SEEK statement for a specific mass storage record, regardless of whether a SEEK statement is executed for this record prior to the execution of the WRITE statement. The imperative statement in the INVALID KEY phrase is executed when the contents of the ACTUAL KEY or NOMINAL KEY being used to obtain the mass storage record is found to be invalid. When the INVALID KEY condition exists, no writing takes place and the information in the record area is available.

5.9.6 The REWRITE Statement

The REWRITE statement is used to update a record of a mass storage file which has been previously READ.

General Format

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

An OPEN statement with the I-O option must be executed

prior to using the REWRITE statement. Further a READ statement for the same record as is specified by the ACTUAL KEY or NOMINAL KEY value, without intervening I/O to the file, must have been executed before the REWRITE statement is executed.

The REWRITE statement updates the current record as specified by the KEY value, or control is passed to the imperative statement following the INVALID KEY phrase.

5.9.7 The ACCEPT Statement

The ACCEPT statement causes low volume data to be transferred from the CONSOLE keyboard or from the system date.

Format 1

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

Format 2

```
ACCEPT identifier FROM DATE
```

In format 1, the ACCEPT statement causes the transfer of data from the CONSOLE keyboard. This data replaces the contents of the data item named by the identifier. The data being transferred will contain no more than 78 characters.

If the size of the data read from the keyboard is less than size of the receiving data item, then the data being transferred replaces the left-most characters of the receiving data item, and the remainder of the data item is filled with spaces. If the size of the data item is less than the size of the data read from the keyboard, then the left-most characters of the data read are transferred to the data item, and the remaining keyboard input is ignored.

In format 2, if the FROM DATE option is used, the system date is transferred to identifier. (See the DATE special register.)

5.9.8 The DISPLAY Statement

The DISPLAY statement causes low volume data to be transferred to the CONSOLE display.

General Format

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

The DISPLAY statement causes the contents of each operand to be transferred to the display in the order listed. At most 80 characters will be transferred.

Each literal may be any figurative constant except ALL. If a figurative constant is specified as one of the operands, only a single occurence of the figurative constant is displayed.

When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered. In the following rules, the sending data item is considered to be the composite of all operands in the DISPLAY statement.

(1) If the size of the data item being sent is greater than or equal to 80 characters, the first 80 characters of the data beginning with the left-most characters are sent to the display.
(2) If the size of the data item being sent is less than 80 characters, the transferred data is aligned to the left on the display.

5.9.9 The CLOSE Statement

The CLOSE statement terminates the processing of files with optional rewind and/or lock where applicable.

General Format

```
CLOSE file-name-1 [WITH {NO REWIND | LOCK}]
  [, file-name-2 [WITH {NO REWIND | LOCK}]]...
```

Each file-name is the name of a file upon which the CLOSE

statement is to operate; it must not be the name of a sort file. (See Section 6.2, The Sort Facility.) The WITH NO REWIND option applies only to files stored on tape devices.

In the discussion below, the term 'unit' applies to all input-output devices; the term 'reel' applies to tape devices. Treatment of mass storage devices in the sequential access mode is logically equivalent to the treament of a file on tape or analoguous medium.

(1) For the purposes of showing the effect of various CLOSE options as applied to various storage media, all input and output and input-output files are divided into the following categories:

   a. Non-reel. A file whose input or output medium is such that the concepts of rewinding and reels have no meaning.
   b. Sequential (single) reel/unit. A sequential file that is entirely contained in one unit.
   c. Random (single) reel/unit. A file in the random access mode that is entirely contained on one mass storage unit.

(2) The results of executing each CLOSE option for each type of file are summarized in Figure 5-1. The definitions of the symbols in the Figure are given below. Where the definitions depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

   A. No Rewind. The current reel is left in its current position.

   B. Standard Close File.

      Input Files and Input-Output Files: If the file is positioned at its end and a label is specified for the file, the label is processed according to the standard convention. The behavior of the CLOSE statement when a label record is not specified but is present, or when a label record is specified but is not present, is undefined. If the file is positioned at its end and label records are not specified for the file, label processing does not take place. If the file is positioned other than at its end, there is no ending label processing. An input file, or an input-output file, is considered to be at the end of the file if the

imperative statement in the AT END phrase of a READ statement has been executed and no CLOSE statement has been executed.

Output Files: If a label record is specified for the file, standard label processing occurs. The behavior of the CLOSE statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If label records are not specified for the file, label processing does not take place.

C. Standard File Lock. When this option is used, the file cannot be processed again during the execution of this object program.

D. Rewind. The file is positioned at its beginning. In the case of tape, this means that the reel is rewound.

X. Illegal. This is an illegal combination of a CLOSE option and a file type.

If a file has been opened and is not closed prior to the execution of a STOP RUN statement, the contents of the file will be indeterminate.

If a CLOSE statement has been executed for a file, a READ, WRITE, START, or SEEK statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

	FILE TYPE		
CLOSE OPTION	NON-REEL	SEQUENTIAL SINGLE-REEL/UNIT	RANDOM SINGLE-REEL/UNIT
CLOSE	B	B, D	B
CLOSE WITH LOCK	B, C	B, C, D	B, C
CLOSE WITH NO REWIND	X	B, A	X

Figure 5-1. Relationship of Types of Files and the Options of the CLOSE Statement.

5.10 Table Manipulating Statements

The only table-manipulating statement is the SET statement. It is discussed in Section 6.1.2.3.

5.11 Program Linkage Statement

The CALL and EXIT PROGRAM statements are discussed in Section 6.4, The Sub-Program Facility.

5.12 Compiler Directing Statements

The compiler directing statements are the COPY, ENTER, and NOTE statements. The COPY statement is discussed in Section 6.3.1

5.12.1 The ENTER Statement

The ENTER statement is provided to allow a means of using more than one language in the same program. The Datapoint COBOL compiler makes no provision for translating other languages, although any compatible relocatable object program may be called using the CALL statement. (See The CALL Statement, Section 6.4.1.)

General Format

ENTER language-name [routine-name].

Since the Datapoint COBOL compiler does not allow mixed source languages, this statement is treated as a comment.

5.12.2 The NOTE Sentence

The NOTE sentence allows the programmer to write commentary which is produced on the listing, but not compiled.

General Format

NOTE character-string.

Any combination of the characters from the ASCII character set may be included in the character-string. If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be part of the character-string. Proper format rules for paragraph structure must be observed in this case. If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space. (See Chapter 1, Overall Language Considerations, Section 1.6.1, for alternate methods of introducing commentary.)

CHAPTER 6. SPECIAL FEATURES

6.1 Table Handling Facility

Datapoint COBOL provides a capability for defining tables of contiguous data items and accessing an item relative to its position in the table. Language facility is provided for specifying how many times an item is to be repeated. Each item may be identified through use of a subscript or an index-name (see Uniqueness of Data Reference, Section 1.5.7). The table handling capability provides for accessing items in up to three-dimensional fixed-length tables.

6.1.1 Data Division Considerstions

6.1.1.1 The OCCURS Clause

The OCCURS clause eliminates the need for separate entries for repeated data and supplies information required for the application of subscripts or indices.

General Format

```
OCCURS integer TIMES
  [INDEXED BY index-name-1 [, index-name-2]...]
```

The OCCURS clause is optional in a data description entry and cannot be specified in a data description entry that has an 01 or 77 level number.

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

The data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described. (See Data Description, Section 4.4)

Integer must be a positive integer. The value of integer represents the exact number of occurrences.

An INDEXED BY phrase is required if the subject of this entry, or an item within it if it is a group item, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere since its allocation and format are dependent on the hardware, and not being data, cannot be associated with any data hierarchy. Three levels of indexing are permitted.

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

6.1.1.2 The USAGE Clause

The USAGE clause specifies the format of a data item in the computer storage.

General Format

[USAGE IS] INDEX

The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table-element. The elementary item cannot be a conditional variable.

If a group item is described with the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in a SET statement or in a relation condition.

An index data item can be referred to directly only in a SET statement or in a relation condition. An index data item can be

part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

6.1.2 Procedure Division Considerations

6.1.2.1 Relation Condition

The result of the comparison of two index-names is the same as if the corresponding occurrence numbers are compared. In the comparison of an index-name and a data item (other than an index data item) or literal, the occurence number that corresponds to the value of the index-name is compared to the data item or literal. In the comparison of an index data item and an index-name or another index data item, the actual values are compared without conversion. The result of the comparison of an index data item with any data item not specified above is unpredictable.

