



MOTOROLA

M68COB(D)

**M6800
RESIDENT COBOL
LANGUAGE REFERENCE MANUAL**

SYSTEMS

Microsystems

M6800

RESIDENT COBOL

LANGUAGE REFERENCE MANUAL

This manual describes the programming language features of the Motorola Resident COBOL compiler. An associated manual—RESIDENT COBOL OPERATIONS REFERENCE MANUAL—describes the use and operation of both the compiler and the operating system.

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the product described any license under the patent rights of Motorola Inc. or others.

Motorola reserves the right to change specifications without notice.

Second Edition
Copyright 1978 by Motorola Inc.
First Edition January 1978

PREFACE

M6800 COBOL is based on the specification of the COBOL standard published by the American National Standards Institute (formerly known as the United States of America Standards Institute) and contained in the publication USA Standard COBOL X3.23—1974.

As its name implies, COBOL (COmmon Business Oriented Language) is especially efficient in the processing of business problems. Such problems typically involve relatively little algebraic or logical processing; instead, they most often manipulate large files of basically similar records in a relatively simple way. This means that COBOL emphasizes mainly the description and handling of data items and input/output records.

This publication explains Motorola M6800 ANS COBOL, which is a compatible subset of American National Standard COBOL and includes a number of extensions to it as well. The compiler supports the processing modules defined in the standard. These processing modules include the following:

NUCLEUS defines the permissible character set and the basic elements of the language in each of the four COBOL divisions: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, PROCEDURE DIVISION.

TABLE HANDLING allows the definition of tables of contiguous data items and accessing these items through subscripts.

SEQUENTIAL ACCESS allows the records of a file to be accessed in an established sequence. It also provides for the specification of rerun points and the sharing of memory area among files.

RANDOM ACCESS allows the records of a mass storage file to be accessed in a random manner specified by the programmer. Specifically defined keys, supplied by the programmer, control successive references to the file. It also provides for the specification of rerun points and the sharing of memory area among files.

LIBRARY allows the programmer to specify text that is to be copied from a library. This supports the retrieval and updating of prewritten source program entries from a user's library, for inclusion in a COBOL program at compile time. The effect of the compilation of library text is as though the text were actually written as part of the source program.

PRINCIPLES OF COBOL

COBOL is one of a group of high-level computer languages. Such languages are problem oriented and relatively machine independent, thus freeing the programmer from many of the restrictions of assembler language and allowing him to concentrate upon the logical aspects of his problem.

COBOL looks and reads much like ordinary business English. The programmer can use English words and conventional arithmetic symbols to direct and control the computer operations. A few typical COBOL sentences follow:

```
ADD NEW-PURCHASES TO TOTAL-CHARGES.  
MULTIPLY QUANTITY BY UNIT-PRICE GIVING INVENTORY-VALUE.
```

PERFORM FEDERAL-TAX-CALCULATIONS.
IF ITEM-CODE IS NUMERIC GO TO CHECK-ACCOUNT-NUMBER.

Such COBOL sentences are easily understandable, but they must be translated into machine language—the internal instruction codes—before they can actually be used.

A special systems program, known as a compiler, is first entered into the computer. The COBOL program (referred to as the source program) is then entered into the machine, where the compiler reads it and analyzes it. The COBOL language contains a basic set of reserved words and symbols. Each combination of reserved words and symbols is transformed by the compiler into a definite set of machine instructions. In effect, the programmer has at his disposal a whole series of “prefabricated” portions of the machine-language program he wishes the compiler to construct.

When the programmer writes a COBOL program, he is actually directing the compiler to bring together, in the desired sequence, the groups of machine instructions necessary to achieve the desired result. From the programmer’s instructions, the compiler creates a new program in machine language. This program is known as an object program.

ORGANIZATION OF MANUAL

A COBOL source program consists of information in four divisions: the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Taken together, these divisions constitute the total program (including a description of the configuration needed, the forms of various data files, and the programming steps necessary to perform these procedures), and are presented to the processor for compilation into a corresponding object program.

In this manual, M6800 ANS COBOL is described as follows:

- Chapter 1 describes the COBOL language structure. It presents the COBOL theory behind word formation, the use of words to name elements in a program, and a discussion of the syntax of the language.
- Chapter 2 contains a discussion of the format and organization of data in files, together with methods used to remove data from, or place data into, such files.
- Chapters 3 through 6 present a detailed description of the IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE DIVISIONS, respectively.
- Chapter 7 contains a description of the statements that affect the COBOL library.
- The appendixes contain supplementary information: a list of ANS COBOL reserved words; a sample M6800 ANS COBOL problem; and a list of M6800 ANS COBOL data types.

M6800 Extensions to the ANS COBOL Standard

Listed below are M6800 extensions to the ANS COBOL standard. Although these extensions do not conform to the ANS Standard, they may be compatible with language forms used by other manufacturers. Wherever possible an attempt has been made to keep all extensions in conformance with the generally accepted industry usage.

LANGUAGE CONCEPTS

Apostrophe is used as a default value for quotation mark.
Hexadecimal constants.

IDENTIFICATION DIVISION

None.

ENVIRONMENT DIVISION

None.

DATA DIVISION

LINE and COLUMN clauses for CRT formatting.

PROCEDURE DIVISION

@(row, column) clause on DISPLAY statement.

ON SIZE ERROR clause on MOVE statement.

De-edit action for move of picture X to picture 9.

The word THEN may be used to separate statements.

ACKNOWLEDGMENT

In compliance with the request of the Executive Committee of the Conference on Data System Languages (CODASYL), and specifically the CODASYL COBOL Committee, the following acknowledgment is extracted from that contained in the publication *COBOL, Edition 1974*.

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

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

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

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

“The authors and copyright holders of the copyrighted material used herein 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.”

†*COBOL, Edition 1965*, produced by joint efforts of the CODASYL COBOL Committee and the European Computer Manufacturers Association (ECMA).

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959 by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

COMMAND SYNTAX NOTATION

Notation conventions used in command specifications and examples throughout this manual are listed below.

Notation	Description
lowercase letters	<p>lowercase letters identify an element that must be replaced with a user-selected value.</p> <p>CRn<code>dd</code> could be entered as CRA03.</p>
CAPITAL LETTERS	<p>Capital letters must be entered as shown for input, and will be printed as shown in output.</p> <p>DPn<code>dd</code> means "enter DP followed by the values for n<code>dd</code>."</p>
[]	<p>An element inside brackets is optional. Several elements placed one under the other inside a pair of brackets means that the user may select any one or none of those elements.</p> <p>[KEYM] means the term "KEYM" may be entered.</p>
$\left. \begin{array}{l} \\ \\ \end{array} \right\}$	<p>Elements placed one under the other inside a pair of braces identify a required choice.</p> <p>A id means that either the letter A or the value of id must be entered.</p>
. . .	<p>The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.</p> <p>name[,name] . . . means that one or more name values may be entered, with a comma inserted between each name value.</p>
<p>.</p> <p>.</p> <p>.</p>	<p>The vertical ellipsis indicates that commands or instructions have been omitted.</p> <p>OPEN MASTER-FILE. : : CLOSE MASTER-FILE.</p> <p>means that there are one or more statements omitted between the two commands.</p>
Numbers and special characters	<p>Numbers that appear on the line (i.e., not subscripts), special symbols, and punctuation marks other than dotted lines, brackets, braces, and underlines appear as shown in output messages and must be entered as shown when input.</p> <p>(value) means that the proper value must be entered enclosed in parentheses; e.g., (234).</p>
Subscripts	<p>Subscripts indicate a first, second, etc., representation of a parameter that has a different value for each occurrence.</p> <p>name₁, name₂, name₃ means that three successive values for name should be entered, separated by commas.</p>

CONTENTS

ACKNOWLEDGMENT	vii
PREFACE	iii
COMMAND SYNTAX NOTATION	viii
CHAPTER 1. COBOL LANGUAGE STRUCTURE	1-1
Introduction	1-1
Character Set	1-2
Words	1-3
Definition and Application	1-3
Reserved Words	1-6
Concept of Computer-Independent Data Description	1-7
Algebraic Signs	1-8
Uniqueness of Data Reference	1-9
Format Notation	1-10
Reference Format	1-11
CHAPTER 2. COBOL INPUT/OUTPUT PROCESSING	2-1
COBOL Files	2-1
File Organization	2-1
File Access	2-1
File-Handling Methods	2-2
Input/Output Processing Summary	2-2
CHAPTER 3. IDENTIFICATION DIVISION	3-1
General Description	3-1
Organization	3-1
PROGRAM-ID Paragraph	3-1
DATE-COMPILED Paragraph	3-1
CHAPTER 4. ENVIRONMENT DIVISION	4-1
General Description	4-1
Configuration Section	4-1
SOURCE-COMPUTER Paragraph	4-1
OBJECT-COMPUTER Paragraph	4-2
INPUT-OUTPUT Section	4-2
FILE-CONTROL Paragraph	4-2
SELECT Sentences	4-2
ASSIGN Clause	4-2
ACCESS Clause	4-3
RECORD KEY Clause	4-3
I-O-CONTROL Paragraph	4-4
SAME AREA Clause	4-4

CHAPTER 5. DATA DIVISION	5-1
General Description	5-1
Physical and Logical Aspects of Data Description	5-1
DATA DIVISION Organization	5-1
DATA DIVISION Structure	5-1
File Section	5-1
Working-Storage Section	5-3
File Description—Complete Entry Skeleton	5-3
RECORD CONTAINS Clause	5-4
LABEL RECORDS Clause	5-4
DATA RECORDS Clause	5-4
Data Description Entries	5-5
REDEFINES Clause	5-8
COPY Statement	5-9
PICTURE Clause	5-9
USAGE Clause	5-15
BLANK WHEN ZERO Clause	5-15
JUSTIFIED Clause	5-15
VALUE Clause	5-16
OCCURS Clause	5-16
CRT Form Descriptions	5-17
 CHAPTER 6. PROCEDURE DIVISION	 6-1
General Description	6-1
PROCEDURE DIVISION Elements	6-1
PROCEDURE DIVISION Structure	6-2
Arithmetic Expressions	6-2
Order of Computation in Compound Expressions	6-3
Conditional Statements	6-4
Relations	6-4
Logical Operators (AND, OR, and NOT)	6-4
Comparison of Numeric Items	6-5
Comparison of Non-Numeric Items	6-5
Conditional Statements with Exception Branches	6-7
Nested Conditional Statements	6-7
Input/Output Statements	6-7
OPEN Statement	6-7
START Statement	6-8
READ Statement	6-9
WRITE Statement	6-10
CLOSE Statement	6-11
ACCEPT Statement	6-11
DISPLAY Statement	6-11
Arithmetic Statements	6-12
Rules for Arithmetic Verbs	6-12
ADD Statement	6-14
SUBTRACT Statement	6-15
MULTIPLY Statement	6-16

DIVIDE Statement	6-17
COMPUTE Statement	6-18
Data Manipulation Statements	6-18
MOVE Statement	6-18
INSPECT Statement	6-23
Sequence Control Statements	6-24
Normal Sequence Control	6-25
GO TO Statement	6-25
PERFORM Statement	6-26
STOP Statement	6-30
EXIT Statement	6-30
IF Statement	6-30
Table Handling Statements	6-31
SEARCH Statement	6-32
SET Statement	6-33
Compiler-Directing Statements	6-34
COPY Statement	6-34
CHAPTER 7. COBOL LIBRARY	7-1
Introduction	7-1
COPY Statement	7-1
CHAPTER 8. DEBUGGING	8-1
EXHIBIT Statement	8-1

APPENDIXES

APPENDIX A. ANS COBOL RESERVED WORDS	A-1
APPENDIX B. SAMPLE M6800 COBOL PROGRAM	B-1
APPENDIX C. INTERNAL DATA TYPES	C-1

CHAPTER 1

COBOL LANGUAGE STRUCTURE

INTRODUCTION

COBOL (the *CO*mmun *B*usiness *O*riented *P*rogramming *L*anguage) consists of selected English words that impart key meanings to the COBOL compiler. The language is arranged into statements, sentences, and paragraphs in a manner similar to written English. The words of this language are selected English words (called "reserved words" because they cannot be used in any other context), names of data and procedures, and numeric or non-numeric "literals." Punctuation is permitted, but the only meaningful punctuation is the period.

COBOL words are arranged into statements using the formats described in this manual in the separate discussion of each statement. One or more statements compose a sentence, which is terminated by a period. One or more sentences, in turn, constitute a paragraph, which can be given a name so that control can pass to the paragraph by referencing its name elsewhere in the program. Similarly, several paragraphs make up a section that can also have a name and, in addition, can be loaded as an "overlay." Several sections constitute a division. There are four divisions in a COBOL program, each describing a different, important part of the program.

Structural hierarchy of the COBOL programming language and the purpose of each level therein are:

- The COBOL Program Contains all the information required to perform a given task on the computer.
- Division Describes a specific category of information essential to the compiler or, in the case of the PROCEDURE DIVISION, specifies processing steps.
- Section In the PROCEDURE DIVISION, defines the smallest block of the program that can be loaded at one time or as an overlay, in other divisions, groups a particular type of information within a division.
- Paragraph Comprises one or more sentences forming the smallest block of the program that can be referenced by name.
- Sentence Consists of one or more statements terminated by a period.
- Statement Consists of a group of words that perform only one operation or function in the program.
- Word Consists of a group of characters and/or symbols that provide the structural basis of a statement.

In addition, another type of structure is permitted and fits into the hierarchy in place of "word." This is the structure of mathematical notation and is discussed in detail in "Arithmetic-Expressions" in Chapter 6.

CHARACTER SET

The complete character set for M6800 ANS COBOL consists of the 51 characters listed below.

<u>Character</u>	<u>Meaning</u>
0-9	digits
A-Z	letters
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equals sign
\$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
“	double quotation mark
(left parenthesis
)	right parenthesis
>	greater than sign
<	less than sign
'	single quotation mark

Characters Used for Punctuation

The following characters are used for punctuation:

<u>Character</u>	<u>Meaning</u>
	space
,	comma
;	semicolon
.	period
“ or ‘	quotation mark
(left parenthesis
)	right parenthesis

The following general rules of punctuation apply in writing a COBOL source program:

1. When any punctuation mark is indicated in a format in this publication, it is required in the program.
2. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except within non-numeric literals.
3. An arithmetic operator or an equal sign *must* be preceded by a space and followed by a space. A unary operator may be preceded by a left parenthesis.

4. A comma may be used as a separator between successive operands of a statement. An operand of a statement is shown in a format as a lower-case word.
5. A comma or a semicolon may be used to separate a series of clauses. For example, DATA RECORD IS TRANSACTION, RECORD CONTAINS 80 CHARACTERS.
6. A semicolon may be used to separate a series of statements. For example, ADD A TO B; SUBTRACT B FROM C.
7. The word THEN may be used to separate a series of statements. For example, IF A = B THEN SUBTRACT B FROM C.

Characters Used for Editing

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

<u>Character</u>	<u>Meaning</u>
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppression
*	check protection
\$	currency sign
,	comma
.	period (decimal point)

(For applications, see the discussion of alphanumeric edited and numeric edited data items in "Data Division.")

Characters Used for Relation Conditions

A *relation character* is a character that belongs to the following set:

<u>Character</u>	<u>Meaning</u>
<	greater than
>	less than
=	equal to

Relation characters are used in relation conditions (discussed in "Procedure Division"). The word NOT may precede the relation character.