6.1.2.2 Overlapping Operands

When a sending and a receiving item in a SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

6.1.2.3 The SET Statement

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

Format 1

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

Format 2

```
SET index-name-4 [, index-name-5]...
  {UP BY | DOWN BY} {identifier-4 | literal-2}
```

All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

All identifiers must name either index data items, or elementary items described as an integer, except that identifier-4 must not name an index data item. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause. In Format 1, the following action occurs:

(1) Index-name-1 is set to a value that corresponds to the same occurrence number to which either index-name-3, identifier-3, or literal-1 corresponds. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.
(2) If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index data item. Literal-1 cannot be used in this case.
(3) If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor literal-1 can be used in this case.
(4) The process is repeated for index-name-2, identfier-2, etc., if specified. Each time the value of index-name-3 or identifier-3 is used, it is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4; thereafter, the process is repeated for

index-name-5, etc. Each time the value of identifier-4 is used, it is used as it was at the beginning of the execution of the statement.

6.2 The Sort Facility

The Sort facility provides the capability to order a file of records according to a set of user-specified keys within each record. Optionally, a user may apply some special processing which may consist of addition, deletion, creation, altering, editing or other modifications of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the sort.

6.2.1 Environment Division Considerations

6.2.1.1 The FILE-CONTROL Paragraph

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows particular hardware assignments.

General Format

```
FILE-CONTROL. {SELECT file-name
  ASSIGN TO [integer-1] device-name-1
  [, device-name-2]... .}...
```

The FILE-CONTROL paragraph is specified in Section 3.3.1 of Chapter 3, The Environment Division.

Each sort-file must be the subject of an ASSIGN clause.

6.2.1.2 The I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies the memory area to be shared by different files.

General Format

```
I-O-CONTROL.
[; SAME {RECORD | SORT} AREA
  FOR file-name-1 {, file-name-2}...]...
```

This paragraph is optional.

A file-name that represents a sort-file must not appear in the SAME clause unless the RECORD or SORT option is used.
The two forms of the SAME clause (SAME RECORD AREA, SAME SORT AREA) are considered separately in the following:

More than one SAME clause may be included in a program, however;

(1) A file-name must not appear in more than one SAME RECORD AREA clause.
(2) A file-name that represents a sort-file must not appear in more than one SAME SORT AREA clause.
(3) If a file-name that does not represent a sort-file appears in a SAME AREA clause and one or more SAME SORT AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA clause(s). (See Chapter 3, The Environment Division, Section 3.4.2)

The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time; however, the logical record of only one of the files can exist in the record area at one time.

If the SAME SORT AREA clause is used, at least one of the file-names must represent a sort-file. Files that do not represent sort-files may also be named in the clause. This clause specifies that storage is shared as follows:

(1) The SAME SORT AREA clause specifies a memory area which will be made available for use in sorting each sort-file named. Thus any memory area allocated for the sorting of a sort-file is available for re-use in sorting any of the other sort-files.
(2) In addition, storage areas assigned to files that do not

represent sort-files may be allocated as needed for sorting the sort files named in the SAME SORT AREA clause.

(3) Files other than sort-files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

(4) During tne execution of a SORT statement that refers to a sort-file named in this clause, any non-sort-files named in this clause must not be open.

6.2.2 Data Division Considerations

An SD File Description in the File Section gives information about the name and size of data records in the sort-file. The name Sort File designates a set of records to be sorted by a SORT statement. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT statement.

6.2.2.1 The Sort File Description - Complete Entry Skeleton

The Sort File Description furnishes information concerning the physical structure, identification, and record names of the file to be sorted.

General Format

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

The level indicator SD identifies the beginning of the Sort File Description and must precede the file-name.

All semicolons are optional in the Sort File Description but the entry must be terminated by a period.

The order of appearance of the clauses which follow the name of the file is immaterial. (See Section 4.1, for an explanation of the clauses.)

6.2.3 Procedure Division Considerations

6.2.3.1 The SORT Statement

The SORT statement creates a sort-file by executing input procedures or by transferring records from another file, sorts the records in the sort-file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort-file, in sorted order, to some output procedures or to an output file. There is no limit to the number of SORT statements allowed in a program, however the number of File Description entries (SD and FD) may not exceed thirty-two (32).

General Format

```
SORT file-name-1 ON {DESCENDING | ASCENDING}
    KEY data-name-1 [, data-name-2]...
  [; ON {DESCENDING | ASCENDING}
    KEY data-name-3 [, data-name-4]...]...
  {INPUT PROCEDURE IS
    section-name-1 [THRU section-name-2]
  | USING file-name-2}
  {OUTPUT PROCEDURE IS
    section-name-3 [THRU section-name-4]
  | GIVING file-name-3}
```

The Procedure Division may contain more than one SORT statement appearing anywhere except in the input and output procedures associated with a SORT statement.

Section-name-1 represents the name of an input procedure.
Section-name-2 represents the name of an output procedure.

Sorted File Sequence

The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY clauses. The collating sequence is ASCII. The data-names may be qualified.

The direction of the sort depends on the use of the ASCENDING or DESCENDING clauses as follows:

(1) When an ASCENDING clause is used, the sorted sequence is from

lowest value of key to highest value according to the rules for comparison of operands in a Relation Condition.

(2) When a DESCENDING clause is used, the sorted sequence is from highest value of key to lowest value according to the rules for comparison of operands in a Relation Condition.

Files Associated with a SORT

File-name-1 must be described in a Sort File Description entry in the Data Division. Each data-name must represent data items described in records associated with file-name-1. File-name-2 and file-name-3 must be described in a File Description entry, not in a Sort File Description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2 and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause equal amounts of computer storage to be allocated for the corresponding records. The file name entry may be assigned to any hardware device.

The record description for every record that is a logical record associated with the sort-file description must contain the KEY items data-name-1, data-name-2, etc. These KEY items are subject to the following rules:

(1) Where more than one record description appears, the key items need only be described in one of the record descriptions. When the key items are described in more than one record the data descriptions must be equivalent and their starting position must always be the same number of character positions from the beginning of each record.

(2) They may not contain nor be subordinate to entries that contain an OCCURS clause.

Input Procedure

The input procedure, if present, must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file. Control must not be passed to the input procedure except when a related SORT statement is being executed, because the RELEASE statements in the input procedure have no meaning unless they are controlled by a SORT statement. The input

procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

(1) The input procedure must not contain any SORT statements.
(2) The input procedure must not contains any explicit transfers of control to points outside the input procedures; ALTER, GO TO and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure.

If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

USING Option

If the USING option is specified, all the records in the file-name-2 are transferred automatically to the file-name-1. At the time of execution of the SORT statement, file-name-2 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of, file-name-2. The terminating function is performed as if the CLOSE statement had been explicitly written without optional phrases. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 to the file area of file-name-1 and the release of the records to the initial phase of the sort operation.

Output Procedure

The output procedure, if present, must consist of one or more sections that appear contiguously in a source program and do not form a part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted records available for processing. Control must not be passed to the output procedure except when a related SORT statement is being executed, because the RETURN statements in the output procedure have no meaning unless they are controlled by a SORT statement. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time in sorted order, from the sort-file. The restrictions on the procedural statements within the output procedure are as follows:

(1) The output procedure must not contain any SORT statements.
(2) The output procedure must not contain any explicit transfers of control to points outside the output procedure; ALTER, GO TO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure.

If an output procedure is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

GIVING Option

If the GIVING option is used, all the sorted records in file-name-1 are automatically transferred to file-name-3 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-3 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of, file-name-3. The terminating function is performed as if the CLOSE statement had been explicitly written without optional phrases. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of the records from the file area for file-name-1 to the file area for file-name-3.

Segmentation as defined in this chapter can be applied to the sections within the input or output procedures.

6.2.3.2 The RELEASE Statement

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

General Format

RELEASE record-name [FROM identifier]

The RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation. A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose sort-file description contains record-name. (See The SORT Statement, preceeding.)

After the RELEASE is executed, the logical record is no longer available. When control passes from the input procedure, the file consists of all those records that were placed in it by the execution of RELEASE statements.

The record-name must be the name of a logical record in the associated sort-file description and may be qualified.

If the FROM option is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort-file. Moving takes place according to the rules specified for the MOVE statement. The information in the record area is no longer available, but the information in the data area associated with identifier is available. Record-name and identifier must not refer to the same storage area.

6.2.3.3 The RETURN Statement

The RETURN statement obtains sorted records from the final phase of a sort operation.

General Format

```
RETURN file-name RECORD [INTO identifier]
  ; AT END imperative-statement
```

The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT statement, to be made available for processing in the records area associated with the sort file. A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name.

After execution of the imperative-statement in the AT END phrase, no RETURN statements may be executed within the current output procedure.

File-name must be described by a Sort File Description entry in

the Data Division. When a file consists of more than one type of logical record, these records automatically share the storage area. This is equivalent to saying that there exists an implicit redefinition of the area, and only the information that is present in the current record is accessible.

The INTO option may only be used when the input file contains just one type of record. The storage area associated with identifier and the storage area associated with file-name must not be the same storage area. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rule for the MOVE statement. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

6.3 The COBOL Source Library Feature

The COBOL source library feature provides a capability for specifying text that is to be copied from a library into the source program, with word substitution as text is copied. The effect of the compilation of library text is the same as if the text were written as part of the source program.

A COBOL source library contains text that is available to a source program at compile time. It may contain text for the Environment Division, Data Division and the Procedure Division available through the use of the COPY statement.

COBOL library text is placed on the COBOL library through the use of the Library Pre-Processor (LIB version 2.1 or later).

6.3.1 The COPY Statement

General Format

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

A word in this format may represent one of the following and must conform to the definition of words (See Character

Strings, Section 1.4.):

data-name
procedure-name
condition-name
mnemonic-name
file-name

The COPY statement may appear as follows:

(1) in any of the paragraphs in the Environment Division, or in any entry of the File Control paragraph,
(2) in any level indicator entries or an 01 or 77 level number entry in the Data Division,
(3) in a section or paragraph in the Procedure Division.

No other statement or clause may appear after the COPY statement in the same entry.

The library text is copied from the library and the result of the compilation is the same as if the text (with substitutions) were actually part of the source program. The copying process is terminated by the end of the library text. The text contained on the library must not contain any COPY statements.

If the REPLACING phrase is used, each occurrence of word-1, word-3, etc., in the text being copied from the library is replaced by the word, identifier, or literal associated with it in the REPLACING phrase. Use of the REPLACING option does not alter the material as it appears in the library.

The source material copied from the library will appear on the listing after the COPY statement, with the substitutions called for in the COPY statement.

6.3.2 Valid Locations for the COPY Statement

The COPY statement is written in any of the following forms:

(1) In the Environment Division:

```
SOURCE-COMPUTER. copy-statement.
OBJECT-COMPUTER. copy-statement.
SPECIAL-NAMES. copy-statement.
FILE-CONTROL. copy-statement.
SELECT file-name copy-statement.
I-O-CONTROL. copy-statement.
```

(2) In the File Section:

```
FD file-name copy-statement.
SD file-name copy-statement.
01 data-name copy-statement.
```

(3) In the Working-Storage or Linkage Sections:

```
77 data-name copy-statement.
01 data-name copy-statement.
```

(4) In the Procedure Division:

```
section-name SECTION [priority-number].
    copy-statement.
paragraph-name. copy-statement.
```

6.4 The Sub-Program Feature

The Datapoint COBOL sub-program feature allows a COBOL program to invoke both machine-language sub-programs and previously compiled COBOL sub-programs.

6.4.1 The CALL Statement

The CALL statement is used to invoke a previously compiled or assembled program.

General Format

```
CALL literal [USING identifier-1
    [, identifier-2]...]
```

The name of the called program is written as a nonnumeric literal. (Only the first eight characters are significant.) The

identifiers in the USING list are the parameters passed on the call. The parameters in the called program must correspond to this list, otherwise the result of the CALL is undefined.

Control is passed to the sub-program on the CALL statement. If the sub-program is a COBOL program, then control is returned when the sub-program executes an EXIT PROGRAM statement. The called sub-program may be an assembler language program which is in the relocatable format produced by SNAP/2.

Since sub-programs may not be segmented, the values of working storage, PERFORM statements, ALTERed GOTO's, etc, remain unchanged when the sub-program is subsequently reentered.

6.4.2 The USING Option

The USING option allows COBOL programs to pass information on a CALL. The USING option is used in a CALL statement to pass information to the called program, and in the Procedure Division header of sub-programs to receive information being passed.

Format 1

```
CALL literal [USING identifier-1
    [, identifier-2]...]
```

Format 2

```
PROCEDURE DIVISION [USING identifier-1
    [, identifier-2]...]
```

Format 1 is used to pass information to the called program.

Format 2 is used in COBOL sub-programs to receive information from a calling program. In this case execution of the sub-program always starts with the first statement in the Procedure Division. Any identifier mentioned in the Division header must be defined in the Linkage Section.

6.4.3 The EXIT PROGRAM Statement

The EXIT PROGRAM statement is used to return control from a COBOL sub-program to its calling program.

General Format

paragraph-name. EXIT PROGRAM.

The EXIT statement must be the only statement in the paragraph.

When a COBOL program has been called, the EXIT PROGRAM statement effects the return of control to the calling program. Otherwise control passes through the EXIT statement to the next paragraph in the program.

6.5 The Segmentation Feature

The segmentation feature provides a capability for specifying object program overlay requirements. COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division is considered in determining segmentation requirements for an object program.

Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to insure uniqueness.

Permanent Segments

A permanent segment is a segment in the fixed portion of the object program which cannot be overlaid by any other part of the program. The fixed portion is defined as that part of the object program which is always in memory.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, another independent segment. An independent segment is effectively in its initial state each time the segment is made available to the program.

6.5.1 Segment Classification

Sections which are to be segmented are classified, using a system of priority-numbers and the following criteria:

(1) Logic Requirements - Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments, sections which are used less frequently are normally classified as independent segments.

(2) Frequency of Use - Generally, the more frequently a section is referred to, the lower its priority-number; the less frequently it is referred to, the higher its priority-number.

(3) Relationship to Other Sections - Sections which frequently communicate with one another should be given the same priority-numbers.

6.5.2 Structure of Program Segments

Section classification is accomplished by means of a system of priority-numbers. The priority-number is included in the section header.

General Format

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

The priority-number must be an integer ranging in value from 0 through 99. Segments with priority-number 0 through 49 must be together in the source program and belong to the fixed portion of the object program. Segments with priority-number 50 through 99 are independent segments.

6.5.3 Restrictions on Program Flow

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

The ALTER Statement

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

The PERFORM Statement

A PERFORM statement that appears in a section whose priority number is less than 50, can have within its range only the following:

(1) Sections each of which has a priority-number less than 50, or
(2) Sections wholly contained in a single segment whose priority-number is greater than 49.

A PERFORM statement that appears in a section whose priority-number is equal to or greater than 50, can have within its range only the following:

(1) Sections each of which has the same priority-number as that containing the PERFORM statement, or
(2) Sections with a priority-number that is less than 50.

When a procedure-name in a segment with a priority-number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority-number, the segment referred to is made available in its initial state for each execution of the PERFORM statement.

Called Programs

COBOL language sub-programs may not use the segmentation feature. COBOL programs which use sub-programs, but are not themselves sub-programs, may be segmented.

6.6 DOS Command-line Interface Feature

The DOS Command-line interface feature gives the COBOL programmer access to the DOS command line (MCR$).

6.6.1 The COMMAND-LINE Special Variable

Pre-defined in every COBOL program is a variable named COBOL-LINE which has the characteristics of a 79 character alphanumeric string i.e., PICTURE X(79). This variable does not occupy storage but simply defines the interface to the programmer.

COMMAND-LINE can be used in any place that a regular alphanumeric variable can be used e.g., MOVE, DISPLAY, or ACCEPT, statements. When the COBOL program starts, COMMAND-LINE contains the information entered by the user to the DOS command interpretor. Information moved into COMMAND-LINE will be there after the program completes, unless an abort occurs during execution.