WORDS

Definition and Application

The character set for words comprises 37 characters: the letters A through Z, the digits 0 through 9, and the hyphen. A word is composed of a combination of not more than 30 such characters chosen from this set with the following exceptions:

1. A word cannot begin or end with a hyphen.
2. The space (blank) is not an allowable character in a word and is used as a word separator. Where a space (blank) is required, more than one may be used except for the restrictions stated in this chapter (see "Reference Format"). A word is ended by a space, period, right parenthesis, comma, or semicolon.

Rules for using punctuation characters in connection with words are:

1. If ANS-68 compatibility is desired, a space should follow a period, comma, or semicolon when one of these punctuation characters is used to terminate a word, and a space should not immediately follow a left parenthesis or immediately precede a right parenthesis.
2. A space must not immediately follow a beginning quotation mark or precede an ending quotation mark, unless a space is desired in the literal (which is enclosed in quotation marks).

Data-Name

A data-name is a word with at least one non-numeric character that names a data item in the DATA DIVISION. A space (blank) is not allowed within a data-name, and ANS COBOL reserved words must not be used. (See appendix A, "M6800 ANS COBOL Reserved Words.")

Procedure-Name

A procedure-name is either a paragraph-name or a section-name. A procedure name may be composed solely of numeric characters. However, two numeric procedure-names are equivalent only when they are composed of the same number of digits and have the same value: for example, 0023 is not equivalent to 23.

Literal

A literal is a string of characters whose value is defined by the set of characters composing the literal. Every literal is one of two types: non-numeric or numeric.

A non-numeric literal is a string of any allowable ASCII characters (including reserved words but excluding the quotation mark character) up to 255 characters in length, bounded by quotation marks. The single quotation mark (') is normally used by default, but the double quotation mark (") may be specified if conformance with the ANS character set is desired. 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 literal and therefore part of the value. All non-numeric literals are classed as alphanumeric.

A numeric literal is a string of characters selected from digits 0 through 9 (to a maximum of 15 digits), the plus sign, minus sign, and decimal point. The value of a numeric literal is the algebraic quantity represented by the characters in the literal. Every numeric literal is classed as numeric.

Rules for the formation of numeric literals are:

1. The literal must contain *at least one digit*.
2. The 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, it is positive.
3. The 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, it is an integer.

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

Figurative-Constants

Figurative-constants are certain constants to which fixed data-names are assigned. Such data-names must not be bounded by quotation marks when used as figurative-constants. Singular and plural forms of figurative-constants are equivalent and may be used interchangeably.

Fixed data-names and their meanings:

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 characters that have the highest value in the ASCII collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more characters that have the lowest value in the ASCII collating sequence.
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 non-numeric literal.
ALL literal	Represents one or more of the string of characters comprising the literals. The literal must be either a non-numeric literal or a figurative-constant other than ALL literal. When a figurative-constant is used, the word ALL is redundant and is used for readability only.

When a figurative-constant represents a string of one or more characters, the compiler determines the length of the string from context in accordance with the following rules:

1. When a figurative-constant is associated with another data item, that is, 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, that is, when the figurative-constant appears in a DISPLAY or STOP statement, the length of the string is one character. The figurative-constant ALL literal may not be used with DISPLAY or STOP.

A figurative-constant can be used wherever a literal appears in the format, except that whenever the literal is restricted to having only numeric characters.

Special Registers

COBOL has several built-in registers that provide an interface to the operating system. These register names may be used in IF and MOVE statements in the procedure division. They may be read by the program, but may not be altered by the program. The special registers are:

- DATE. Yields a 6-character date in the form MMDDYY.
77 TODAY PIC 99/99/99.
MOVE DATE TO TODAY.
- BREAK-KEY. Yields a one-character code to indicate whether or not the break key has been depressed on the CRT. BREAK-KEY will return "Y" if a break has occurred and an "N" if a break has not occurred.
IF BREAK-KEY EQUALS 'Y' THEN STOP RUN.
- LINAGE-COUNTER. Yields the current printer line number. The value does not include "TOP" lines specified by the printer FD clause "TOP IS."
IF LINAGE-COUNTER EQUALS 60 THEN PERFORM TOP-PAGE.

Hexadecimal Constants

A hexadecimal constant is a string of hexadecimal digits preceded by a dollar sign. Each pair of digits represents the contents of one M6800 byte. All hexadecimal constants are considered non-numeric literals and may be used in any context where an alphanumeric literal is allowed.

Examples:

```
77 FLD-MARK PIC X VALUE $FF.  
MOVE $30313233 TO DATA.  
DISPLAY $8386.
```

Reserved Words

Reserved words are used for syntactical purposes and cannot appear as user-defined words. (See Appendix A, "M6800 ANS COBOL Reserved Words.")

The three types of reserved words are key words, optional words, and connectives.

Key Words

A key word 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.

The three types of key words are

1. Verbs such as ADD, READ, PERFORM.
2. Required words (in statement and entry formats) such as TO and GIVING.
3. Words that have a specific functional meaning such as NUMERIC, SECTION, etc.

Optional Words

Within each format, uppercase words that are not underlined are called optional words and can appear at user discretion. The presence or absence of each optional word within a format does not alter compiler translation. Misspelling an optional word or its replacement by another word of any kind is not allowed.

Connectives

The two types of connectives are:

1. Qualifier connectives (used to associate a data-name or a paragraph-name with its qualifier) such as OF and IN.
2. Logical connectives (used in the formation of conditions) such as AND, OR, AND NOT, OR NOT.

Concept of Computer-Independent Data Description

To make data as computer-independent as possible, 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; it 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.

Logical Record and File Concept

The following discussion defines file information by distinguishing 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 data as it appears on the input or output media and include such features as:

1. The mode in which the data file is recorded on the external medium.
2. The grouping of logical records within the physical limitations of the file medium.
3. 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 logical record and a physical 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 contained on a device.

A logical record can be contained within a single physical unit or it may require more than one physical unit to contain it. There are several source language methods available for describing the relationship between logical records and physical units. Once the relationship is established, control of accessibility of logical records as related to the physical unit is the responsibility of the object program. In this manual, references to records are to logical records unless the term "physical record" is specified.

The concept of a logical record is not restricted to file data but applies also to the definition of working-storage and linkage section. Thus, working-storage and linkage section items may be grouped into logical records and defined by a series of Record Description entries.

Record Concepts

The Record Description entry consists of a set of Data Description entries that describe the characteristics of a particular record. Each Data Description entry comprises a level-number followed by a data-name (if required) and a series of independent clauses (as required).

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 record for the purpose of data reference. Once a subdivision is specified, it may be subdivided further to permit more detailed data referencing.

The most basic subdivisions of a record—that is, those not further subdivided—are called elementary items; consequently, a record consists of a sequence of elementary items, or the record itself may be an elementary item.

For ease of reference, a set of elementary items is combined into a group. Each group consists of a named sequence of one or more elementary items. These groups, in turn, may be combined into multiples of two or more; 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 to a maximum of 15. Special level-number 77 is an exception 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 an elementary or a group item) immediately following the last elementary item of the previous group must be the same as that of one of the groups to which the prior elementary item belongs.

Noncontiguous working-storage and linkage section items that are not subdivisions of other items and are not themselves subdivided are assigned the special level-number 77.

Initial Values of Tables

In the WORKING-STORAGE SECTION, initial values of elements within tables are specified in the following way:

The table may be described as a record by a set of contiguous Data Description entries, each of which specifies the "value" of an element, or part of an element, of the table. In defining the record and its element any Data Description clause (USAGE, PICTURE, etc.) may be used to complete the definition, where required. This form is necessary when the elements of the table require separate handling. The hierarchical structure of the table is then shown by the use of the REDEFINES entry and its associated subordinate entries; these subordinate entries, which are repeated due to OCCURS clauses, must not contain VALUE clauses.

Algebraic Signs

Algebraic signs are used (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.

Most forms of representation have a standard or normal manner of depicting an operational sign. Thus, an indication that an operational sign is associated with an item is usually sufficient. Since some forms of representation allow alternative methods for depicting operational signs, it is possible to describe certain types of operational signs that deviate from

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

Uniqueness of Data Reference

Every name used in a COBOL source program must be unique, that is, no other name may have the identical spelling.

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that are not assigned individual data-names. (See "OCCURS Clause" under "Physical and Logical Aspects of Data Description" in Chapter 5).

The subscript can be represented by a numeric literal that is an integer, or by a data-name, or by a combination of data-name and numeric literal. 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 can not be subscript.

The subscript may contain a sign, but the lowest permissible subscript value is 1. The highest permissible subscript value in any particular case is the number of maximum 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 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.

The composite format of a subscripted data-name is:

$$\text{data-name} \left(\text{subscript-1} \left[, \text{subscript-2} \left[, \text{subscript-3} \right] \right] \right)$$

The composite format of a subscript is:

$$\left[\begin{array}{l} \text{integer - 1} \\ \text{data-name - 1} \left[\left\{ \pm \right\} \text{integer - 1} \right] \end{array} \right]$$

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 by 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 "SET Statement" under "Table-Handling Statements" in Chapter 6.)

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, and all are enclosed in parentheses immediately after the terminal space of the data-name.

The composite format is

$$\text{data-name} \left(\begin{array}{l} \text{index-name-1} \left[\left\{ \pm \right\} \text{integer-1} \right] \\ \left[\text{index-name-2} \left[\left\{ \pm \right\} \text{integer-2} \right] \right] \\ \left[\text{index-name-3} \left[\left\{ \pm \right\} \text{integer-3} \right] \right] \end{array} \right)$$

Restrictions on Indexing and Subscripting

Tables may have one, two, or three dimensions. Therefore, references to an element in a table may require up to three subscripts or indexes.

A data-name cannot be subscripted or indexed when it is used in table-element references as an index subscript.

Subscripting and indexing must not be used together in a single reference. Where subscripting is not permitted, indexing is also not permitted.

An index can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values of the index-names as data without conversion; such data items are called index data items.

Format Notation

The format of a COBOL statement is described in this manual using the uniform notations itemized below. See also COMMAND SYNTAX NOTATION.

1. A COBOL reserved word, printed entirely in capital letters, is a word that is assigned specific meaning in the COBOL system. It must not be used in any context or position other than that shown in the format description. SUBTRACT, FROM and ROUNDED in the example below are reserved words.
2. One or more COBOL elements vertically stacked and enclosed in a set of square brackets indicate that this portion of the syntax is optional and may be included or omitted at the discretion of the programmer.
3. A pair of braces is used to enclose vertically stacked COBOL elements when one, and only one, of the elements is required; the others are to be omitted. Refer to the example below.
4. The ellipsis . . . denotes a succession of operands or repeated COBOL elements that may be used in the same particular statement, even though the operands or elements are omitted in the text. An ellipsis is associated with the last complete element preceding it, i.e., if a group of operands and key words are enclosed within brackets and the right bracket is followed by the ellipsis, the group (and not merely the last operand) may be repeated in its entirety.
5. An underlined word is required unless the part of the format containing it is itself optional (enclosed in brackets). If a required word is omitted or incorrectly spelled, it causes an error in the interpretation of the program.
6. All COBOL words that are optional words (not underlined) may be included or omitted at the option of the programmer. These words are used only for the sake of readability; misspelling, however, constitutes an error.
7. Lowercase words represent information that is supplied by the programmer. The nature of the information required is indicated in each case. In most instances the programmer is required to provide an appropriate data-name, procedure-name, literal, etc. Refer to the example below.

8. The period is the only required punctuation. Other punctuation, where shown, is optional.
9. Special characters (such as the equal sign) are essential where shown, although they may not be underlined.
10. The notation ▲ indicates the position of an assumed decimal point in an item.
11. A numeric character with a plus or minus sign above it (n) indicates that the value of the item has an operational sign that is stored in combination with the numeric character.
12. Character positions in storage are shown by boxes

A	B	C	D
---	---	---	---

. An empty box means an unpredictable result.
13. The symbol △ indicates a space (blank).

The following example shows a typical COBOL statement and use of the notation described above.

SUBTRACT { identifier-1 } [,identifier-2] . . . FROM Identifier-m
 { literal-1 } [,literal-2]
 [ROUNDED]

Reference Format

General Description

The reference format, which provides a method for describing COBOL source programs, is described in terms of character positions or columns on a CRT line. The line may be up to 80 characters in length. Rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing. Division of a source program is 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.

The standard COBOL line format is as follows:

Columns 1-6	six-digit sequence number
Column 7	continuation area
Columns 8-11	area A
Columns 12-72	area B
Columns 73-80	identification area

Since the M6800 COBOL programs are maintained by the M6800 EDITOR, a slightly more compact format is used:

Columns 1-4	four-digit line number
Column 6	continuation area
Columns 6-7	area A
Columns 8-80	area B

The sample program shown in Appendix B is an example of the compressed format. If line format compatibility with the COBOL standard is desired, the following format should be used:

Columns 1–4	four-digit line number
Column 6	continuation area
Columns 7–10	area A
Columns 11–71	area B

The line numbers may then be easily expanded to six digits with the EDITOR prior to writing the source program to external media.

Reference Format Representation

Margin L	designates the line number area consisting of four digits followed by a space.
Margin C	represents the continuation column — column 6. An * (asterisk) in margin C causes the compiler to treat the entire line as a comment line. A / (slash) in Margin C will cause the compiler to start printing the source program on the top of a new page. The remainder of the line is treated as a comment. A - (hyphen) in margin C is used to continue a non-numeric literal from one line to the next.
Margin A	represents the first column in the coding area. Normally, this will be the same column as margin C (column 6). However, column 7 may be used if desired.
Margin B	represents the second area in coding portion of the line. Normally, column 8 is used. However, column 11 may be used if compatibility with the standard COBOL line format is desired.

Continuation of Non-Numeric Literals. When a non-numeric literal is continued from one line to another, a hyphen is placed in Margin C of the continuation line and a quotation mark is placed in Area B following the hyphen. All spaces at the end of the continued line and any spaces following the quotation mark of the continuation line and preceding the final quotation mark of the literal are considered part of the literal. Note that each line in this system is terminated by a carriage return. If it is desired that additional spaces are to be included at the end of the continued line, they must actually be typed in.

Division, Section, and Paragraph Formats

Division Header. The division header must be the first line of a division reference format. The division header starts in margin A with the division-name followed by a space, the word DIVISION, and a period. No other text may appear on the same line as the division header.

Section Header. The section header begins on any line except the first line of a division reference format. The section header starts in Area A with the section-name followed by a space, the word SECTION, and a period followed by a space. No other text may appear on the same line as the section header.

A section consists of paragraphs in the ENVIRONMENT and PROCEDURE DIVISIONS and Data Description entries in the DATA DIVISION. Paragraph-names but not section-names are permitted in the IDENTIFICATION DIVISION.

Paragraph-Name and Paragraphs. The name of a paragraph starts in Area A of any line following the first line of a division reference format (or section header if sections are used) and ends with a period followed by a space.

A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins in Area B of either the same line as the paragraph-name or the line immediately following. Successive sentences begin either in Area B of the same line as the preceding sentence or in Area B of the next line.

A sentence *consists of one or more statements followed by a period and a space.* When the sentences of a paragraph require more than one line, they may be continued on successive lines.

DATA DIVISION Entries. Each DATA DIVISION entry begins with a level indicator or a level-number followed by at least one space, the name of a data item, and a sequence of independent clauses describing the data item. Each clause, except the last clause of an entry, may be terminated by a semicolon 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 that begin with a level indicator and those that begin with a level-number.

FD is a level indicator. In DATA DIVISION entries that begin with a level indicator, the level indicator begins in Area A, followed by its associated file-name and appropriate descriptive information in Area B.