6.7 SPECIAL I/O Feature

The SPECIAL I/O feature allows the COBOL programmer to interface to unsupported I/O devices using the standard I/O statements.

6.7.1 Environment Division Considerations

For files that are maintained on unsupported I/O devices, the device specification in the ENVIRONMENT DIVISION has the form

```
SELECT <filename> ASSIGN TO
         SPECIAL -<rtnname>[-<integer>]
```

<rtnname> is the name of a SNAP2 program that will perform the operations; <integer> is the number of bytes to be reserved in the File Descriptor Block (FDB) for the file. If <integer> is not specified, no space will be reserved.

6.7.2 Procedure Division Considerations

OPEN, CLOSE, READ, and WRITE statements may be used with SPECIAL files.

6.7.3 I/O Subroutine

Associated with each SPECIAL file is a SNAP2 program to perform the actual operations; more than one file can be associated with a single program. This program is called once for every OPEN, CLOSE, READ, or WRITE operation. The address of the File Descriptor Block is passed in (HL) and a code specifying what operation is to be performed is passed in (A). The file IFDBDEF/TXT, which defines the format of a File Descriptor Block, contains the definitions or these codes.

6.7.4 Reserve Area

In each File Descriptor Block associated with a SPECIAL file is a block of memory called the Reserve Area. The size of this area is specified in the SELECT clause for the file. Initially, the area contains binary zeros; the area is supplied solely for the use of the I/O sub-routine and its contents are not interogated or modified by the COBOL runtime system.

APPENDIX A. COMPOSITE LANGUAGE SKELETON

This appendix is intended to display complete and syntactically correct formats for Datapoint COBOL. It does not imply legal combinations of language elements.

Four margins are used in the presentation of the skeleton: the leftmost, margin 1, is the name of the paragraph, section, or verb for which syntax is provided; the second margin is for distinguishing different formats; the third margin is the start of the syntax; the fourth margin is for the continuation of the Identification division

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

Environment division

```
ENVIRONMENT DIVISION.
```

Configuration section

```
CONFIGURATION SECTION.
```

Source-computer

Format 1:

```
SOURCE-COMPUTER. COPY-statement.
```

Format 2:

```
SOURCE-COMPUTER. computer-name.
```

Object-computer

```
    Format 1:
        OBJECT-COMPUTER. COPY-statement.

    Format 2:
        OBJECT-COMPUTER. computer-name
            [, MEMORY SIZE integer CHARACTERS]
            [, USING DEBUGGER].
```

Special-names

```
    Format 1:
        SPECIAL-NAMES. COPY-statement.

    Format 2:
        SPECIAL-NAMES. [system-name
            {IS mnemonic-name
              [, ON STATUS IS condition-name-1
                [, OFF STATUS IS condition-name-2]]
            | IS mnemonic-name
              [, OFF STATUS IS condition-name-2
                [, ON STATUS IS condition-name-1]]
            | ON STATUS IS condition-name-1
              [, OFF STATUS IS condition-name-2]
            | OFF STATUS IS condition-name-2
              [, ON STATUS IS condition-name-1}]...
            [, QUOTE IS APOSTROPHE]
            [, CURRENCY SIGN IS literal]
            [, DECIMAL-POINT IS COMMA].
```

Input-Output section

```
        INPUT-OUTPUT SECTION.
```

File-control

```
    Format 1:
        FILE-CONTROL. COPY-statement.
```

```
    Format 2:
        FILE-CONTROL. {SELECT file-name
            { COPY-statement
            | ASSIGN TO [integer-1] device-name-1
              [, device-name-2]...
              [FOR MULTIPLE {REEL | UNIT}]
            [, RESERVE {integer-2 | NO}
              ALTERNATE [AREA | AREAS]]
            [, {FILE-LIMIT IS | FILE-LIMITS ARE}
              {data-name-1 | literal-1} THRU
              {data-name-2 | literal-2}]
            [, ACCESS MODE IS {SEQUENTIAL | RANDOM}]
            [, PROCESSING MODE IS SEQUENTIAL]
            [, ACTUAL KEY IS data-name-3
            | , NOMINAL KEY IS data-name-3]
            [, RECORD KEY IS data-name-4].}...
```

I-O-Control

```
    Format 1:
        I-O-CONTROL. COPY-statement.

    Format 2:
        I-O-CONTROL. [; RERUN ON device-name
              EVERY integer RECORDS OF file-name-1]...
            [; SAME [SORT | RECORD] AREA
              FOR file-name-2 {, file-name-3}...]...
            [; APPLY CORE-INDEX TO data-name ON file-name-4
              {, file-name-5}...]... .
```

Data division

```
        DATA DIVISION.
```

File section

```
        FILE SECTION.
```

File description

```
    Format 1:
        FD file-name; COPY-statement.
```

```
    Format 2:
        FD file-name
            [; BLOCK CONTAINS integer-1
              {RECORDS | CHARACTERS}]
            [; DATA {RECORD IS | RECORDS ARE}
              data-name-1 [, data-name-2]...]
            ; LABEL {RECORD IS | RECORDS ARE}
              {STANDARD | OMITTED}
            [; LINAGE IS integer-2 LINES
              [WITH FOOTING AT integer-3]]
            [; RECORD CONTAINS [integer-4 TO]
              integer-5 CHARACTERS]
            [; VALUE OF data-name-3 IS literal-1
              [, data-name-4 IS literal-2]...].
```

Sort description

```
    Format 1:
        SD file-name; COPY-statement.

    Format 2:
        SD file-name;
            [; DATA {RECORD IS | RECORDS ARE}
              data-name-1 [, data-name-2]...]
            [; RECORD CONTAINS [integer-1 TO]
              integer-2 CHARACTERS].
```

Record description

```
    Format 1:
        01 data-name-1; COPY-statement.

    Format 2:
        level-number {data-name-1 | FILLER}
            [; REDEFINES data-name-2]
            [; BLANK WHEN ZERO]
            [; {JUSTIFIED | JUST} RIGHT]
            [; OCCURS integer TIMES]
            [; INDEXED BY index-name-1 [, index-name-2]...]
            [; {PICTURE | PIC} IS character-string]
            [; {SYNCHRONIZED | SYNC} [LEFT | RIGHT]]
            [; [USAGE IS]
              {COMPUTATIONAL | COMP | DISPLAY | INDEX}]
            [; VALUE IS literal].

    Format 3:
```

```
        88 condition-name
            ; VALUE IS literal.

    Format 4:
        77 data-name COPY-statement
```

Working storage section

```
        WORKING-STORAGE SECTION.
```

Linkage section

```
        LINKAGE SECTION.
```

Procedure division

```
        PROCEDURE DIVISION
            [USING identifer-1 [, identifier-2]...].
```

Accept

```
        ACCEPT identifier [FROM mnemonic-name
            | FROM DATE]
```

Add

```
    Format 1:
        ADD {identifier-1 | literal-1}
            [, identifier-2 | , literal-2]...
            TO identifier-m [ROUNDED]
            [; ON SIZE ERROR imperative-statement]

    Format 2:
        ADD {identifier-1 | literal-1} ,
            {identifier-2 | literal-2}
            [, identifier-3 | , literal-3]...
            GIVING identifier-m [ROUNDED]
            [; ON SIZE ERROR imperative-statement]
```

Alter

```
ALTER procedure-name-1 TO
    [PROCEED TO] procedure-name-2
    [, procedure-name-3 TO
      [PROCEED TO] procedure-name-4]...
```

Call

```
CALL literal [USING identifier-1
    [, identifier-2]...]
```

Close

```
CLOSE file-name-1 [WITH {NO REWIND | LOCK}]
    [, file-name-2 [WITH {NO REWIND | LOCK}]]...
```

Compute

```
COMPUTE identifier-1 [ROUNDED] =
    {identifier-2 | literal | arithmetic-expression}
    [; ON SIZE ERROR imperative-statement]
```

Copy

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

Display

```
DISPLAY {literal-1 | identifier-1}
    [, literal-2 | , identifier-2]...
    [UPON mnemonic-name]
```

Divide

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

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

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

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

    Format 5:
        DIVIDE {identifier-1 | literal-1}
            BY {identifier-2 | literal-2}
            GIVING identifier-3 [ROUNDED]
            REMAINDER identifier-4
            [; ON SIZE ERROR imperative-statement]
```

Enter

```
        ENTER language-name [routine-name].
```

Examine

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

Exit

```
        EXIT [PROGRAM].
```

Go

```
    Format 1:
        GO TO [procedure-name-1]

    Format 2:
        GO TO procedure-name-1 [, procedure-name-2]...
            , procedure-name-n DEPENDING ON identifier
```

If

```
        IF condition; {statement-1 | NEXT SENTENCE};
            ELSE {statement-2 | NEXT SENTENCE}
```

Move

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

Multiply

```
    Format 1:
        MULTIPLY {identifier-1 | literal-1}
            BY identifier-2 [ROUNDED]
            [; ON SIZE ERROR imperative-statement]

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

Note

```
        NOTE character-string.
```

Open

```
OPEN [INPUT file-name-1 [, file-name-2]...]
    [, OUTPUT file-name-3 [, file-name-4]...
    [, I-O file-name-5 [, file-name-6]...
```

Paragraph names

```
{paragraph-name. {sentence}...}...
```

Perform

Format 1:

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

Format 2:

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

Format 3:

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

Format 4:

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

Read

```
READ file-name RECORD [INTO identifier];
    {AT END | INVALID KEY} imperative-statement
```

Release

```
RELEASE record-name [FROM identifier]
```

Return

```
RETURN file-name RECORD [INTO identifier]
    ; AT END imperative-statement
```

Rewrite

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

Section names

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

Seek

```
SEEK file-name RECORD
```

Set

Format 1:

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

Format 2:

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

Start

```
START file-name; INVALID KEY imperative-statement
```

Sort

```
SORT file-name-1 ON {DESCENDING | ASCENDING}
      KEY data-name-1 [, data-name-2]...
    [; ON {DESCENDING | ASCENDING}
      KEY data-name-3 [, data-name-4]...]...
    {INPUT PROCEDURE IS
      section-name-1 [THRU section-name-2]
    | USING file-name-2}
    {OUTPUT PROCEDURE IS
      section-name-3 [THRU section-name-4]
    | GIVING file-name-3}
```

Stop

```
STOP {RUN | literal}
```

Subtract

Format 1:

```
SUBTRACT {literal-1 | identifier-1}
    [, literal-2 | , identifier-2]...
    FROM identifier-m [ROUNDED]
    [; ON SIZE ERROR imperative-statement]
```

Format 2:

```
SUBTRACT {literal-1 | identifier-1}
    [, literal-2 | , identifier-2]...
    FROM {literal-m | identifier-m}
    GIVING identifier-n [ROUNDED]
    [; ON SIZE ERROR imperative-statement]
```

Write

Format 1:

```
WRITE record-name [FROM identifier-1]
    [{BEFORE | AFTER} ADVANCING
      {{identifier | integer} [LINE | LINES]
      | mnemonic-name | PAGE}]
    [; AT {END-OF-PAGE | EOP}
      imperative-statement]
```

Format 2:

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

APPENDIX B. ERROR MESSAGES

Note: F preceding message indicates fatal error. A value is inserted by the compiler between the slashes in the message below (example: /name/).

1		Invalid blank preceding delimiter
2		Invalid blank preceding right parenthesis
3		Invalid character preceding left parenthesis
4		Invalid character following left parenthesis
5		Invalid character following non-numeric literal
6		Invalid character following right parenthesis
7		Item exceeds maximum length
8		First character on continuation card in area A
9	F	Unexpected end of source file found
10		Invalid character in continuation field
11		No leading quote for non-numeric literal
12		No closing quote for non-numeric literal
13		First character on non-continuation card following continued non-numeric literal is a quote, but card is not marked as continued
14		Invalid blank card(s) or comment cards in continuation
15		Invalid continuation card
16		Invalid figurative constant after ALL
17		Invalid COPY statement
18		Invalid library name in COPY statement
19	F	COPY library member not found
20		Invalid string in replacing option
21	F	Library not open for COPY
22		End of COPY library found before end of member
23		Invalid replacing string item /string/
24	F	Maximum replacement string length exceeded
30	F	Missing or misplaced IDENTIFICATION header
31	F	Invalid IDENTIFICATION DIVISION header
32		Paragraph name /name/ not in area A
33		Missing or misplaced PROGRAM-ID paragraph
34		Invalid program-name
35		Paragraph name /name/ is either out of sequence or a duplicate
36		Period missing following paragraph name or program-name
37		Invalid item /insert/
40		Syntax error
41		Invalid item /insert/
42		Missing period

```
43      Expected item in area A not found
44      Invalid or missing name clause in SELECT statement
45      Invalid or missing mnemonic name
46      Invalid condition status clause
47      Both Actual Key and Nominal Key were specified
48      Invalid device name
49      Invalid file-type in device name
50      Invalid filename in device name
51      Invalid cassette-type in device name
52      Invalid tape specification in device name
53      Invalid filename-flag in device name
54      Invalid memory size clause
55  F   Invalid Environment Division header
56      Invalid Section header
57      Invalid Source-Computer paragraph
58      Invalid Object-Computer paragraph
59      Invalid currency sign clause
60      Invalid name clause
61      Missing item in name clause
62      Invalid or missing special name
63      Missing Status clause for Condition clause
64      Invalid or missing filename
65      Invalid device name in RERUN clause
66      Invalid program name for SPECIAL device in SELECT clause
80      Syntax error
81      Invalid item /insert/
82  F   Invalid DATA DIVISION header
83      Invalid File name /insert/
84      Invalid BLOCK size
85      Invalid DATA RECORD clause
86      Invalid LABEL RECORD clause
87      Invalid LINAGE /insert/
88      Invalid FOOTING /insert/
89      Invalid RECORD CONTAINS clause
90      Invalid VALUE OF clause
91      Invalid LABEL VALUE clause
92      Invalid BLANK WHEN ZERO clause
93      Invalid VALUE item /insert/
94      Invalid OCCURS item /insert/
95      Invalid SYNCHRONIZED clause
96      Invalid USAGE
97      Duplicate name clause in record entry
98      Invalid 77 level item
99      Invalid or misplaced level number
100     Duplicate clause in record, first one used
101     Invalid item /insert/ in name clause
102     Expected item in area A not found
103     Missing period
```

104 Missing LABEL clause in FD
105 Missing name SECTION header
106 Unexpected item /insert/ in area A
110 Syntax error
111 Invalid item /insert/
112 Syntax error in verb statement
113 Invalid item /insert/ in verb statement
114 Invalid operand list in verb statement
115 Invalid receiving field in verb statement
116 Invalid procedure name /name/ in verb statement
117 Missing /item/ in verb statement
118 Invalid expression in verb statement
119 Duplicate option in OPEN statement
120 Required INVALID KEY clause missing in verb statement
121 Option missing in verb statement
122 Missing period
123 Missing first section header
124 Missing first paragraph this section
140 Conflict of usage from /name/ to /name/
141 Unmatched level number for /name/
142 Too many subscripts for /name/
143 OCCURS clause specified for level 01 or 77 named /name/
144 Group item named FILLER
145 Multiple items at the same level in a group named /name/
146 No related data item for condition name /name/
147 Element /name/ of array has initial value
160 Picture specified for group item /name/
161 USAGE INDEX specified for group item /name/
162 RIGHT-JUSTIFIED specified for group item /name/
163 BLANK WHEN ZERO specified for group item /name/
164 BLANK WHEN ZERO specified for non-DISPLAY item /name/
165 Neither PICTURE nor USAGE INDEX specified for elementary item /name/
166 Both PICTURE and USAGE INDEX specified for elementary item /name/
167 USAGE other than DISPLAY for non-numeric item /name/
168 USAGE COMPUTATIONAL specified for DATABUS Numeric elementary item /name/
169 Invalid PICTURE specification for elementary item /name/
170 Invalid length for elementary item /name/
171 Invalid scale for elementary item /name/
172 Invalid character in picture for item /name/
173 USAGE EBCDIC specified for non-numeric item /name/
180 Redefinition or record /name/ is larger than redefined item or work area for file /name/
181 Record /name/ is smaller than minimum specified
182 No record of minimum size for file /name/
183 No record of maximum size for file /name/