DATA DIVISION entries that begin with level-numbers are called Data Description entries. A level-number may be one of the following set: 1 through 15, 77. Level-numbers less than 10 are written either as a single digit or as zero followed by a digit. At least one space must separate a level-number from the word succeeding it. In DATA DIVISION entries that begin with a Data Description entry, the first Data Description entry starts with a level-number in Area A, followed by the descriptive information in Area B.

CHAPTER 2

COBOL INPUT/OUTPUT PROCESSING

COBOL FILES

M6800 ANS COBOL supports all file organizations, record formats, and access methods provided by the file management system.

File Organization

There are two types of file organization: indexed and sequential.

Indexed File Organization

Indexed files are those in which each record is associated with an identifying key. Indexed files may be accessed directly or sequentially; however, they must be assigned to input/output devices capable of direct access. Indexed file organization is indicated in the COBOL language by the statement *ORGANIZATION IS INDEXED* in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

Sequential File Organization

A sequential file is one whose records are organized in a consecutive manner. There is no identifying key associated with each record; therefore, records can be accessed sequentially only. Consecutive files may be assigned to any type of input/output device. Consecutive file organization is indicated when *ORGANIZATION IS SEQUENTIAL* is written or when the ORGANIZATION clause is omitted altogether.

File Access

The three methods of accessing files are sequential, random, and dynamic.

Sequential Access

Sequential access is the technique of referencing records serially within a file. The order in which records are read or written is determined implicitly by relative physical position within the file. This access method is specified by the ACCESS MODE IS SEQUENTIAL clause or is implied by the omission of that clause.

Random Access

Random access is the technique of reading and writing records of a file in an order dictated by the programmer. It may only be used with ORGANIZATION IS INDEXED files. The record to be referenced is indicated by the value of a key at the time that the input/output command is issued. This access method is specified by the ACCESS MODE IS RANDOM clause. The RECORD KEY clause specifies the key.

Dynamic Access

Dynamic access mode allows the file to be accessed either sequentially or randomly depending upon the I/O statement. It may only be used with files having ORGANIZATION INDEXED. This access mode is specified by the ACCESS IS DYNAMIC clause. The RECORD KEY clause is also required.

File-Handling Methods

A file-handling method is the effect of the combination of access technique, file organization, and the manner in which the file is opened.

Sequential Access

1. *OPEN OUTPUT. This combination creates a consecutive file. The new records replace any previous contents of the file.*
2. *OPEN EXTEND. New records will be added to the end of a consecutive file.*
3. *OPEN INPUT. If the file organization is consecutive, READ statements obtained records serially in the order in which they were originally written. If the file organization is indexed, READ statements obtain records serially in key value order (not necessarily in the order in which they were written).*

Random Access

1. *OPEN OUTPUT. This combination creates an indexed file. A RECORD KEY MUST be specified and its contents consulted upon each WRITE statement.*
2. *OPEN INPUT. Organization of the file must be indexed. A RECORD KEY must be specified and the contents consulted for each READ statement to locate the desired record within the file.*
3. *OPEN INPUT-OUTPUT. The sole essential difference between OPEN INPUT and OPEN INPUT-OUTPUT is that the latter permits the file to be updated instead of merely referenced; thus, WRITE statements are allowed to address the file.*

Input/Output Processing Summary

Table 2-1 summarizes the COBOL language file manipulation statements. *Each file must be named in an ENVIRONMENT DIVISION SELECT sentence and defined by an FD entry in the DATA DIVISION.* Each of the language elements concerned is described fully in succeeding chapters of this manual.

TABLE 2-1. File Manipulation Statements

File Organization	ACCESS MODE IS	Type of OPEN STATEMENT	PERMISSIBLE I/O Statement	RECORD KEY Required
Sequential	SEQUENTIAL (or unspecified)	INPUT	READ ... AT END	No
		OUTPUT EXTEND	WRITE ... [{BEFORE} {AFTER}] [ADVANCING] WRITE ...	No No
Indexed	SEQUENTIAL (or unspecified)	INPUT	START ... INVALID KEY READ ... AT END	Yes
		OUTPUT	WRITE ... INVALID KEY	Yes
		I-O	START ... VALID KEY READ ... AT END WRITE ... INVALID KEY REWRITE ... INVALID KEY DELETE ... INVALID KEY	Yes
Indexed	RANDOM	INPUT	READ ... INVALID KEY	Yes
		OUTPUT	WRITE ... INVALID KEY	Yes
		I-O	READ ... INVALID KEY WRITE ... INVALID KEY REWRITE ... INVALID KEY DELETE ... INVALID KEY	Yes
Indexed	DYNAMIC	INPUT	START ... INVALID KEY READ ... INVALID KEY READ NEXT ... AT END	Yes
		OUTPUT	WRITE ... INVALID KEY	Yes
		I-O	START ... INVALID KEY READ ... INVALID KEY READ NEXT ... AT END WRITE ... INVALID KEY REWRITE ... INVALID KEY DELETE ... INVALID KEY	Yes

CHAPTER 3

IDENTIFICATION DIVISION

GENERAL DESCRIPTION

The format of the IDENTIFICATION DIVISION is:

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

[AUTHOR. comment-sentences.]

[INSTALLATION. comment-sentences.]

[DATE-WRITTEN. comment-sentences.]

[DATE-COMPILED. comment-sentences.]

[SECURITY. comment-sentences.]

[REMARKS. comment-sentences.]

The IDENTIFICATION DIVISION specifies information essential to identification such as the name of the program, the date the program was written, programmer's name, security, etc. The listing contains all information specified in this division, but the specified information in no way affects the object program. Allowable information is presented in seven separate paragraphs: one mandatory, the others optional. If the optional paragraphs are included in the program, they must be in the order indicated above.

ORGANIZATION

The IDENTIFICATION DIVISION header is always the first line in a source program and appears as shown above, including the punctuation. This header and the fixed paragraph-name(s) must conform to COBOL Coding Sheet specifications. Only the PROGRAM-ID paragraph is mandatory; all others are optional. Comment-sentences for the optional paragraphs consist of any sentence or group of sentences.

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph must always appear as the first paragraph in the IDENTIFICATION DIVISION. This paragraph permits the programmer to declare the name of the source program.

DATE-COMPILED Paragraph

The DATE-COMPILED paragraph should be used to provide the compilation date in the source program listing.

Example:

The IDENTIFICATION DIVISION of a typical program might be written
IDENTIFICATION DIVISION.

PROGRAM-ID. Inventory.

AUTHOR. John Smith.

DATE-WRITTEN. October 15, 1977.

DATE-COMPILED. November 1, 1977.

REMARKS. This program prints the inventory report.

CHAPTER 4

ENVIRONMENT DIVISION

GENERAL DESCRIPTION

The format of the ENVIRONMENT DIVISION is:

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. source-computer entry.

OBJECT-COMPUTER. object-computer entry.

INPUT-OUTPUT SECTION.

FILE-CONTROL. file-control entry.

[I-O-CONTROL. input/output control entry.]

The ENVIRONMENT DIVISION describes those aspects of the data processing program that depend on the physical characteristics of a specific computer. The information presented in this division enables the compiler to link the operations indicated in the DATA and PROCEDURE DIVISIONS to the physical aspects of computer hardware and the executive system that is to execute the object program. Thus, the ENVIRONMENT DIVISION is entirely computer-oriented.

The ENVIRONMENT DIVISION is divided into the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

The CONFIGURATION SECTION deals with the characteristics of the computing system on which the source program is to be compiled and on which the object program is to operate. This section is divided into two paragraphs: the SOURCE-COMPUTER paragraph describing the computer on which the COBOL compiler is to run and the OBJECT-COMPUTER paragraph defining the computer on which the translated program is to run.

The INPUT-OUTPUT SECTION provides information needed to control transmission and handling of data between external media and the object program. There are two fixed paragraph-names in this section: the FILE-CONTROL paragraph naming and associating the files with external media, and the I-O-CONTROL paragraph specifying certain other file information.

CONFIGURATION SECTION

SOURCE-COMPUTER Paragraph

The formats of this paragraph are:

Format 1

SOURCE-COMPUTER. copy-statement.

Format 2

SOURCE-COMPUTER. computer-name.

The SOURCE-COMPUTER paragraph enables the programmer to describe to the compiler the computing system on which source program translation is to take place. Format 1 is used when the COBOL library contains the entire description of the SOURCE-COMPUTER configuration. See Chapter 7 for a complete description of the COBOL library.

OBJECT-COMPUTER Paragraph

The formats of this paragraph are:

Format 1

OBJECT-COMPUTER. copy-statement.

Format 2

OBJECT-COMPUTER

computer-name [MEMORY SIZE integer CHARACTERS]

Format 1 is used when the COBOL library contains the entire description of the OBJECT-COMPUTER configuration.

The contents of the OBJECT-COMPUTER paragraph, as with the entire contents of the SOURCE-COMPUTER paragraph, is not significant to the compiler and is treated as commentary.

INPUT-OUTPUT SECTION

FILE-CONTROL Paragraph

The formats of this paragraph are:

Format 1

FILE-CONTROL. copy statement.

Format 2

FILE-CONTROL

SELECT file-name-1 [ASSIGN-clause] [ORGANIZATION-clause]
[ACCESS-clause] [RECORD KEY-clause]. . . .

Format 1 is used when the complete FILE-CONTROL paragraph description desired exists in an element in the current COBOL source library. For additional information see "COBOL Library." A discussion of format 2 follows.

SELECT Sentence

Each file defined in the FILE SECTION of the DATA DIVISION must be named once and only once as file-name-1 in a SELECT sentence. Each select file must have a File Description entry in the DATA DIVISION.

The following clauses that compose the SELECT sentence are all optional; except for the ASSIGN clause, they may be written in any order.

ASSIGN Clause.

The format of this required clause is
ASSIGN TO implementor-name-1]

The ASSIGN clause permits a file to be associated with a particular type of hardware device.

Acceptable implementor-names are:
PRINTER
DISK diskid:number

Where: diskid—represents an eight-character disk file identification.
number—represents the file number suffix for the diskid.

Refer to the COBOL operations reference manual for an explanation of the meaning of diskid:number as related to different disk types.

ORGANIZATION Clause.

The format of this clause is:

$$\left[\text{ORGANIZATION IS } \left\{ \begin{array}{l} \text{SEQUENTIAL} \\ \text{INDEXED} \end{array} \right\} \right]$$

SEQUENTIAL denotes that the file is a sequential file.

INDEXED denotes that the file has an indexed organization. It does not necessarily imply that file is to be accessed randomly.

Sequential organization is implied when this clause is omitted.

ACCESS Clause.

The format of this clause is:

$$\left[\text{ACCESS MODE IS } \left\{ \begin{array}{l} \text{SEQUENTIAL} \\ \text{RANDOM} \\ \text{DYNAMIC} \end{array} \right\} \right]$$

SEQUENTIAL denotes that records are obtained or placed sequentially: that is, the next logical record is available from the file on a READ statement execution, or a specific logical record is placed in the next position in the file on a WRITE statement execution.

If RANDOM or DYNAMIC is specified, the RECORD KEY clause (see below) must also be specified, and the file must be assigned to a direct-access device. In this case, the specified logical record (located using RECORD 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 (located using RECORD KEY data-name contents) on a WRITE statement execution. DYNAMIC access mode differs from RANDOM access mode in that the file may be accessed sequentially or randomly, depending on the I/O statement. That is, after a record is located by a random read, the records following it can be read sequentially. Another random read can then be issued to switch back to random access.

Sequential access is assumed when these clauses are omitted.

RECORD KEY clause.

The format of this clause is:

$$\left[\text{RECORD KEY IS data-name } [\text{WITH DUPLICATES}] \right]$$

The RECORD KEY clause must be specified if INDEXED organization is specified; it is not meaningful to SEQUENTIAL organization. Data-name must be contained within the record. In addition, it must conform to the rules for the file management system outlined in the COBOL operations reference manual.

The contents of data-name are used by the READ, and WRITE statements to locate a specific record in a mass storage file. The symbolic identity of the record to be read or written must be placed in data-name before the appropriate input/output statement is executed.

The optional WITH DUPLICATES clause specifies that records with duplicate keys are to be permitted in the file.

I-O-CONTROL Paragraph

The formats of this paragraph are:

Format 1

I-O-CONTROL. copy-statement.

Format 2

I-O-CONTROL. [SAME AREA-clause]

Format 1 causes the library element to be retrieved from the current COBOL source library and inserted into the source program at this point. A discussion of Format 2 follows. For additional information see "COBOL Library."

SAME AREA Clause

The format of this clause is:

[SAME AREA FOR file-name-1 [,file-name-2]

When SAME AREA is written, the data areas for all of the files mentioned overlap. Thus, only one of the list of files may be open at the same time. More than one SAME AREA clause may appear in a COBOL program, but no one file-name may appear in more than one such clause.

CHAPTER 5

DATA DIVISION

GENERAL DESCRIPTION

The DATA DIVISION describes data that the object program accepts as input in order to manipulate, create, or produce output. Data to be processed fall into three categories:

1. Data that is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
2. Data that is developed internally and placed into intermediate or working storage, or into specific format for output reporting purposes.
3. Constants that are defined by the use.

PHYSICAL AND LOGICAL ASPECTS OF DATA DESCRIPTION

DATA DIVISION Organization

The DATA DIVISION is subdivided into the FILE and WORKING-STORAGE 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 that are not part of external data files but are developed and processed internally.

DATA DIVISION Structure

The DATA DIVISION is identified by and must begin with the header
DATA DIVISION.

Each of the sections of the DATA DIVISION (except the WORKING-STORAGE SECTION) is optional and may be omitted from the source program. The fixed names of these sections in their required order of appearance as section headers in the DATA DIVISION are

FILE SECTION.

WORKING-STORAGE SECTION.

Section headers for the FILE SECTION are followed by one or more sets of entries composed of file clauses, followed by associated Record Description entries. WORKING-STORAGE SECTION headers are followed by Data Description entries for noncontiguous items, followed by Record Description entries. See Figure 1.

File Section

In a COBOL program the File Description (FD) entry represents the highest level of organization in the FILE SECTION. The FILE SECTION is composed of the section header FILE SECTION and a period, followed by a File Description entry consisting of a level indicator (FD), a data-name, and a series of independent clauses. These clauses specify the size of the physical records, and the names of the data records and reports that compose the file. The entry itself is terminated by a period.

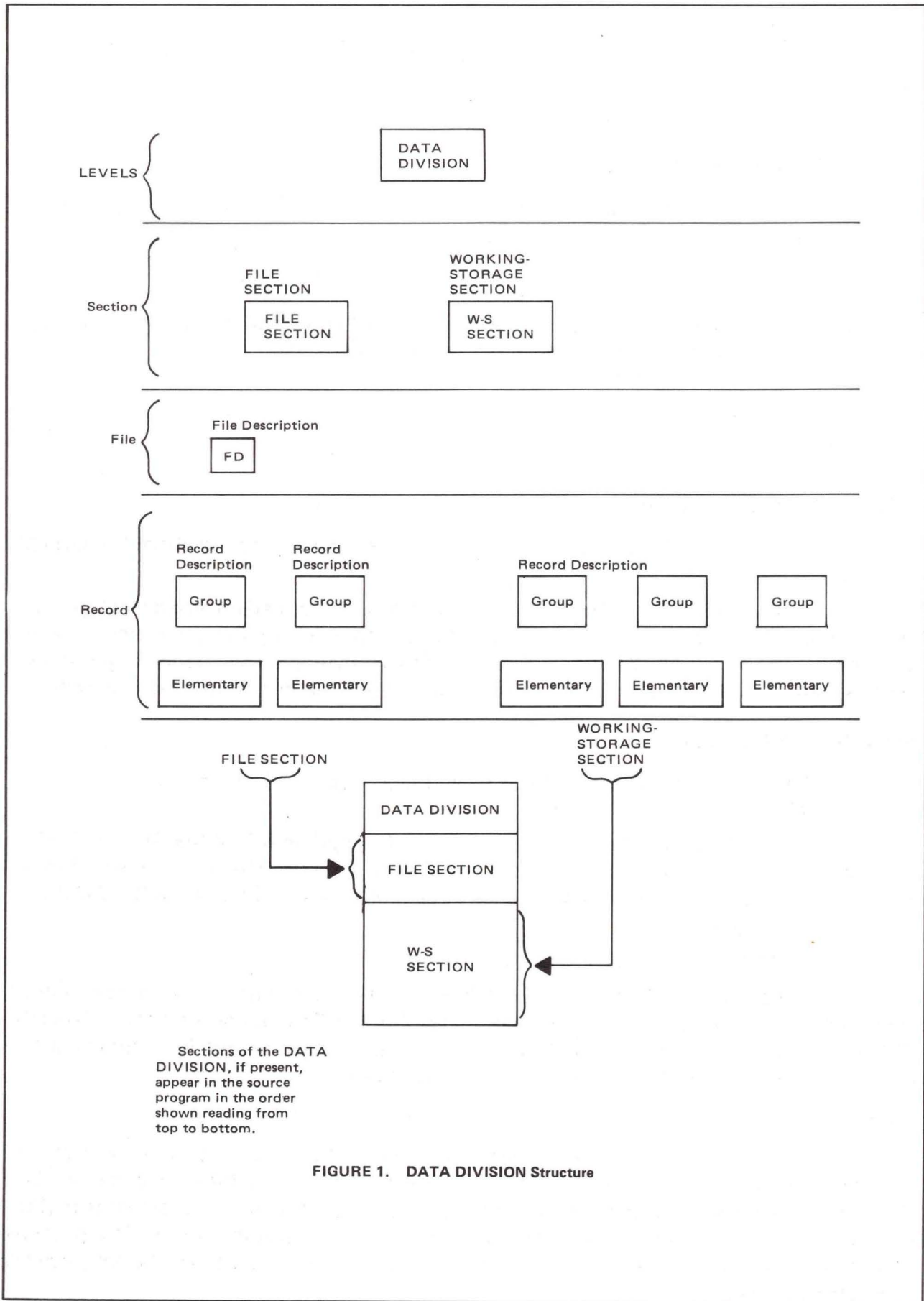


FIGURE 1. DATA DIVISION Structure

Record Description Structure. A record description consists of a set of Data Description entries that describe the characteristics of a particular record. Each Data Description entry consists of a level-number followed by a data-name, followed by a series of independent clauses, as required. A record description has a hierarchical structure; 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 "Concepts of Levels" in Chapter 1; elements allowed in a record description are specified in "Data Description Entries" later in this chapter.

Working-Storage Section

The WORKING-STORAGE SECTION is composed of the section header WORKING-STORAGE SECTION and a period, followed by Data Description entries for noncontiguous working-storage items and Record Description entries (in that order).

Noncontiguous Working-Storage. Items in working-storage that bear no relationship to one another need not be grouped into 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 that begins with the special level-number 77.

Data clauses required in each Data Description entry are

1. Level-number.
2. Data-name.
3. The PICTURE clause.

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

Working-Storage Records. Data elements in working-storage that bear a definite relationship to one another must be grouped into records according to the rules for formation of record description. All clauses that are used in normal input or output record descriptions can be used in a working-storage record description.

Initial Values. The initial value of any item in the WORKING-STORAGE SECTION except an index data item is specified by using the VALUE clause of the record description. The initial value of any index data item is determined at compile time.

File Description—Complete Entry Skeleton

The general formats of this entry are:

Format 1

FD file-name copy-statement.

Format 2

FD file-name

[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

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

[DATA { RECORD IS } data-name-7 [data-name-8] . . .]
 { RECORDS ARE }

Format 3

FD file-name
[LINAGE IS lines-on-page]
[TOP IS top-margin]
[BOTTOM IS bottom-margin]

The File Description entry furnishes information concerning the physical structure, identification, and record names pertaining to a given file. In Format 1 the COPY clause enables a prewritten File Description entry to be included in the DATA DIVISION; this entry is contained in the COBOL library. For additional information see Chapter 7, "COBOL Library." A description of Format 2 follows.

RECORD CONTAINS Clause.

The format of this clause is:

[RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

The RECORD CONTAINS clause specifies the size of data records. Since the size of each data record is completely defined within the Record Description entry, this clause is not required.

If only "integer-4" is specified, it represents the exact number of characters in the data record. If both "integer-3" and "integer-4" are specified, 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.

LABEL RECORDS Clause.

The format of this clause is:

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

Since all file labels are internal to the M6800 file management system, this clause is not required in M6800 COBOL and is treated as a comment entry.

The OMITTED option specifies that no explicit labels exist for the file or the device to which the file is assigned.

The STANDARD option specifies that standard system labels exist for the file or the device to which the file is assigned. Such labels are written when the file is opened for output and checked automatically by the operating system when the file is opened for input or input/output.

DATA RECORDS Clause.

The format of this clause is

[DATA { RECORD IS } data-name-7 [data-name-8] . . .]
 { RECORDS ARE }

The DATA RECORDS clause cross-references the description of data records with their associated file description. Each logical record in the file may be named in this clause; the order of listing the names is not significant. Since the record names are available following the FD description, this clause is not required.

The appearance of multiple data-names means that the file contains a corresponding number of different types of records. These records may be of differing sizes and formats. The order in which they are listed in the clause is not important. It must be remembered that no two records of the same file are available for processing at the same time; in other words, if one record is read from a file and then another record is read from the same file, the second record replaces the first.

Format 3 of the FD clause is used with print files.

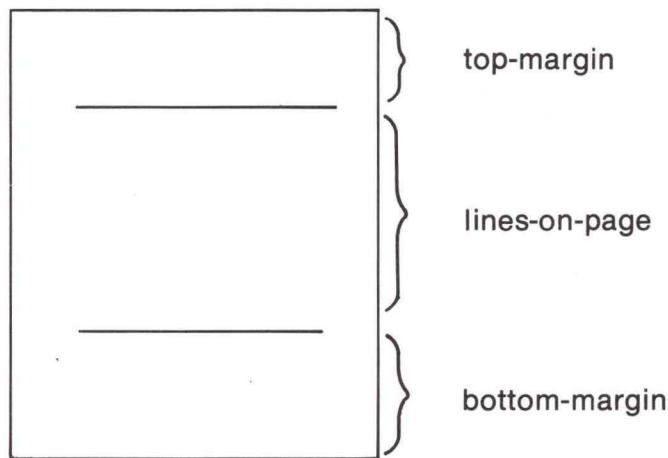
FD file-name

[LINAGE IS lines-on-page]

[TOP IS top-margin]

[BOTTOM IS bottom-margin]

If these clauses are not specified for a print file, the default is 3 lines at the top, 69 body lines, and 3 lines at the bottom. The clauses have the following meaning:



Example:

```
FD PRINT-FILE
  LINAGE IS 60,
  TOP IS 3,
  BOTTOM IS 3.
```

The values for top-margin and bottom-margin may be zero. The value for lines-on-page may *not* be zero. The special register LINAGE-COUNTER may be used to reference the current line number within lines-on-page during execution.

Example:

```
IF LINAGE-COUNTER EQUALS 60 THEN PERFORM NEW-PAGE.
```

Data Description Entries

General Format:

```
Level-number { data-name } [REDEFINES-clause] [COPY statement]
              FILLER
              [PICTURE-clause] [USAGE-clause]
              [BLANK-clause] [JUSTIFIED-clause]
              [VALUE-clause] [OCCURS-clause]
```

A Data Description entry (see Figure 2) describes characteristics of each item within a data record. Each item is accorded a separate entry that must appear in the order in which the item occurs in the record, since the relative location of each entry is communicated to the compiler by its position in the record description. Each entry consists of a level-number, data-name, and series of clauses terminated by a period.

The reserved word FILLER may be substituted for a programmer-defined data-name when an unused portion of a logical record or data item that is not referenced directly is defined.

Specific formats for individual types of data items are shown below. In each of these formats, clauses that do not appear are categorically forbidden in that data type, while clauses that are mandatory are depicted without brackets.

Detailed Formats of Data Items:

Group Item

level-number { data-name } [REDEFINES-clause] [OCCURS-clause]
 { FILLER }
 [USAGE-clause]
 [VALUE is non-numeric-literal].

Example

```
01 GROUP-ITEM.
02 FIELD01 PICTURE X.
02 FIELD-2 PICTURE X.
```

FIGURE 2. Various Data Description Entries Listing

```
01 VARIOUS-DATA-DESC.
02 ALPHABETIC-TYPES.
03 A1 PICTURE AAAAAAAAA.
03 A2 REDEFINES A1 PICTURE A(8).
03 A3 PICTURE A(4) OCCURS 4 TIMES.
03 A4 PICTURE A(6) VALUE IS 'XYZ A'.
03 A5 PICTURE A(2) USAGE IS DISPLAY.
03 A6 PICTURE A(8).
03 A7 REDEFINES A6 PICTURE A(2) USAGE DISPLAY
      OCCURS 4 TIMES.
02 ALPHANUMERIC-TYPES REDEFINES ALPHABETIC-TYPES.
03 AN1 OCCURS 8 TIMES PICTURE IS X9A.
03 AN2 PICTURE X(16) USAGE IS DISPLAY.
03 AN3 REDEFINES AN2 PICTURE X(4) OCCURS 4 TIMES.
02 ALPHA-EDITED-TYPES.
03 AE1 PICTURE XXBXXBXX.
03 AE2 PIC IS XXXXBXX99B00BXXX.
03 AE3 REDEFINES AE2 PIC X(10)B09AAX DISPLAY.
02 NUMERIC-EDITED-TYPES.
03 NE1 PICTURE IS ZZ,999+.
03 NE2 REDEFINES NE1 PICTURE **,**9-.
03 NE3 OCCURS 4 TIMES PICTURE ZZZ9.
02 NUMERIC-TYPE.
03 N1 PICTURE 9999 OCCURS 5 TIMES USAGE DISPLAY.
03 N2 PIC S9999 VALUE IS-1234.
03 N3 REDEFINES N2 PICTURE S99V99.
```

Alphanumeric Elementary Item

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

{PICTURE
PIC} IS on-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal] [{JUSTIFIED
JUST} RIGHT]

Example:

02 CUST-NAME PICTURE X (21) DISPLAY
02 CUST-ADR PIC X (45)

Alphanumeric Edited Elementary Item

Level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]

{PICTURE
PIC} IS ae-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal] [{JUSTIFIED
JUST} RIGHT]

Example:

02 DATE PICTURE XXBXXXBXXXX VALUE '15 DEC 1977'.

Numeric Edited Elementary Item

Level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]

{PICTURE
PIC} IS { numeric-type BLANK WHEN ZERO
ne-type BLANK WHEN ZERO }

[USAGE IS DISPLAY].

Example:

02 DEPT-NO PIC ZZ999.
02 GROSS-SALES PICTURE \$Z, ZZZ,ZZZ,ZZZ.99-.

Alphabetic Elementary Item

Level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]

{PICTURE
PIC} IS alpha-type [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal]

Example:

02 COUNTY-NAME PICTURE A(35) USAGE IS DISPLAY.

ASCII Decimal Elementary Item

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

{ PICTURE } IS numeric-type [USAGE IS DISPLAY]
 PIC }

[VALUE IS numeric-literal].

Example:

02 COST PIC 999V99 VALUE 10.39.

Packed Decimal Elementary Item

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

{ PICTURE } IS numeric-type USAGE IS { COMPUTATIONAL }
 PIC } COMP }

[VALUE IS numeric-literal].

Example:

02 TOTAL-RECORDS PIC 9(4) COMPUTATIONAL.

Index Item

77 index-name USAGE IS INDEX.

Example:

77 X1 INDEX

REDEFINES Clause.

The format of this clause is:

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

The REDEFINES clause overlaps items in storage (allocates the same storage space for different items at different times) or provides an alternate grouping or description of the same data (redefines an elementary item or a group item).

The level-numbers of data-name-1 and data-name-2 must be identical.

The REDEFINES clause is not used at the record 01 level in the FILE SECTION. The DATA RECORDS clause in the FD entry indicates the existence of more than one type of record; thus, an implied redefinition exists at the 01 level.

Redefinition begins at data-name-2 and continues until a level-number whose value is equal to or less than data-name-2 is encountered; therefore, between data-names-1 and -2 there must not be a level-number lower than that of data-names-1 and -2. Data-name-1 must follow data-name-2 such that, if data-name-2 is a group entry, the entry for data-name-1 must appear immediately after the entries for all items in that group. However, additional entries that redefine the same area may intervene.

Data-name-1 may be a group or an elementary item irrespective of the nature of the data-name-2 item. If it is a group, the data-name-2 entry is followed by all the entries in that group, since such entries are part of the redefinition; if it is an elementary item, it completely redefines data-name-2. A REDEFINES clause may be specified for an item within the scope of

an area being redefined; that is, REDEFINES clauses may be specified for items subordinate to items containing REDEFINES clauses.

When the REDEFINES clause is used with certain other clauses, entries (except for condition-name entries) containing or subordinate to the REDEFINES clause must not contain VALUE clauses.

When an area is redefined, all descriptions of that area remain in effect for the entire program. The one that is selected depends on the particular reference made to the area. For example, if items A and B share the same area, MOVE X TO A moves X to the area according to the description of A, MOVE Y TO B moves Y to the same area according to the description of B. These statements could be executed anywhere in a program; final contents of the area depend on the order in which they are executed. A table of constant items is redefined so that any item in the table can be referenced by position rather than by individual name. This does not redefine the area according to different patterns, but simply permits the same pattern of items to be considered in a different way.

COPY Statement.

The format of this clause is:

level-number data-name-1 [REDEFINES data-name-2] copy statement.

The COPY statement enables prewritten Record Description entries to be included in the DATA DIVISION. These entries are from the COBOL library, eliminating the need for specifying the entries each time they are needed. Information being copied is inserted at the point in an entry where the COPY statement appears; thus data-name-1 and its level-number are not replaced by the information being copied, nor is the REDEFINES clause if it is present.

For additional information see Chapter 7, "COBOL Library."

PICTURE Clause.

The format of this clause is

$\left. \begin{array}{l} \text{PICTURE} \\ \text{PIC} \end{array} \right\}$ IS character-string

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

The character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. These allowable combinations determine the category of the item. The five categories of data that can be described with a PICTURE clause are

1. Alphabetic
2. Alphanumeric
3. Numeric
4. Alphanumeric Edited
5. Numeric Edited

The following rules apply to use of the PICTURE clause.

1. GENERAL

The number of occurrences of any of the characters indicates the size of an item described by the PICTURE clause. Size may be indicated either by repeating the character or, in a shorthand way, by writing the character once and putting the number of its occurrences in parentheses. Thus, Z (10)9(2) is equivalent to ZZZZZZZZZZ99.