184 Size of item /name/ overflowed
185 Misplaced redefinition named /name/
186 Redefinition /name/ does not follow item redefined /name/
187 Item with value /name/ is in redefinition or group having value
188 FD or SD /name/ in more than one SAME AREA clause
189 SD /name/ found in SAME RECORD AREA clause
190 SAME RECORD AREA is subset of more than one SAME SORT AREA
191 No FD or SD for file /name/ in SAME AREA clause
200 Priority number for section /name/ exceeds maximum
210 Index item /name/ has same name as another item
220 F Ambiguous or undeclared name /name/ in SELECT clause
221 Name /name/ specified in SELECT clause is not an FD or SD
222 F ACTUAL or NOMINAL KEY specification for file /name/ is ambiguous or undeclared - /name/
223 F RECORD KEY specification for file /name/ is ambiguous or undeclared - /name/
224 More than one SELECT clause found for file /name/
225 F No SELECT clause found for file /name/
226 Device name for file matches that of RERUN file
227 Same file is specified in more than one RERUN clause
228 Illegal to specify RERUN ... EVERY 0 RECORDS ...
229 Invalid file specification in RERUN clause
230 Device specification must be the same on all RERUN clauses
231 Length of RECORD key and NOMINAL key not equal for /name/
232 Tape file /name/ not fixed format
233 Maximum blocksize of device exceeded for file /name/
234 Invalid ACCESS MODE for unit record or tape file /name/
235 Invalid KEY clause for file /name/
236 Invalid RECORD CONTAINS clause for fixed disk file /name/
237 F No ACTUAL KEY for random file /name/
238 Invalid KEY type for access method for file /name/
239 F No NOMINAL KEY for randomly accessed Indexed file /name/
240 F No RECORD KEY for Indexed file /name/
241 Invalid use of RECORDS option in BLOCK clause /name/
242 ACCESS MODE not specified for disk file /name/ - mode assumed.
243 Invalid ACCESS MODE for variable disk file /name/
244 Block size not a multiple of record size for tape file /name/
245 F Record key not part of a record of ISAM file /name/
250 F Undeclared symbol /name/
251 F Ambiguous reference for symbol /name/
252 F Invalid qualifiers given for item /name/
253 F Statement labelled /name/ is not ALTERable
254 F /name/ is not a procedure-name
255 F /name/ used in I/O statement is not an FD or SD
256 F OPEN statement type does not match characteristics of

		device specified for file /name/
257	F	Invalid item used as record in WRITE or REWRITE statement - /name/
258	F	Item used as record in WRITE or REWRITE statement is not a record in an FD or SD - /name/
259	F	Index or subscript applied to non-data item /name/
260	F	Index or subscript applied to non-table item /name/
261	F	Undeclared index name /name/
262	F	Item used as index is not an index name - /name/
263	F	Ambiguous index name /name/
264	F	Index item /name/ is not an index for table /name/
265	F	Wrong number of subscripts or indices given for table /name/
266	F	Item used as subscript is not a numeric data item - /name/
267	F	Data item used as subscript is not an integer - /name/
268	F	Data item used as subscript is a table element - /name/
269	F	Index displacement too large for table element /name/
270	F	No subscripts or indices supplied for table element /name/
271	F	Invalid OPEN statement for SD /name/
272	F	Invalid CLOSE statement for SD /name/
273	F	I/O verb does not match type of file /name/
274	F	Invalid WRITE on sequentail ISAM file /name/
275	F	Invalid OPEN OUTPUT on ISAM file /name/
290		No OPEN statement found for file /name/
291		No CLOSE statement found for file /name/
292		READ statement found for file /name/ which was never opened for INPUT
293		WRITE statement found for file /name/ which was never opened for OUTPUT
294		REWRITE statement found for file /name/ which was never opened for I/O
295		No READ statement found for INPUT or UPDATE file /name/
296		No WRITE statement found for OUTPUT file /name/
297		No READ, WRITE or REWRITE statement found for UPDATE file /name/
298		REWRITE found, but no READ found for UPDATE file /name/
299		No SORT statement found for sort file /name/
300		No input statement found for sort file /name/
301		No output statement found for sort file /name/
302		Invalid LINAGE clause for input or random file /name/
303	F	No NOMINAL KEY given for Indexed file /name/ used with START statement
304	F	OPEN option invalid for device or file type /name/
310	F	Invalid item in PERFORM ... TIMES or GO TO ... DEPENDING /insert/
311	F	GO TO not the only statement in paragraph
312	F	EXIT statement not only statement in paragraph
330		Invalid ROUNDED option ignored

331	F	Arithmetic operand not numeric /name/
332	F	Arithmetic result not numeric or numeric edited /name/
333	F	Invalid sending field for MOVE statement /name/
334	F	Invalid receiving field for MOVE statement /name/
335	F	Invalid combination of operand types in statement - MOVE /name/ TO /name/
336	F	Invalid expression in IF or PERFORM statement
337	F	Invalid expression in computational statement
338	F	Invalid expression in MOVE statement
339		Invalid use of ON SIZE ERROR ignored
340	F	Invalid sending item in SET statement - /name/
341	F	Invalid receiving item or combination of sending and receiving items in SET statement - /name/
342	F	Invalid index comparison in IF or PERFORM
343	F	Type of item /name/ is inconsistent with a NUMERIC class test
344	F	Type of item /name/ is inconsistent with an ALPHABETIC class test
345	F	Invalid identifier specified in EXAMINE statement - /name/
346	F	Item scanned for in EXAMINE is not a literal of length one - /string/
347	F	Replacement item in EXAMINE is not a literal of length one - /string/
348		Composite exceeds maximum lenght. Truncation may occur.
360	F	Missing exception condition in READ or REWRITE statement
361	F	Invalid source in ACCEPT or DISPLAY statement
362	F	Invalid operand in ADVANCING option of WRITE statement for file /name/
363	F	ADVANCING option specified in WRITE statement for non-sequential file /name/
364	F	Invalid ADVANCING option in WRITE statement for file /name/
365	F	ADVANCING option with mnemonic name is invalid for file with page size /name/
366	F	Invalid item in DISPLAY statement /name/
367	F	Invalid item in ACCEPT statement /name/
368		Invalid exception option for READ of /name/ - /insert/ assumed.
369		EOP exception option requires LINAGE clause in FD - page size of 66 assumed /name/
380		Type of literal does not match that of item /name/
381		Non-zero digits or non-blank characters in value for /name/ must be truncated
382		Unsigned item /name/ has signed value

APPENDIX C. COMPILATION PROCEDURE

In order to compile a COBOL source program prepared by the editor, type a command to DOS in the following format:

```
COBOL <sf>{,<cf>}{,rlf}{,clf}{;options}
```

Files

<sf> is the name of the source file; <cf>, the final command object file; <rlf>, a user relocatable library file; <clf>, the COPY library file. The source file is assumed to have an extension of 'TXT' if no explicit extension is used. If no command object file is specified, the source name will be used. The object file is assumed to have an extension of 'CMD'. In order to use the COPY feature, a COPY library file must be specified. COPY libraries are assumed to have an extension of 'INC'. If a relocatable library file is specified, a 'LIBRARY <rlf>' command will be issued to the link editor. This allows the user to specify where COBOL or SNAP subroutines, called from his program are found. The default extension for <rlf> is 'REL'.

In addition to these files, the COBOL system uses the following default file names, unless overidden by the F option. 'COBOLOBJ/REL' is the name of the intermediate relocatable object file produced by the compiler. 'COBOLINK/TXC' is the name of the command file for the link editor, required to create a 'CMD' file from the relocatable object program. 'COBOLSYM/SYM' is the default symbol table file created when the 'USING DEBUGGER' option is specified. In addition to these files 'CBLWORK0/SYS', 'CBLWORK1/SYS', 'CBLWORK2/SYS', and 'CBLWORK3/SYS' will be used as work files for the compiler, unless the W option is used. The user should inspect his disks to determine if adequate space remains on the disks for these seven files or the subsets which will be used.

Options

An option list, indicated by a semi-colon (;) following the file specifications, may be included. The options are order-independent and must be one of the following characters: 'LPOAFEWX'. The action each causes is described below.

L - List on local or servo printer
This option causes the source program to be listed, and the PABs for each segment to be listed as well.