A maximum of 30 characters is allowed in a PICTURE clause. This limit does not refer to the number of characters in the item itself, but only to the number of characters (including parentheses) used in the PICTURE specifying the item. For example, the same item may be described by a PICTURE containing 12 characters, ZZZZZZZZZZ99, or by a PICTURE containing only 9 characters, Z(10)9(2). In either case, the actual size of the item is 12 characters. An item containing 75 alphabetic characters may be specified by the PICTURE A(75), which uses only 5 characters, but the same item may not be specified by a PICTURE in which A is repeated 75 times. The size of an alphabetic or alphanumeric item described by the PICTURE is limited to a maximum of 255 characters except for numeric display items, which are limited to 15 digits. The size of an entire Group Item is also limited to 4095 characters.

2. Categories of Data

a. Alphabetic (alpha-type)

The PICTURE of an alphabetic item contains only the character A. The number of A's in the character-string denotes the size of the data item, and each A represents one character that at execution time may contain one of the twenty-six letters of the English alphabet or the space character.

b. Alphanumeric (an-type)

The PICTURE of an alphanumeric item may contain only the Character X or a combination of the characters X, A, and 9. An X indicates that the corresponding character position of the data item may contain any one of the characters in the ASCII set. When the PICTURE is described with a combination of characters, each character is treated as though it were an X, since no examination of the data placed in the item is made at execution time. Thus, this type of PICTURE description may have documentary significance only to the programmer.

c. Numeric (numeric-type)

The PICTURE of a numeric data item may contain only the characters 9, S, and V. The character 9 represents a digit position containing a numeral and is counted in the size of the item.

The character S indicates the presence of an operational sign and must be written as the leftmost character in the PICTURE.

The character V indicates the position of the assumed decimal point and may occur only once in the character-string. The V does not represent a digit position and therefore is not counted in the size of the item. When a V is written as the last (rightmost) character in the PICTURE, it is redundant.

d. Alphanumeric Edited (ae-type)

The PICTURE of an alphanumeric edited item contains any combination of the characters X, A, and 9 together with one or more occurrences of the insertion characters 0 (zero) or B. Each 0 represents a character position into which the character 0 is to be inserted; each B represents a character position into which the space character is to be inserted. Thus, an alphanumeric edited field is one that contains certain character positions into which insertion characters are forced whenever data is stored in the item at execution time.

e. Numeric Edited (ne-type)

Editing alters the format and punctuation of data in an item; characters can be suppressed or added. Editing is accomplished by moving a data item to an item described as containing editing symbols. Movement may be direct or indirect: The programmer can specify a MOVE statement or arithmetic statement in which the result of computation is stored in such an item.

Characters that may be used in a PICTURE of a numeric edited item are
9 V \$ + - . , 0 B / CR DB Z *

The characters 9 and V are discussed above; their use is exactly the same as in numeric items. The remainder are insertion and replacement characters (see below).

3. Insertion Characters

When an insertion character is specified in the PICTURE, it appears in the edited data item; therefore, the size of the item must reflect these additional characters. Insertion characters and their characteristics are:

- \$ When a single dollar sign is specified as the leftmost symbol, it appears as the leftmost character in the size of the item.
- + When a plus sign is specified as the first or last symbol, a plus sign is inserted in the indicated character position of the edited data item provided the data is positive (contains a positive operational sign) or is unsigned. If the data is negative, a minus sign is inserted in the indicated character position. This sign is counted in the size of the item.
- When a minus sign is specified as the first or last symbol, a minus sign is inserted in the indicated character position of the edited data item provided the data is negative (contains a negative operational sign). If the data is not negative, a blank is inserted in the indicated character position. This sign or blank is counted in the size of the item.
- .
- The period character represents an actual decimal point as differentiated from an assumed decimal point. When used, a decimal point appears in the edited data item as a character in the indicated character position; therefore, the decimal point is counted in the size of the item. A PICTURE can never contain more than one decimal point, actual or assumed.
- ,
- When a comma is used, a comma is inserted in the corresponding character position of the edited data item. It is counted in the size of the item.
- 0 When a zero is used, a zero is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.
- B When a character B is used, a space is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.
- / When the slash character is used, a slash character is inserted in the corresponding character position in the edited data item. It is counted in the size of the item.
- CR The credit symbol CR may be specified only at the right end of the PICTURE character-string. It is inserted in the last two character positions of the edited data item provided the value of the data is negative; if the data is positive or unsigned, these last two character positions are set to spaces. Since this symbol always results in two characters (CR or spaces), it is included as two characters in the size of the item.
- DB The debit symbol DB may be specified only at the right end of the PICTURE. It functions in the same manner as the credit symbol.

Examples of Insertion Characters:

Source Data	Editing PICTURE	Edited Item
4 8	\$99	\$ 4 8
4 8 ▲ 3 4	\$99.99	\$ 4 8 . 3 4
4 8 3 4	9,999	4 , 8 3 4
2 9 2	+999	+ 2 9 2
2 9 ⁺ 2	+999	+ 2 9 2
2 9 ⁻ 2	+999	- 2 9 2
2 9 ⁻ 2	999-	2 9 2 -
2 9 ⁺ 2	-999	- 2 9 2
2 9 2	999-	2 9 2 △
2 4 3 ▲ 2 1	\$BB999.99	\$ △△ 2 4 3 . 2 1
2 4 3 ▲ 2 1	\$00999.99	\$ 0 0 2 4 3 . 2 1
1 1 ▲ 3 ⁻ 4	99.99CR	1 1 . 3 4 C R
1 1 ▲ 3 4	99.99CR	1 1 . 3 4 △△
2 3 ▲ 7 6	99.99DB	2 3 . 7 6 D B
2 3 ▲ 7 ⁻ 6	99.99DB	2 3 . 7 6 △△
1 2 3 4 5 6	99/99/99	12/34/56

4. Replacement Characters

A replacement character suppresses leading zeros in data and replaces them with other characters in the edited data item. Only one replacement character may be used in a PICTURE, although Z or * may be used with any one of the insertion characters. Replacement characters and their characteristics are:

Z One character Z is specified at the left end of the PICTURE character string for each leading zero that is to be suppressed and replaced by blanks in the edited data item. Z's may be preceded by one of the insertion characters \$ + or - and interspersed with any of the . , 0 or B insertion characters.

Only the leading zeros that occupy a position specified by Z are suppressed and replaced with blanks. No zeros are suppressed to the right of the first non zero digit whether or not a Z is present, nor are any zeros to the right of an assumed or actual decimal point suppressed unless the value of the data is zero and all the character positions in the item are described by a Z. In this special case, even an actual decimal point is suppressed and the edited item consists of all blanks.

If a \$ + or - is present preceding the Z's, it is inserted in the far left character position of the item even if succeeding zeros in the item are suppressed. In the special case where the value of the data is zero and all the character positions following the \$ + or - are specified by Z's, the \$ + or - is replaced by a blank.

If an 0 or B or , in the PICTURE is encountered before zero suppression terminates, the character is not inserted in the edited data item but is suppressed, and a blank inserted in its place.

- * The asterisk replaces the leading zeros it edits by an asterisk instead of a blank. It is specified in the same way as the editing character Z and follows the same rules, except that an actual decimal point is never replaced.
- \$ When the dollar sign is used as a replacement character to suppress leading zeros, it acts as a floating dollar sign and is inserted directly preceding the first nonsuppressed character. One more dollar sign must be specified than the number of zeros to be suppressed. This dollar sign is always present in the edited data whether or not any zero suppression occurs. The remaining dollar signs act in the same way as Z to effect the suppression of leading zeros. No other editing character may precede the initial dollar sign. Each dollar sign specified in a PICTURE is counted in determining the size of the report item.
- + When a plus sign is used as a replacement character, it is a floating plus sign. The plus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating dollar sign: a plus sign is placed directly preceding the first nonsuppressed character if the edited data is positive or unsigned, and a minus sign is placed in this position if the edited data is negative.
- When a minus sign is used as a replacement character, it is a floating minus sign. The minus sign is specified one more time than the number of leading zeros to be suppressed. It functions in the same way as the floating plus sign, except that a blank is placed directly preceding the first nonsuppressed character if the edited data is positive or unsigned.

Examples of Replacement Characters:

Source Data	Editing PICTURE	Edited Item
0 0 9 2 3	ZZ999	△△9 2 3
0 0 9 2 3	ZZZ99	△△9 2 3
0 0 0 0 △ 0 0	ZZZZ.99	△△△△. 0 0
0 0 9 △ 2 3	\$***.99	\$ * * 9 . 2 3
0 0 0 8 △ 2 4	\$\$\$\$9.99	△△△\$ 8 . 2 4
0 0 5 △ 2 6	- - - 9.99	△△- 5 . 2 6
3 2 △ 6 5	\$\$\$99	\$ 3 2 . 6 5

Examples of PICTURE Editing:

DATA to be Edited	PICTURE of Report Item	Edited Item
0 1 2 3 4 5	ZZZ,999.99	△1 2 , 3 4 5 . 0 0
0 0 1 2 3 4	Z99,999.99	△0 0 , 0 1 2 . 3 4
0 0 0 1 2 3	\$ZZZ,ZZ9.99	\$△△△△△△1 . 2 3
0 0 0 0 1 2	\$ZZZ,ZZZ.99	\$△△△△△△△ . 1 2
0 0 1 2 3 4	\$**,**9.99	\$ * * 1 , 2 3 4 . 0 0
1 2 3 4 5 6	\$**,**9.99	\$1 2 3 , 4 5 6 . 0 0
1 2 3 4 5 6	\$**,**9.99	\$ * * * * * 1 . 2 3
0 0 0 0 1 2 ⁺	+999,999	+ 0 0 0 , 0 1 2
0 0 0 0 1 2 ⁻	-ZZZ,ZZZ	△△△△△1 2
1 2 3 4 5 6 ⁻	\$ZZZ,ZZ9.99CR	\$1 2 3 , 4 5 6 . 0 0 C R
0 0 0 1 2 3 ⁺	\$ZZZ,ZZ9.99DB	\$△△△△△△1 . 2 3
0 0 1 2 3 4	\$(4),\$9.99	△△△△ \$1 2 3 . 4 0
0 0 0 0 0 0	\$(4),\$9.99	△△△△△△△\$. 0 0
0 0 0 0 1 2 ⁻	-----,----.99	△△△△△△△- . 1 2
0 0 0 0 1 2 ⁺	-----,----.99	△△△△△△△△. 1 2
0 0 0 0 0 1	\$\$\$\$,\$ZZ.99	Illegal PICTURE

5. Summary

- Only one of the characters of the set Z * \$ + and - can be used within a single PICTURE as a replacement character, although it may be specified more than once.
- If one of the replacement characters Z or * is used with one of the insertion characters \$ + or -, the plus or minus signs may be specified as either the leftmost or rightmost character in the PICTURE.
- A plus sign and a minus sign may not be included in the same PICTURE.
- A leftmost plus sign and a dollar sign may not be included in the same PICTURE.
- A leftmost minus sign and a dollar sign may not be included in the same PICTURE.
- The character 9 may not be specified to the left of a replacement character.
- Symbols that may appear only once are V S . CR and DB.
- The decimal point may not be the rightmost character in a PICTURE.

USAGE Clause

The format of this clause is:

USAGE IS { DISPLAY
[COMPUTATIONAL]
COMP
INDEX }

The USAGE clause specifies the form in which data is represented in the computer. It 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; in addition, 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 computer. 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 referent operands.

DISPLAY denotes that the item is carried in the ASCII format. DISPLAY mode is assumed when a USAGE clause is not written. One character is stored in each byte of the item; if the item is numeric, the leftmost byte can contain an operational sign in addition to a digit.

COMPUTATIONAL defines a packed decimal data item whose length is specified by the accompanying PICTURE clause.

INDEX defines an item that is called an index data item and will contain a value that corresponds to an occurrence number of a table element. Index data items must be elementary data items. Since USAGE IS INDEX totally defines the internal representation of the data, a PICTURE clause is not used with an index data item.

BLANK WHEN ZERO Clause.

The format of this clause is:

BLANK WHEN ZERO

The BLANK WHEN ZERO clause may be supplied only in conjunction with a numeric edited item. It specifies that when the source item has a value of zero, the edited data item is to contain all spaces.

JUSTIFIED Clause.

The format of this clause is:

{ JUSTIFIED
JUST } RIGHT

This clause is applicable only to alphabetic or alphanumeric items. Normally, when data is moved into an alphabetic or alphanumeric field, the source data is aligned at the leftmost character position of the receiving data item and moved with space fill or truncation on 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 leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and is larger than the sending data item, the data are aligned at the rightmost character position in the data item with other characters space-filled.

VALUE Clause.

The format of this clause is:

Value IS literal

The VALUE clause defines the value of constants, or the initial value of working-storage items. This clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the form. The following rules apply:

1. General

- a. If the category of the item is numeric, the literal in the VALUE clause must be a numeric literal. The literal is aligned according to the alignment rules except that the literal must not have a value requiring truncation of digits.
- b. If the category of the item is alphabetic or alphanumeric the literal in the VALUE clause must be a nonnumeric literal. The literal is aligned according to the alignment rules except that the number of characters in the literal must not exceed the size of the item.
- c. The numeric literal in a VALUE clause of an item must have a value within the range of values indicated by the USAGE or PICTURE clause.
- d. The function of any editing clauses or editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item.

2. Data Description Entries

- a. Rules governing the use of the VALUE clause differ with the respective section of the DATA DIVISION:
 - (1) In the FILE SECTION, the VALUE clause is not allowed.
 - (2) In the WORKING-STORAGE the VALUE clause may 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 description, the initial value may be unpredictable.
- b. The VALUE clause must not be stated in a Record Description entry containing an OCCURS clause or in an entry subordinate to an entry containing an OCCURS clause.
- c. The VALUE clause must not be stated in a Record Description entry containing a REDEFINES clause or in an entry subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.
- d. The VALUE clause may not be used in an entry at the group level.

OCCURS Clause.

The format of this clause is:

OCCURS integer-1 TIMES

[INDEXED BY index-name-1 [,index-name-2] . . .]

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

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data; when it is used, the data-name that is the subject of this entry must either be subscripted whenever it is referenced in a statement. Furthermore, if the subject of this entry is the name of a group item, all data-names belonging to the group must be subscripted whenever they are used as operands.

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

An INDEXED BY clause is required if the subject of this entry, or an item within it if it is a group item, is to be referenced by indexing. The index-name identified by this clause is not defined elsewhere; the compiler allocates storage for it unassociated with any data hierarchy.

CRT Form Descriptions

In order to facilitate form layout for the CRT, two special clauses, "LINE" and "COLUMN," have been provided for use with CRT data description sentences. These clauses are allowed only with CRT data description sentences and are written between the "PICTURE" and "VALUE" clauses of the CRT data description sentence. The "LINE" and "COLUMN" clauses are entirely optional. However, if they are not used, the programmer must provide all necessary CRT control codes using hexadecimal constants when writing to the CRT.

LINE IS clause

The format of this clause is:

$$\left[\text{;LINE is } \left\{ \begin{array}{l} \text{integer-1} \\ \text{NEXT PAGE} \end{array} \right\} \right]$$

The "LINE" clause indicates the line number on which the data item is to be displayed.

When the LINE clause specifies integer-1, the data item will be displayed on the specified CRT line. All previous information displayed on the CRT will not be affected.

When the LINE clause specifies NEXT PAGE, the CRT will be cleared prior to writing the new data lines at the top of the display. The NEXT PAGE option may be used only at the 01 level.

COLUMN IS clause

The format of this clause is:

$$[\text{;COLUMN IS integer - 1}]$$

The COLUMN clause specifies the column number of the CRT in which the leftmost character of the data item is to be displayed.

A group defined by the 01 level becomes a CRT form description group by the occurrence of one or more LINE or COLUMN clauses. The first occurrence of a LINE or COLUMN clause must appear prior to any VALUE clauses or the group will not be considered a CRT form description. However, once a LINE or COLUMN clause has been specified, the entire group becomes a CRT form description and subsequent data description sentences do not have to specify LINE or COLUMN clauses unless they are needed to position the data item. If the LINE

clause is omitted, the data will be displayed on the same line as the previous data item if the new column is greater than the column of the previous data item. Otherwise, if the new column is less than the column number of the previous data item, the new data item will be displayed on the following line. If the COLUMN clause is omitted, the data will be displayed starting in column two of the line. If both the LINE and COLUMN clauses are omitted, the new data will be displayed one column after the end of the previous data item.

When specifying a line and column position for a data item, at least one column must be reserved preceding the data item to allow for CRT control attributes.

Video Attributes

Video attributes are automatically generated for CRT data descriptions based upon the type of the literal used in the VALUE clause.

- numeric or nonnumeric literal—the data item will be protected, normal video.
- figurative constant—the data item will be unprotected and underlined.
- hexadecimal constant—the data item is considered to be composed entirely of video control codes. No other attributes will be added by the COBOL compiler. In addition, the default line and column positions are not changed. It is assumed that the screen is left protected by the programmer.

The compiler generates the equivalent of a FILLER data description entry preceding the CRT data description sentence. This "FILLER" contains CRT control codes. Thus, CRT control cannot be specified on a level 77 data description and the size of groups are automatically increased to include the internally generated codes.

Examples

```
01 INVENTORY-RECORD, LINE IS NEXT PAGE.
   02 FILLER PIC X(11) LINE 3, COLUMN 5 VALUE 'DESCRIPTION.'
   02 FILLER PIC X(15) VALUE SPACES.
01 EMPLOYEE-RECORD.
   02 FILLER PIC X(8) LINE 2; VALUE 'EMP-NAME.'
   02 NAME PIC X(10) VALUE SPACES.
   02 FILLER PIC X(3) COLUMN 35; VALUE 'AGE.'
   02 AGE PIC ZZ COLUMN 45; VALUE SPACES.
01 BLINK-HELLO, LINE IS NEXT PAGE.
   02 FILLER PIC X LINE 5, COLUMN 35 VALUE $E0.
   02 FILLER PIC X(5) VALUE "HELLO."
```

CHAPTER 6

PROCEDURE DIVISION

GENERAL DESCRIPTION

The PROCEDURE DIVISION of a COBOL source program specifies the procedures—the precise sequence of processing operations—needed to solve a given problem. These operations (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English.

PROCEDURE DIVISION Elements

Statements

A statement consists of a COBOL verb followed by appropriate operands (data-names or literals) and reserved words. The three types of statements are

1. Compiler directing
2. Imperative
3. Conditional

Compiler Directing Statement. A compiler directing statement directs the compiler to take certain actions at compilation time. Compiler directing statements are: COPY.

Imperative Statement. An imperative statement specifies an action to be taken unconditionally by the object program. An imperative statement may consist of a series of imperative statements.

Conditional Statement. A conditional statement describes a condition that is tested to determine which of alternate paths of programmed processing flow is to be taken. Conditional statements are:

1. READ and RETURN statements that have the AT END or INVALID KEY options.
2. WRITE statements with the INVALID KEY option.
3. Arithmetic statements with the SIZE ERROR option.
4. IF statements.

Sentences

A sentence is a single statement or series of statements terminated by a period. A single semicolon may be used as a separator between statements within a sentence.

Paragraphs

A paragraph consists of one or more sentences identified by a beginning paragraph-name.

Sections

A section comprises one or more successive paragraphs, and must begin with a section header. A section header consists of a section-name followed by the word SECTION and a period.

Paragraph and Section Naming

Every paragraph or section has a programmer-supplied name that is given in the header entry. This name is used for reference (as, for example, when specifying a GO TO paragraph-name or GO TO section-name).

PROCEDURE DIVISION Structure

The formats of the PROCEDURE DIVISION are:

Format 1

PROCEDURE DIVISION

$$\left\{ \text{section-name } \underline{\text{SECTION.}} \right\}$$
$$\left\{ \text{paragraph-name. } \left\{ \text{sentence.} \right\} \dots \left\{ \dots \right\} \dots \right\} \dots$$

Format 2

PROCEDURE DIVISION

$$\left\{ \text{paragraph-name. sentence.} \right\} \dots \left\{ \dots \right\} \dots$$

Execution of the program begins at the first statement of the first section.

ARITHMETIC-EXPRESSIONS

An arithmetic-expression is a combination of numeric literals and data item identifiers (data-names) joined by one or more arithmetic operators in such a way that the entire expression can be reduced to a single numeric value. An arithmetic operator is a symbol representing addition, subtraction, etc. Spaces must be left on either side of an operator included in an arithmetic-expression. The operators are:

- + Addition
- Subtraction
- * Multiplication
- / Division

Also, the operator ‘-’ may be used as a unary - to indicate logical negation.

The following are examples of arithmetic-expressions:

RATE * TIME

GROSS - DEDUCTIONS

OVERTIME * 1.5 + REGULAR-TIME

Note that each of the above expressions is a combination of identifiers or literals joined by arithmetic operators. At object time each identifier represents a value and, in each of the above examples, one numeric value results from the specified computation. An arithmetic-expression may be used in the COMPUTE statement or in conditional expressions. It is therefore possible to test a given arithmetic-expression to see whether it reduces to a specific value.

ORDER OF COMPUTATION IN COMPOUND CONDITIONS

The method of evaluation of an arithmetic-expression can be specified by parentheses. Thus the expression $A * B + C$ might be considered ambiguous, because $(A * B) + C$ or $A * (B + C)$ are possible. If parentheses are not written to specify the order of computation, COBOL evaluates an arithmetic-expression using the following rules:

1. The unary $-$ is performed first.
2. Then, multiplication and division are performed.
3. Finally, addition and subtraction are performed.
4. In each of the three steps above, computation starts at the left of the expression and proceeds to the right. Thus $A * B / C$ is computed as $(A * B) / C$ and $A / B * C$ is computed as $(A / B) * C$.
5. When parentheses are present, computation begins with the innermost set and proceeds to the outermost. Items grouped in parentheses are evaluated in accordance with the above rules, and the result is then treated as if the parentheses were removed.

Rules for specifying operators, left and right parentheses, and a variable (data-name, literal, figurative-constant) are given in Table 6-1.

TABLE 6-1. Rules for Constructing Arithmetic-Expressions

First Symbol	Second Symbol					
	Variable	*or/	- or +	unary -	()or End of Expression
Variable	-	P	P	-	2	P
* or /	P	-	1	P	P	-
- or +	P	-	1	P	P	-
unary -	P	-	-	-	P	-
(or Beginning or Expression	P	-	P	P	P	-
)	-	P	P	-	-	P

1. This is permitted when $-$ indicates the sign of a numeric literal.
 2. Parentheses immediately following a data-name indicate the presence of a subscript. The subscript is considered part of the variable.
- P. A specified pair of symbols is permitted.
 $-$ A specified pair of symbols is not permitted.

Note that the use of a complex arithmetic-expression may require the computer to compute intermediate results that overflow on the high-order end or truncate on the low-order end.

CONDITIONAL STATEMENTS

A conditional statement describes a condition that is tested to determine selection of alternate paths of programmed processing flow. The programmer can accomplish this branching using the following types of statements:

1. The GO TO . . . DEPENDING ON . . . , which branches to one of several procedure-names.
2. Statements with exception branches: AT END, INVALID KEY, and ON SIZE ERROR.
3. The IF, and PERFORM, in which the condition is explicitly stated.

Relations

Relational-operators in the COBOL language are

IS [NOT] { GREATER THAN }
 { \geq }

IS [NOT] { LESS THAN }
 { \leq }

IS [NOT] { EQUAL TO }
 { \equiv }

EQUALS

Underlined words in the above list must be present when the relational-operator is used. Words not underlined may be omitted if the programmer desires, with no effect on the meaning of the relational-operator.

Relational-operators are combined with identifiers or literals to create relation conditions. The general format is

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

Logical Operators (AND, OR, and NOT)

The three logical operators are AND, OR, and NOT. AND and OR are used to create a "compound condition" when two or more tests are specified in the same expression. NOT is used to specify the negation of a condition.

Consider the following example:

IF CODE IS ZERO AND AGE NOT GREATER THAN 21 ADD A TO B.

Notice how AND and NOT are used to augment the two basic tests. Because the tests are connected by AND, they both must be true for A to be added to B.

Consider the following:

IF CODE IS NOT ZERO OR AGE GREATER THAN 21 ADD C TO D.

This time the logical operator OR specifies that C is to be added to D if either or both conditions are fulfilled.

NOT can be used in two ways with a simple relational condition: in the relational-operator as in AGE NOT GREATER THAN 21, or preceding the entire condition as in NOT AGE GREATER THAN 21. AGE NOT GREATER THAN 21 and NOT AGE GREATER THAN 21 are exactly equivalent in meaning. If NOT precedes a simple relational condition that contains NOT in the relational-operator, a double negative results and causes an error.

Other Condition Tests

Sign Test

The format of this test is

IF { data-name
arithmetic-expression } IS [NOT] { POSITIVE
ZERO
NEGATIVE }

The sign test is also effectively a special case of relation testing equivalent to testing whether an expression is GREATER THAN, LESS THAN, or EQUAL TO ZERO. The data-name must be a numeric value that, if unsigned and not equal to zero, is assumed to be positive. The value zero is considered neither positive nor negative. The statement GROSS IS NEGATIVE is equivalent to GROSS IS LESS THAN 0; GROSS IS POSITIVE is equivalent to GROSS IS GREATER THAN 0. Any condition that can be expressed as a sign condition can be expressed as a simple relational condition; the sign condition is merely a convenient way of expressing certain situations.

Class Test

The format of this test is

IF data-name IS [NOT] { NUMERIC
ALPHABETIC }

The data-name must be defined in the DATA DIVISION as USAGE DISPLAY. Table 6-2 lists cases where the class test is valid and meaning of the results.

Comparison of Numeric Items

For numeric items a relation test determines that the value of one of several items is less than, equal to, or greater than the others, regardless of the length. Numeric items are compared algebraically after alignment of decimal points. Zero is considered a unique value regardless of length, sign, or implied decimal-point location of an item.

Comparison of Non-Numeric Items

For nonnumeric items a comparison determines that one of the items is less than, equal to, or greater than the other with respect to the binary collating sequence of characters in the ASCII character set. If the nonnumeric items are of equal length, the comparison proceeds by comparing characters in corresponding character positions starting from the high-order position and continuing until either a pair of unequal characters or the low order position of the

item is compared. If the non-numeric items are of unequal length, comparison proceeds as described for items of equal length. If this process exhausts the characters of the shorter item, the shorter item is less than the longer unless the remainder of the longer item consists solely of spaces, in which case the items are equal.

Table 6-3 indicates characteristics of the compared items and the type of comparison made.

TABLE 6-2. Valid Class Tests

PICTURE		Allowable Characters	Valid Tests	Meaning
Must Contain	May Contain			
A	B	Alphabetic (A–Z and space)	[NOT]ALPHA-BETIC	(Not) only characters A–Z and space appear
A 9	X B O	Alphanumeric (any character)	[NOT]ALPHA-BETIC	(Not) only characters A–Z and space appear
X	A 9 B O			
S 9	O V P	Zoned decimal with operational sign	[NOT] NUMERIC	(Not) only characters 0–9 appear
9	O V P	Zoned decimal without sign	[NOT] NUMERIC	(Not) only characters 0–9 appear.

TABLE 6-3. Permissible Comparisons

Item Characteristics		GR	X	ND
Group Item	GR	A	A	A
Alphabetic, Alpha-numeric, and Edited	X	A	A	A
Numeric Display	ND	A	A	9

- A. Alphanumeric or byte comparison, byte-by-byte from left to right
 9. Numeric comparison

Conditional Statements with Exception Branches

The format of these statements is

$$\left\{ \begin{array}{l} \text{AT END} \\ \text{INVALID KEY} \\ \text{ON SIZE ERROR} \end{array} \right\} \left\{ \text{imperative-statements} \dots \right\}$$

The READ, RETURN, WRITE, REWRITE, DELETE, ADD, SUBTRACT, MULTIPLY, and DIVIDE verbs specify the exception branch as either an optional or a required part of the statement. When the exception branch is present, the verb in whose format it is written is considered to be a conditional statement. Normally, control bypasses the exception branch to the first statement in the next sentence or the first statement beyond the next ELSE (within an IF statement), but when the exception condition is met, control is given to the imperative-statement following the AT END, INVALID KEY, or SIZE ERROR. None of the statements up to the next period or ELSE (within an IF statement) may be a conditional statement: thus “nesting” of exception branches is not allowed.

Nested Conditional Statements

The IF statement may have conditional statements in either of the branches taken because of the outcome of the condition test. Furthermore, the conditional statement can be another IF, thus it is possible to “nest” IFs (in other words, IFs may be contained within IFs). Refer to the “IF Statement” discussion later in this chapter.

INPUT/OUTPUT STATEMENTS

OPEN Statement

The general format of this statement is:

OPEN [INPUT [file-name] . . .]

[OUTPUT [file-name] . . .]
 [EXTEND [file-name] . . .]
 [I-O [file-name] . . .]

The OPEN statement initiates processing of the files named in the statement.

One of the INPUT, OUTPUT, EXTEND or I-O options must be specified. The I-O option pertains only to files on direct access media used when ACCESS IS RANDOM is specified.

The EXTEND option means that the file is to be opened for output and that new records are to be added after the last record currently in the file.

An OPEN statement must be executed prior to any other input/output statement. A second OPEN statement for a given file cannot be executed prior to the execution of a CLOSE statement for that file. The OPEN statement itself does not obtain or dispatch data; a READ or WRITE statement must execute to obtain or release, respectively, the first data record.

START Statement

The START statement provides a means for logical positioning within an indexed file for subsequent sequential retrieval of records.

Format:

START file-name [KEY IS $\left. \begin{array}{l} \text{EQUAL TO} \\ = \\ \text{GREATER THAN} \\ > \\ \text{NOT} \{ \text{LESS} \} \text{THAN} \\ < \end{array} \right\}$ data-name]

[INVALID KEY imperative-statement]

When the START statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must name an indexed file with sequential or dynamic access. File-name must be defined in an FD entry in the Data Division.

When the KEY option is not specified, the EQUAL TO relational operator is implied. When the START statement is executed, the EQUAL TO comparison is made between the current value in the RECORD KEY and the corresponding key field in the file's records. The Current Record pointer is positioned to the logical record in the file whose key field satisfies the comparison.

When the KEY option is specified, *data-name* may be either

- The RECORD KEY for this file, or
- Any alphanumeric data item subordinate to the RECORD KEY whose leftmost character position corresponds to the leftmost character position of the RECORD KEY (that is, a generic key).

When the START statement is executed, the comparison specified in the KEY relational operator is made between *data-name* and the key field in the file's records. The Current Record Pointer is positioned to the first logical record in the file whose key field satisfies the comparison.

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, and the position of the Current Record Pointer is undefined.

READ Statement

For sequential access, the READ statement makes available the next logical record from file. For random access, the READ statement makes available a specified record from a file.