P - List on disk file
This option is used instead of the L option to cause a listing file to be produced for subsequent output. If used the compiler displays the following message:

LIST ON FILE:

and the user should specify a partial file name. If no main name is entered, the main name of the source file is used. If no extension is entered, 'PRT' is assumed. Thus if the source program file name is 'TESTPROG/TXT' and if no name is specified for a print file, 'TESTPROG/PRT' will be the listing file.

O - List the object program
If the L or P option is used, this option causes a SNAP-like listing of the object program to be produced as well as the source listing.

A - Automatic link edit
When the A option is used, the link editor will be invoked immediately after compilation, unless the program contains fatal errors. Otherwise the user is responsible for using the link editor himself.

F - Request intermediate file names
If the F option is used, the compiler will display the message:

ENTER INTERMEDIATE OBJECT FILE NAMES (R,L,S):

and the user should enter only those names which he wishes to change. The R file is the relocatable object file, the L file is the link edit command file, and the S file is the symbol table file. The default extensions for these files are 'REL', 'TXC', and 'SYM', respectively.

E - Continue compilation through fatal errors
The compiler displays diagnostic messages after the source program has been analyzed, before the object program is generated, and after object program generation. Unless the E option is specified, the compiler will terminate compilation at any of these points if one or more fatal errors have been found in the source program.

W - Modify the work file names
This option is used to change the names or assign the drives to be used for the compiler work files. If this option is used, the compiler displays the message:

```
WORK FILE n:
```

where n takes on the values 0 through 15, and allows the user to specify the work file name, as in the P option.

X-Generate cross reference
This option must be used in conjunction with the L or P option. When XL is specified, a cross reference will follow the source listing on the printer. When XP is specified, a cross reference will follow the source listing on the disk file.

APPENDIX D. THE RUNTIME DEBUGGER

The runtime debugging facility allows the user to examine and change the state of his program while it is executing, making the debugging process simpler and faster. In order to use the debugger, the program must contain the USING DEBUGGER option on the OBJECT-COMPUTER specification. This causes a symbol table file to be created for use with the debugging module of the object library, and the debugging module to be link edited into the object program.

When the program is executed, the debugger will be entered before the user's program starts to execute. At this point the user must load the symbol table file for the program using the 'L' command (see below). After loading the symbol table file, any other debugger command may be used. Note that at this time the function of the 'P' command (PROCEED) is to start at the first statement of the user's program.

Commands

ABORT - The program may be terminated by typing 'A' to the debugger when it is waiting for input. At this point, the debugger exits to the operating system.

BREAK - This command is used to specify a paragraph or statement at which normal execution should stop, and the debugger called instead. The debugger allows up to ten breakpoints to be set. These breakpoints remain set until cleared with the CLEAR command. A breakpoint is set by typing 'B', space, and either the name of a paragraph (possibly qualified by 'IN' or 'OF') or '#' followed by a statement number in the PROCEDURE DIVISION.

CLEAR - This command clears breakpoints set by the previous command. Typing 'C', space, and either a paragraph-name or statement-number will clear the breakpoint (if any) on that statement. Typing 'C', enter will clear ALL breakpoints.

DISPLAY - Data items in the COBOL program may be displayed on the screen by typing 'D' followed by the name of the item, which may be qualified and subscripted using integer subscripts. The same rules for uniqueness of data reference apply in the debugger as in the compiler. If the item displayed contains

more than 80 characters, the display will be continued on several lines. The display key on the console may be used to suspend the display at the end of each line.

GO - Control may be transferred to any paragraph or statement by typing 'G' followed by either a paragraph name or '#' statement-number. Control returns to the debugger only at the next breakpoint or if a fatal execution error occurs.

HELP - By typing 'H' enter, a list of the debugger commands will be displayed.

LOAD - A symbol table file for a program or sub-program may be loaded by typing 'L', space, file-name. The default extension is assumed to be 'SYM'. In order to load a symbol table, you must make a link edit listing so that you may enter the correct PAB addresses when requested by the LOAD command. The command will ask for the starting locations of the DATADIV and CODE PABs. After loading a symbol table, the names of data items and procedure names, and the statement numbers for this program are accessible to the debugger.

PROCEED - By typing 'P',enter you may continue the execution of your program after a breakpoint has been encountered. The 'P' command is illegal after an execution error has occurred. Control returns to the debugger only at the next breakpoint or execution error.

SET - The values of data items may be set by typing 'S' followed by the name of the data item (as in DISPLAY). Then the user may type in characters to set the value of the item. The backspace and cancel keys have their normal effect. If more than one line is being entered, the DEL key can be used to end the current line and start a new one. When the ENTER key is used to terminate the line, it also terminates the SET command.

(Debugger) - The ROM debugger may be entered from the COBOL debugger by typing '*', enter. Control is returned to the COBOL debugger by typing 'E' in the ROM debugger. This facility is available for use by machine language programmers, and should be approached with care by less experienced users.

APPENDIX E. RESERVED WORDS

The following list contains all reserved words used by the Datapoint COBOL Compiler

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APOSTROPHE
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BLANK
BLOCK
BY
C01
CALL
CHARACTERS
CLOSE
COMMA
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
COPY
CORE-INDEX
CSP
CURRENCY
DATA
DATABUS
DATE
DATE-WRITTEN
DEBUG
DECIMAL-POINT
DEPENDING
DESCENDING
DISPLAY
DIVIDE
DIVISION
DOWN
DRIVE-NUMBER
EBCDIC
ELSE
END
ENTER
ENVIRONMENT
END-OF-PAGE
EOP
EQUAL
ERROR
EVERY
EXAMINE
EXIT
EXTENSION
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILE-NAME
FILLER
FIRST
FOOTING
FOR
FROM
GIVING
GO
GREATER
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED

KEY
LABEL
LEADING
LEFT
LESS
LINAGE
LINAGE-COUNTER
LINE
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
MAIN-NAME
MEMORY
MODE
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOMINAL
NOT
NOTE
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OR
OUTPUT
PAGE
PERFORM
PIC
PICTURE
POSITIVE
PROCEDURE
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID
QUOTE
QUOTES
RANDOM
READ
RECORD
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
REPLACING
RERUN
RESERVE
RETURN
REWIND
REWRITE
RIGHT
ROUNDED
RUN
SAME
SD
SECTION
SECURITY
SEEK
SELECT
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
SUBTRACT
SW-0
SW-1
SW-2
SW-3
SW-4
SW-5
SW-6
SW-7
SYNC
SYNCHRONIZED
TALLY
TALLYING
THAN
THEN
THROUGH
THRU
TIMES
TO
UNIT
UNTIL
UP
UPON
USAGE
USING
VALUE
VARYING
WHEN
WITH
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

APPENDIX F. COBOL SYSTEM OVERLAY FUNCTIONS

Parser Overlays

BA - Parser initializing Overlay
BC - Identification Division Overlay
BE - Environment Division Overlay
BG - Data Division Parse Overlay
BI - Procedure Division Parser Pass 1
BK - Main Table re-organization
BM - Procedure Division Parser Pass 2
BQ - Parser Termination Overlay

Error message generation

BZ/OA/ZA - Error Message Generation

Symbol Table Overlays

DA - Node Linking Phase
DC - Picture Analysis Phase
DE - Size Computation Phase
DG - Procedure Division Name Generation Phase

Merge Overlays

FA - Build Symbol Table Chains
FC - File Attribute Merge
FK - Data Division Attribute Merge
FM - Build I/O table

Translation Ovelays

KA - Control Translation
KD - Arithmetic Translation
KG - I/O Translation

Index Generation/Optimization Overlay

OE - Subscript/Index Generation and Optimization

PIT Generation Overlays

PA - Parameter PIT Generation

PC - Data Division PIT Generator
PE - Literal Allocation
PG - FDB PIT Generation
PK - Procedure Division PIT Generator
PM - GET PIT Generation
PO - PUT PIT Generation
PP - UPDATE PIT Generation
PW - Generate Prologues & Sort Segments
PZ - Subscript Finalization

Object Program Generation Overlays

WA - OPG initialization
WC - OPG Pass 1
WE - OPG Pass 2
WG - OPG Listing Pass
WI/WJ - Recycle OPG
WM - Generate /SYM file
WO - Generate XREF sort file
WQ - Sort and print XREF
WX - Generate TXC file

Termination Overlays

ZW - Error Messages Lister
ZZ - Termination Phase

APPENDIX G. EXECUTION-TIME ERROR MESSAGE

G.1 Primative Functions

001 Cancel or Abort?

This message is issued following every other execution-time error message when the debugger is not being used. The COBOL system expects a single character reply of either 'C' for Cancel or 'A' for Abort. Cancel has the effect of executing a STOP RUN statement; Abort exits immediately to DOS and aborts any CHAINING that may be going on.