The formats of this statement are:

Format 1

READ file-name [NEXT] RECORD [INTO identifier]
[AT END imperative-statement]

Format 2

READ file-name RECORD [INTO identifier]; INVALID KEY imperative-statement

Functions of the READ verb are:

1. Sequential file processing (Format 1) makes available the next logical record from an input file and allows execution of a specified series of imperative-statements when the end-of-file is detected.
2. Random file processing (Format 2) makes available a specific record from an indexed file and allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found to invalid.

When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must be defined in an FD entry in the Data Division.

Format 1: When ACCESS MODE SEQUENTIAL is specified or assumed for a file, this format must be used. For such files the statement makes available the next logical record from the file. For indexed files, the NEXT option need not be specified; for sequential files, the NEXT option must not be specified.

When ACCESS MODE DYNAMIC is specified for indexed files, the NEXT option must be specified for sequential retrieval. For such files, the READ NEXT statement makes available the next logical record from the file.

Before a Format 1 READ statement is executed, the Current Record Pointer must be positioned by the successful prior execution of an OPEN START, or READ statement. When the Format 1 READ statement is executed the record indicated by the Current Record Pointer is made available. For sequential files, the next record is the succeeding record in logical sequence. For a sequentially accessed indexed file, the next record is that one having the next higher RECORD KEY in collating sequence.

Format 2: This format must be used for indexed files in random access mode, and for random record retrieval in the dynamic access mode.

Execution of a Format 2 READ statement causes the value in the RECORD KEY to be compared with the values contained in the corresponding key field in the file's records until a record having an equal value is found. The Current Record Pointer is positioned to this record, which is then made available.

If no record can be so identified, an INVALID KEY condition exists, and execution of the READ statement is unsuccessful.

Immediately following execution of a READ statement, the next logical record in the file is accessible in the logical record area associated with the file as defined by the Record Description entry. When multiple record descriptions follow a File Description (FD) entry, it is the responsibility of the programmer to recognize which record is present in the area at any

given time. The record is available in the logical record area until another READ statement or a CLOSE statement for that file is executed.

The INTO option is equivalent to a READ statement followed by a MOVE, and results in the record obtained by execution of the READ becoming available in both the record area for the file and in the location indicated by the identifier. The record is moved from the record area into the identifier in accordance with the rules for the MOVE statement.

In the case where the file contains records of varying lengths, the size of the longest record is assumed for the input record for the purpose of executing the MOVE.

The AT END clause is required for files that are accessed sequentially. The statements introduced by this clause are executed when end-of-file is encountered.

For files with SEQUENTIAL organization, when the AT END condition has been recognized, a READ statement for this file must not be executed until a successful CLOSE statement followed by a successful OPEN statement have been executed for this file.

For files with INDEXED organization, when the AT END condition is recognized, a Format 1 READ statement for this file must not be executed until one of the following has been successfully executed:

- A CLOSE statement followed by an OPEN statement
- A Format 2 READ statement (dynamic access)
- A START statement

The INVALID KEY clause must be written for files for which ACCESS IS RANDOM is specified. The imperative-statements are executed if a record corresponding to the contents of the RECORD KEY cannot be located in the file.

The contents of the RECORD KEY data item must be appropriately established prior to execution of the READ statement itself.

WRITE Statement

The formats of this statement are:

Format 1

WRITE record-name [FROM identifier-1] $\left[\begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right] \text{ADVANCING}$
 $\left. \begin{array}{l} \text{identifier-2 LINES} \\ \text{integer LINES} \\ \text{PAGE} \end{array} \right\}$

Format 2

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

The WRITE statement releases a logical record to an output file. For random access files the statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN OUTPUT, OPEN EXTEND, or OPEN INPUT-OUTPUT must be executed before a WRITE statement can be executed for a file. Once the WRITE is executed there is no guarantee that the logical record released thereby still exists in the logical record area for the file.

A WRITE statement bearing the FROM option is equivalent to a MOVE identifier-1 TO record-name statement followed by WRITE record-name. Moving takes place in accordance with rules for the MOVE statement.

Format 1 relates to files opened for sequential access. The ADVANCING option applies to files containing output destined to be printed. Integer should be an unsigned integer, and identifier-2, similarly, should contain a nonnegative integer. The line is printed BEFORE or AFTER the specified number of lines is spaced.

Format 2 is used for mass storage files. Statements following the INVALID KEY clause are executed when:

1. No space exists on the file media to accommodate the record.
2. The file is open for OUTPUT or I-O and a record corresponding to the contents of the RECORD KEY already exists in the file.

REWRITE Statement

The format of this statement is:

REWRITE record-name [FROM identifier-1]; INVALID KEY imperative-statement.

The REWRITE statement rewrites a previously read logical record to the output file. The statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN I-O must be executed before a REWRITE statement can be executed for a file. Once the REWRITE is executed there is no guarantee that the logical record rewritten still exists in the logical record area for the file.

The statements following the INVALID KEY clause are executed when the record corresponding to the contents of the RECORD KEY clause was not previously read.

DELETE Statement

The format of this statement is

DELETE file-name; INVALID KEY imperative-statement

The DELETE statement deletes a logical record from the output file. The statement also allows execution of a specified series of imperative-statements if the contents of the associated RECORD KEY data item are found invalid.

An OPEN I-O must be executed before a DELETE statement can be executed for a file.

The statements following the INVALID KEY clause are executed when the record corresponding to the contents of the RECORD KEY clause is not found in the file.

CLOSE Statement

The format of this statement is:

CLOSE [file-name] [WITH DELETE] . . .

The CLOSE statement terminates the processing of files. Execution of a CLOSE statement causes the standard closing procedures to be carried out on the file named. An OPEN statement must be executed before a CLOSE can be honored for a file; once closed, a file may not be referenced again until another OPEN statement is executed for that file.

If the DELETE option is specified, all records in the file will be deleted.

ACCEPT Statement

The format of this statement is:

ACCEPT identifier-1 [, identifier-2] . . .

The ACCEPT statement specifies acceptance of data from the CRT. It is normally used to read unprotected CRT fields.

The identifier must be an unedited DISPLAY data item or a group item. Refer to the operations manual for additional information on reading unprotected fields from the CRT.

DISPLAY Statement

The format of this statement is:

$$\text{DISPLAY } \left\{ \begin{array}{l} @(row, column) \\ \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[, \left\{ \begin{array}{l} @(row, column) \\ \text{identifier-2} \\ \text{literal-2} \end{array} \right\} \right] \dots$$

The DISPLAY statement enables data to be written to the CRT. The clause, @(row, column), allows the cursor to be positioned to any location on the CRT screen. The identifiers "row" and "column" may be either a numeric literal or a numeric data item. The cursor positioning clause may appear alone or in conjunction with other operands in the DISPLAY statement. If the cursor positioning clause is not present, a carriage return/line feed pair is sent to the display station. When a DISPLAY statement contains more than one operand, the characters comprising the items named and any literals specified in the statement are displayed consecutively, with no spaces between characters unless specified.

Any remaining positions on a line at the end of the data transfer are left unchanged. Any number of literals or data names may be specified. The data-name may be that of a group or an elementary item and may also be subscripted. A literal in a DISPLAY statement may be numeric or nonnumeric and may be a hexadecimal constant to specify CRT or field attributes.

Example:

```
DISPLAY @(3,25), $8085, DATA-1, $84F3.
```

Refer to the operations manual for additional information on CRT control.

ARITHMETIC STATEMENTS

The basic arithmetic operations are specified by the four verbs ADD, SUBTRACT, MULTIPLY, and DIVIDE.

Rules for Arithmetic Verbs

The following general rules apply to all arithmetic verbs:

1. All literals specified in arithmetic statements must be numeric.
An identifier used in an arithmetic statement must be an elementary item and must be numeric.
2. The maximum size of an operand is 15 decimal digits. If the entry for an operand in the DATA DIVISION specifies a size greater than 15 digits or if a literal contains more than 15 digits, an error is indicated at compilation time.
3. The items in an arithmetic statement may be mixed sizes as long as they are all numeric. Any necessary decimal-point alignment is supplied automatically throughout computations.
4. No item used in computations may contain editing symbols. If such an item is used, a compilation-time diagnostic results. Operational signs and assumed decimal points are not editing symbols. An item used to receive results may

contain editing symbols if it is not used in subsequent computations as an operand. When an item used to receive results contains editing symbols, the result is edited according to editing specifications before it is moved to the item. **ROUNDED**, **GIVING** and **SIZE ERROR** options apply to all arithmetic statements.

GIVING Option

If the **GIVING** option is written, the value the identifier that follows the word **GIVING** is made equal to the calculated result of the arithmetic operation.

If the **GIVING** option is not written, each operand following the words **TO**, **FROM**, **BY**, and **INTO** in the **ADD**, **SUBTRACT**, **MULTIPLY**, and **DIVIDE** statements, respectively, must be an identifier (not a literal). Each identifier is used in the computation, and also receives the result.

ROUNDED Option

If the **ROUNDED** option is not specified, truncation occurs when the number of places calculated (after decimal-point alignment) for the result is greater than the number of places in the data item that is to be set equal to the calculated result. When the **ROUNDED** option is specified, the least significant digit of the resultant data-name increases in value by 1 whenever the most significant digit of the excess is greater than or equal to 5.

Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative.

Table 6-4 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

TABLE 6-4. Rounding or Truncation of Calculations

Calculated Result	Item to Receive Calculated Result		
	PICTURE	Value After Rounding	Value After Truncating
-12.36	S99V9	-12.4	-12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
0.0055	V999	0.006	0.005

SIZE ERROR Option

An arithmetic statement, if written with a **SIZE ERROR** option, is not an imperative-statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative-statements are allowed.

Whenever the number of integer places in the calculated result exceeds the number

of integer places specified for the resultant item, a size error condition arises. If the SIZE ERROR option is specified and a size error condition arises, the value of the resultant item is not altered and the series of imperative-statements specified for the condition is executed.

If the SIZE ERROR option is not specified and a size error condition arises, no assumption should be made about the correctness of the final result even though the program flow is not interrupted.

ADD Statement

The formats of this statement are:

Format 1

ADD { identifier-1 } [, identifier-2] . . . , identifier-n
 { literal-1 } [, literal-2]
 [ROUNDED] [; ON SIZE ERROR imperative-statement]

Format 2

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

Format 3

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

The ADD statement sums the values of two or more numeric items and/or literals and sets one or several items equal to the resultant value. Operands used in an ADD statement must conform to "Rules for Arithmetic Verbs" above in addition to specific rules applying to this individual statement. Use of the SIZE ERROR and ROUNDED options is also discussed in the referenced paragraph.

When Format 1 is used the values of all the operands including identifier-n are added together and the result is stored as the new value of identifier-n, the resultant-identifier.

Example:

Given the statement ADD A, B, C, the values of A, B, and C before and after execution are

	A	B	C
Before	5	6	8
After	5	6	19

Note that the values of A and B do not change as a result of the addition.

Format 2 adds the values of the operands (identifier-1 or literal-1 and identifier-2 or literal-2) preceding the reserved word TO, and this intermediate result is added to the data items specified by identifier-m, identifier-n, etc.

Example:

Given the statement ADD W, X, Y TO Z, the values of W, X, Y and Z before and after execution are:

	W	X	Y	Z
Before	2	7	8	12
After	2	7	8	29

Note that the value of all operands participates in the addition.

Format 3 adds the values of the operands (identifier-1 or literal-1 and identifier-2 or literal-2, etc.) preceding the reserved word GIVING, and this intermediate result is placed in identifier-m, identifier-n, etc.

Example:

Given the statement `ADD A, B, C, GIVING D`, the values of A, B, C and D before and after execution are:

	A	B	C	D
Before	1	2	3	5
After	1	2	3	6

Note that the intermediate result replaces the value of D and is not added to D.

SUBTRACT Statement

The formats of this statement are:

Format 1

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

FROM identifier-m [ROUNDED] [; ON SIZE ERROR imperative-statement]

Format 2

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

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

[; ON SIZE ERROR imperative-statement]

The SUBTRACT statement subtracts the value of a numeric item from another item and stores the result in a third item.

Format 1 subtracts the operands preceding the word FROM from identifier-m placing the result in identifier-m.

Format 2 subtracts the operands preceding the word FROM from identifier-m (literal-m) without changing the contents of identifier-m, placing the result in the item following GIVING.

Example:

Given the statement `SUBTRACT A FROM B GIVING C` the values of the operands before and after execution are:

	A	B	C
Before	10	80	90
After	10	80	70

MULTIPLY Statement

The formats of this statement are:

Format 1

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

[; ON SIZE ERROR imperative-statement]

Format 2

MULTIPLY { identifier-1 } BY { identifier-2 } GIVING
 { literal-1 } { literal-2 }

identifier-3 [ROUNDED] [;ON SIZE ERROR imperative statement]

The MULTIPLY statement can be used to multiply two items with the value of a third item being set to the product. Operands used in a MULTIPLY statement must conform to "Rules for Arithmetic Verbs" above, in which the SIZE ERROR and ROUNDED options are also discussed.

Format 1 allows the multiplicand (identifier-1 or literal-1) to be multiplied by the multiplier (identifier-2) and the value of identifier-2 to be set to the product. A literal cannot be used in place of identifier-2.

Example:

Given the statement MULTIPLY A BY B the values of the operands before and after execution are:

	A	B
Before	10	20
After	10	200

Note that the values of operand B change to reflect the multiplication.

Format 2 allows the multiplicand (identifier-1 or literal-1) to be multiplied by the multiplier (identifier-2 or literal-2).

Example:

Given the statement MULTIPLY A BY B GIVING C the values of the operands before and after execution are:

	A	B	C
Before	5	10	20
After	5	10	50

Note that the values of operands A and B remain the same, while the value of operand C changes.

DIVIDE Statement

The formats of this statement are:

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]

The DIVIDE statement divides the value of one numeric item into the value of one or more numeric items and sets the value of one or more items to the quotient. Operands used in a DIVIDE statement must conform to "Rules for Arithmetic Verbs" above in addition to specific rules applying only to this individual statement. Use of the SIZE ERROR and ROUNDED options is also discussed in the reference paragraph.

Format 1 allows one division, with the quotients stored as the value of the item following INTO. The dividend (identifier-2) divided by the divisor (identifier-1 or literal-1) and the value of the dividend set to the value of the associated quotient. Literals cannot be used in place of identifiers-2. The size error condition results when the divisor is zero or the quotient contains more integer positions than are available.

Example:

Given the statement DIVIDE A INTO B the values of the operands before and after execution are:

	A	B
Before	5	10
After	5	2

Format 2 allows the single quotient resulting from a division to be stored in a third item. If Format 2 is used, the dividend (identifier-2 or literal-2) is divided by the divisor (identifier-1 or literal-1), and the value of the resultant quotient becomes the new value of identifiers-3.

Example:

Given the statement DIVIDE A INTO B GIVING C the values of the operands before and after execution are:

	A	B	C
Before	5	10	15
After	5	10	2

COMPUTE Statement

The format of this statement is:

$$\underline{\text{COMPUTE}} \text{ identifier-1 } [\underline{\text{ROUNDED}}] = \left\{ \begin{array}{l} \text{identifier-n} \\ \text{literal-1} \\ \text{arithmetic-expression} \end{array} \right\}$$

[; ON SIZE ERROR imperative-statement]

The COMPUTE statement specifies computation that combines the individual processing of the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements by the use of an arithmetic-expression and stores the results in one or more items. This statement can also duplicate a MOVE statement when the arithmetic-expression is replaced by a literal or identifier.