002 Uninitialized GO TO.

A statement of the form GO TO was executed without having been ALTERed.

003 Uninitialized Return Point.

A COBOL program was entered at some point other than the normal entry point and the return mechanism was not properly established. See an S.E.

004 Uninitialized Perform exit.

A paragraph that was being performed was not entered properly and the proper linkage was therefore not established. Consult an S.E.

005 *** Invalid input.

This message is issued whenever a COBOL system function takes input from the keyboard and the input that was entered was invalid. Normally, the function will request the input again and the user should enter the proper information.

006 Enter external switch settings in binary.

A program that makes use of the external switches is being run. The user should enter up to eight 1's and 0's in the proper order for the program. A '1' indicates a switch is ON; a '0' indicates a switch is OFF. All eight switches are set and if fewer than eight settings are entered, the ones that are not entered will be set to OFF.

007 Continue execution?

This message follows the literal specified by the execution of a STOP <literal> statement. The system expects a reply of 'Y' for YES or 'N' for NO. The no reply has the effect of a STOP RUN statement. The yes reply causes execution to continue at the next statement in the program.

008 Program xxxxxxxx/yyy not found.

Independent segment 'yyy' of the COBOL program 'xxxxxxxx' was not found in the DOS directory.

009 Loaded over caller.

An independent segment of the COBOL program was not correctly link-edited and loads over a portion of the permanent segment.

G.2 Data Type Conversion Functions

050 Negative number has excessive magnitude.

A negative number being moved to data item described with USAGE DATABUS contains too many digits.

051 Invalid character in convert to computational.

A data item being moved to a numeric field contains characters that are not numeric.

052 Invalid digit in comp to binary conversion.

A numeric data item being used as a subscript or being moved to an index name contains characters that are not numeric.

G.3 Arithmetic Functions

100 Invalid data in Exponentiation.

An attempt has been made to raise a negative number to a non-integral power.

G.4 Disk Input/Output Functions

150 File format error (filename).

A file (filename) with GEDIT format contains invalid format.

151 File format error (filename).

A file (filename) with fixed length records contain invalid format

152 Input record shorter than record length (filename).

A record on a file (filename) having fixed length records is shorter than the record length specified for that file.

153 Input record longer than record length (filename).

A record on a file (filename) having fixed length records is longer than the record length specified for that file.

154 Index matched low key (filename).

The format of indexed file (filename) is incorrect. Consult an S.E.

155 DR$ Read error.

An error was detected when trying to read the indexed file (filename). Consult an S.E.

156 File format error (filename).

File (filename) has invalid format.

157 Attempted to REWRITE I-O file with no READ (filename).

(Self-explanatory).

158 File format error when extending.

An error was detected when WRITEing a new record to a keyed file. Consult an S.E.

159 Attempted READ on file not open INPUT or at EOF (filename).

(Self-explanatory).

160 Attempted START on file not open INPUT (filename).

(Self-explanatory).

161 Update file must be open I-O to write on (filename).

A file for which REWRITE statements are used must be opened for I-O when WRITE statements are issued.

162 Attempted to update a record with no previous READ (filename).

(Self-explanatory).

163 Attempted to write to file not open for OUTPUT (filename).

(Self-explanatory).

164 Tried to OPEN file already open (filename).

(Self-explanatory).

165 ISAM file key does not match FDB specification (filename).

The key description for an indexed file specified by the RECORD KEY clause for a file does not match the key upon which the file is indexed.

166 ISAM data file open error (filename).

An error occurred while trying to open an indexed file.

167 GETLFT called an unopened file (filename).

This is an internal error. Consult an S.E.

168 Invalid LFT # from FDB in GETLFT. (filename).

This is an internal error. Consult an S.E.

169 Invalid LFT # in RLSLFT.

This is an internal error. Consult an S.E.

G.5 Console Input/Output

200 Tried to open file already open. (filename).

(Self-explanatory).

G.6 Printer Input/Output

250 Invalid opening of printer file. (filename).

Either file is already opened or another file is currently open on the SERVO-PRINTER.

251 Invalid record-length. (filename).

This is an internal error. Consult an S.E.

G.7 Magnetic Tape INPUT/OUTPUT

300 No write ring for output tape. (filename).

(Self-explanitory).

301 Deck not in service. (filename).

The magnetic tape deck is not ready.

302 Attempted to open second tape file. (filename).

Only one magnetic tape file may be open at one time.

303 End of tape found. (filename).

Physical end of tape was reached during input. This indicates a tape format error.

304 Parity error on tape. (filename).

(Self-explanatory).

305 Bad tape. (filename).

A parity error was detected on the tape during output.

306 FDB Addresses do not match in tape close.

Internal error;consult an S.E.

307 Format Correct, Continue? (Y or N).

The format of the labels for an input tape as been analyused and is correct. The labels are displayed before the message. The operator is requested to verify that the tape mounted is correct, that the labels are correct, and, if so, reply 'Y' to continue execution. The user should reply 'N' if either are not correct.

308 Missing VOL1.

A labeled tape does not contain a VOL1 label.

309 Invalid label record length.

The length of a label record is not correct.

310 Not ANS VOL1 label.

The VOL1 label for a file described as TAPE-A is not of the proper format.

311 Missing or invalid HDR1.

(Self-explanatory).

312 Missing or invalid HDR2.

(Self-explanatory).

313 Format not = fixed.

Only F(fixed) format magnetic tapes may be processed.

314 Wrong block size in label.

The block size specified in the tape label does not match that specified for the file.

315 Wrong record size in label.

The record size specified in the tape label does not match that specified for the file.

316 Operator requested.

The operator is requested in order to verify the labels being generated for a tape output file.

317 Missing EOF1.

(Self-explanatory).

318 Block count not = count in trailer.

The number of blocks read from a labeled tape does not match the number of blocks specified in the trailer label.

G.8 Card Reader

350 Invalid opening of card file. (filename).

An attempt was made to open a card file for some use other than input.

351 Tried to open file already open. (filename).

(Self-explanatory).

352 More Cards?

The card reader is empty. The operator should either put more cards in the reader and reply 'Y' or reply 'N' if all of the cards have been read.

353 Reply A for ABORT; R for RESUME.

This occurs on abnormal condition with card reader. The operator is requested to reply appropriately.

354 ** READER CHECK !

(Self-explanatory).

G.9 Cassette Input/Output

400 Tried to open file already open. (filename).

(Self-explanatory).

401 Invalid opening of cassette file. (filename).

The cassette drive for the file is already in use.

402 Read parity error. (filename).

(Self-explanatory).

403 End of tape. (filename).

(Self-explanatory).

404 Cannot position tape. (filename).

(Self-explanatory).

405 End of Cassette tape.

(Self-explanatory).

406 *** ERROR X on Deck Y ***

(See DOS manual regarding cassette tapes.)

G.10 Sort Facility

450 SORT attempted with insufficient memory.

A minimum of 8K free space is needed in order to SORT.

451 Sort key too long.

The longest key allowable is 244 bytes.

452 SPACE TRAP in sort key output.

(Self-explanatory).

453 Attempt to sort null file.

(Self-explanatory).

454 SPACE TRAP in merge file output.

(Self-explanatory).

G.11 SPECIAL I/O Facility

500 Attempt to OPEN file already open. (filename).

(Self-explanatory).

Manual Name_____

Manual Number_____

READER'S COMMENTS

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____

All comments and suggestions become the property of Datapoint.

Fold Here

Fold Here and Staple

First Class
Permit
5774
San Antonio
Texas

BUSINESS REPLY MAIL
No Postage Necessary if mailed in the United States

Postage will be paid by:

DATAPOINT CORPORATION
Product Marketing
8400 Datapoint Drive
San Antonio, Texas 78284