The arithmetic-expression can consist of any meaningful combination of data-names, numeric literals, and the figurative-constant ZERO, joined by the arithmetic operators. The arithmetic-expression is evaluated and the resulting numeric value replaces the contents of identifier-1. All identifiers in the statement (including those in the arithmetic-expression) must be described in the DATA DIVISION as elementary numeric items.

The arithmetic-expression may be simple or complex. If it consists of one identifier (an elementary numeric item), the COMPUTE statement is equivalent to a MOVE statement, and the identifier-1 item is set to the value of this single item. Similarly, the arithmetic-expression may consist solely of a numeric literal.

Examples:

COMPUTE PAY= HOURS * RATE.

COMPUTE NET= (HOURS * RATE)– DEDUCTIONS.

DATA MANIPULATION STATEMENTS

MOVE Statement

The format of this statement is:

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

... [ON SIZE ERROR imperative-statement]

The MOVE statement moves data from one area of main storage to another. It edits the data (inserts, deletes, or replaces characters) if the PICTURE of the receiving item so requires.

This statement moves data in identifier-1 (or the specified literal) to identifier-2. Literal-1 may be a numeric literal, an alphanumeric literal, or a figurative-constant. Figurative-constants are treated as alphanumeric items. The same information may be moved simultaneously to additional areas as specified by identifier-3, etc.; such movement does not destroy the original data in identifier-1 but copies it in the designated areas. Identifier-1 or literal-1 is the source item; identifier-2, identifier-3, etc., are the receiving items or areas. Both the source and receiving items can be elementary or group items. (For purposes of the MOVE statement, a literal is considered an elementary item.) The manner in which the MOVE is performed depends not only on the type of source and receiving items but also on their classes.

The imperative-statement of the ON SIZE ERROR clause will be executed whenever significant characters (non-blank or non-zero) are truncated as a result of the move. This feature facilitates editing of input data.

The types of MOVE statements are discussed in the following paragraphs.

Alphanumeric Moves

Source data is stored left-justified in the receiving area. If the receiving area is not completely filled by data, remaining positions are filled with spaces. If the receiving item is alphabetic, it is treated as alphanumeric.

Examples:

<i>Source Data</i>	<i>PICTURE of Receiving Item</i>	<i>Receiving Item</i>
A B C D	A(4) or X(4)	A B C D
A B C D	A(5) or X(5)	A B C D
A B C D 1 2 3	X(8)	A B C D 1 2 3 △
1 2 3	X(8)	1 2 3 △ △ △ △ △
A B C D	A(3) or X(3)	A B C

If the receiving item is alphanumeric, the literal may be any literal or figurative-constant. If the figurative-constant takes the form of ALL any-literal, the literal must be enclosed in quotation marks and is considered an alphanumeric item. The size of an ALL any-literal item is determined by the size of the receiving item, with characters repeated from left to right.

Examples:

<i>Source Data</i>	<i>PICTURE of Receiving</i>	<i>Receiving Item</i>
'ABCD'	X(4)	A B C D





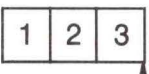
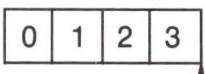

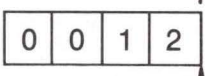





'123'	X(2)	1 2
123	X(5)	1 2 3 △ △
All 'X'	X(5)	X X X X X

Numeric Moves

When the source data is moved into the receiving area, it is aligned according to its decimal point and the decimal point in the receiving area. If there is no decimal point in either the source or receiving item, one is assumed of the right end of the item. Alignment by decimal points may result in the loss of leading or trailing digits, or both.

Any positions in the receiving area not filled with data are automatically filled with zeros. Such a situation could arise because of decimal-point alignment, difference in sizes between source and receiving items, or both. Any necessary conversion from one USAGE to another, together with any editing, takes place during the move.

Examples:

Source Data	PICTURE of Receiving Item	Receiving Item
	99V9	
	999V99	
	9999	
	9999	
-1. 2 3 (literal)	S9V99	
	9V9	
	9V9	

Editing

If the receiving item format specifies editing, the source data are edited concurrently with data movement. Editing occurs after decimal-point alignment. Editing symbols in the receiving item (currency signs, commas, etc.), make this item alphanumeric; if it is subsequently referenced as a source item in a MOVE statement, it is moved in accordance with the rules for alphanumeric items.

Examples:

<i>Source Data</i>	<i>PICTURE of Receiving Item</i>	<i>Receiving Item</i>
1 2 3 4 5 ↑	\$**9.99	\$ 1 2 3 . 4 5
1 2 3 4 5 ↑	999.9	1 2 3 . 4
0 0 0 1 2 ↑	\$**9.99	\$ * * 0 . 1 2

If the receiving item is numeric or numeric edited, the literal can be any numeric literal. The point location and size of the literal are determined by the actual literal in the source statement. Further examples of editing are given in "PICTURE Clause" under "DATA DIVISION Structure" in Chapter 5.

Examples:

<i>Source Data</i>	<i>PICTURE of Receiving Item</i>	<i>Receiving Item</i>
+1.23	S9V99	1 2 3 ⁺
+1.23	S9V9	1 2 ⁺
123	9(5)	0 0 1 2 3
+37	S999V99	0 3 7 0 0 ⁺
03737.3	\$***9.9	\$ 3 7 3 7 . 3

De-editing

In order to facilitate editing of CRT input fields, a de-edit is performed whenever a move from an alphanumeric to a numeric field occurs. The following actions are performed during a de-edit move.

1. All leading and trailing blanks of the source field are deleted.
2. Any leading sign (+ or -) is removed and added to the destination field.
3. The source and destination fields are aligned at the decimal point and the data moved to the destination field.
4. A SIZE ERROR occurs if any non-zero digits are truncated during the move.

Examples:

Source Data	PICTURE of Receiving Item	Receiving Item	SIZE ERROR?
1	S99V99	0 1 0 0	no
-1	S99V99	0 1 0 0̄	no
1.2	S99V99	0 1 2 0	no
1.23	S99V99	0 1 2 3	no
12.34	S99V99	1 2 3 4	no
123	S99V99	2 3 0 0	yes
1.234	S99V99	0 1 2 3	yes
A.BC	S99V99	0 A B C	no
1b2	S99V99	b 2 0 0	yes

Sample Field Edit Program:

MOVE SOURCE TO DESTINATION; ON SIZE ERROR
DISPLAY "NUMBER TOO BIG."

IF DESTINATION IS NOT NUMERIC;
DISPLAY "NUMBER NOT NUMERIC."

TABLE 6-5. Permissible Moves

Source Item		Receiving Field		
		GR	X	ND
Group	GR	A	A	A
Alphabetic, Alphanumeric, or Edited	X	A	A	9 ¹
Numeric Display	ND	A	A ¹	9
Numeric Literal		A	A ¹	9
Nonnumeric Literal		A	A	9 ¹

A Alphabetic or byte move, byte-by-byte from left to right with blank fill.

A¹ Permissible if source is an integer. In this case the integer is converted to numeric display, moved byte-by-byte into the field, and left justified with space fill.

9 Numeric MOVE

9¹ Any nonnumeric characters in the source field cause unpredictable data. De-editing is performed.

INSPECT Statement

The INSPECT statement provides the ability of replace occurrences of characters in a data item.

Format

INSPECT identifier-1 REPLACING { ALL
LEADING } { identifier-2 }
FIRST } { literal-1 }
BY { identifier-3 }
{ literal-2 }

Identifier-1 must reference either a group item or any category of an elementary item, described implicitly or explicitly as USAGE IS DISPLAY. Identifier-2 through identifier-3 must reference a one-byte elementary alphabetic, alphanumeric, or numeric item described im-

plicity or explicitly as USAGE IS DISPLAY. Literals must be nonnumeric and may be any figurative constant except ALL.

Rules Applicable to All Formats

Inspection begins at the leftmost character position of the data referenced by identifier-1, regardless of its class, and proceeds on a character-by-character basis to the rightmost character position. The contents of the data item referenced by identifier-1 is treated subject to whether the identifier is described as alphanumeric, unsigned numeric, or signed numeric:

1. Alphanumeric—identifier treated as a character string.
2. Unsigned numeric—inspected as though it had been redefined as alphanumeric and the INSPECT statement had been written to reference the redefined data.
3. Signed numeric—inspected as though the data item had been moved to an unsigned numeric data item of the same length, subject to the rules set forth above.
4. The rules for replacement are as follows:
 - a. When literal-1 is a figurative constant, each character in the data referenced by identifier-1 that is equal to the figurative constant is replaced by the single character referenced by literal-2 or identifier-3.
 - b. When literal-2 is a figurative constant, each character in the data referenced by identifier-1 that is equal to the character referenced by literal-1 or identifier-2 is replaced by the character referenced by the figurative constant.
5. The required words ALL, LEADING, and FIRST are adjectives that apply to the succeeding BY phrase:
 - a. If ALL identifier-2/literal-1s are to be replaced, this is done according to the replacement rules specified in paragraph 4.
 - b. If the adjective LEADING is used, all occurrences of the character string referenced by literal-1 or identifier-2 are replaced by the character string referenced by literal-2 or identifier-3, provided that the leftmost such occurrence, in the data referenced by identifier-1, is at the point where replacement begins.
 - c. If the adjective FIRST is used, the leftmost occurrence, to the right of the point where replacement of the character string referenced by literal-1 or identifier-2 begins, is replaced, in the data referenced by identifier-1, by the character string referenced by literal-2 or identifier-3.

Example:

```
77 SS-NUMBER PIC 9(9) VALUE 123456789.  
77 EDITED-SS-NUMBER PIC 999/99/9999.  
MOVE SS-NUMBER TO EDITED-SS-NUMBER.  
INSPECT EDITED-SS-NUMBER REPLACING ALL '/' BY '-.'
```

The new value of EDITED-SS-NUMBER will be 123-45-6789.

SEQUENCE CONTROL STATEMENTS

COBOL provides the programmer with the following commands that control the order in which statements are executed:

1. GO TO permanently releases control to the first statement in the procedure named.
2. PERFORM causes statements in a remote procedure to be executed and control return to the statement following the PERFORM.
3. STOP allows the program to terminate in an orderly manner.
4. IF causes control to branch into either a "true" or "false" path, depending on the outcome of a condition test written in the program. The paths rejoin at the beginning of the next sentence unless a GO TO branch is used in one or both paths.
5. EXIT merely declares that the paragraph in which it is contained is a transfer point that may be referenced by other sequence control statements.

Normal Sequence Control

The starting location for the program is at the first statement of the PROCEDURE DIVISION. Control then proceeds to subsequent successive statements until the end of paragraph or section is reached. Unless the paragraph or section is executed under control of a PERFORM statement, control then passes to the first statement in the next paragraph or section. Execution of a sequence control statement, of course, alters the normal sequence of control.

GO TO Statement

The format of this statement is:

Format 1

GO TO [procedure-name-1]

Format 2

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

The GO TO statement permanently transfers control, conditionally or unconditionally, to another point in a program.

Format 1 represents the unconditional GO TO statement: control is transferred to another paragraph or section of the PROCEDURE DIVISION as specified by procedure-name-1. GO TO can appear as the last of several statements in a series of statements.

Examples:

1. GO TO TEST-ROUTINE.
2. IF A EQUALS B GO TO SINE-ROUTINE ELSE ADD A TO B GO TO START-ROUTINE.

Format 2, referred to as the conditional GO TO, can constitute a multiple branch point. These branch points may be paragraphs or sections as specified by procedure-name-1, -2, etc. Since the branch is predicated on certain conditions, the value of a particular data item, identifier-1, is tested at the time the statement is executed to determine which branch point to take.

When the GO TO statement is executed, control is transferred to the paragraph or section specified by procedure-name-1, -2, or -n, depending on whether the data item value is equal to 1, 2, or n. Identifier-1 must be an elementary integral numeric item. Identifier-1 can be subscripted if necessary. If the value of identifier-1 is not within the range 1 through n, no transfer transpires; control passes to the next statement following the GO TO statement. A maximum of 16 procedure-names may be used in one GO TO statement.

Example:

GO TO FEDERAL-TAX, STATE-TAX, LOCAL-TAX DEPENDING ON GROSS-SALARY CODE.

PERFORM Statement

The formats of this statement are:

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 }
{ identifier-2 }
{ literal-2 } BY { identifier-3 }
{ literal-3 }

UNTIL condition-1

[AFTER { index-name-4 }
{ identifier-4 } FROM { index-name-5 }
{ identifier-5 }
{ literal-5 }] BY

{ identifier-6 }
{ literal-6 } UNTIL condition-2

[AFTER { index-name-7 }
{ identifier-7 } FROM { index-name-8 }
{ identifier-8 }
{ literal-8 }] BY

{ identifier-9 }
{ literal-9 } UNTIL condition-3]]

The PERFORM statement causes a departure and return from normal procedures execution to another part of the program to execute one or more procedures. These procedures are executed a predetermined number of times or until a specified condition is satisfied, after which normal procedures execution resumes. In its simplest format the PERFORM provides a branch, execution of the procedure, and a return; in the more complex formats a branch is made, but the number of executions is contingent upon a condition

controlled and tested by the statement. Thus, the PERFORM statement permits repetitive execution or looping using one statement, that is, it initializes and maintains loop criterion (variable), tests the criterion and performs operations.

The return point for the PERFORM statement is determined by whether the procedure to which it branches is a paragraph or section. When the instructions compiled from a PERFORM Statement are executed, they transfer control to the first statement of the specified procedure. Instructions that provide return to the statement following PERFORM are set up as follows:

1. If procedure-name-1 is a paragraph-name and a procedure-name-2 is not specified, control is returned after the last statement of the procedure-name-1 paragraph.
2. If procedure-name-1 is a section and a procedure-name-2 is not specified, control is returned after the last statement of the last paragraph of the procedure-name-1 section.
3. If procedure-name-2 is specified and is a paragraph-name, control is returned after the last statement of the procedure-name-2 paragraph.
4. If procedure-name-2 is specified and is a section-name, control is returned after the last statement of the last paragraph of the procedure-name-2 section.

Note: The "last statement" referenced in each of the above cases must not be an unconditional GO TO statement.

When procedure-name-2 is specified, the only required relationship between procedure-name-1 and procedure-name-2 is that of logical sequence, that is, execution sequence must proceed from procedure-name-1 to the last statement of the procedure-name-2 paragraph or section. GO TO statements and other PERFORM statements are permitted between procedure-name-1 and the last statement of procedure-name-2 provided that the sequence ultimately returns to the final statement of procedure-name-2.

If the logic of a procedure requires a conditional branch prior to the final sentence, the EXIT statement may be used to satisfy the foregoing requirements. In this case, procedure-name-2 must be the name of a paragraph consisting solely of the EXIT statement; all paths must eventually lead to this point. (See the "EXIT Statement" discussion below).

It is not necessary for procedures to be referenced by a PERFORM statement before they can be executed. Procedures can also be executed in normal sequence from the preceding statement, in which case return of control does not apply after execution of the last sentence in a particular procedure.

"Nested" PERFORM Statements

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with them included PERFORM must itself be either totally included in, or totally excluded from the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement whose execution point begins within the range of another PERFORM must not contain within its range the exit point of the other active PERFORM statement.

TIMES Option

In Format 2 the procedure is executed repetitively a certain number of times. The number of executions may be specified explicitly as an integer or implicitly as the value of an elementary data item.

If an identifier is used it may be of any numeric usage, and it may be subscripted. When this option is included, a counter is set up with a value equal to the value of the identifier-1 item or integer-1. Before each execution of the specified procedure, the counter is tested to see if it is negative or zero. If it is neither negative nor zero, the procedure is executed and the value of the counter decreased by one; when the value of the counter is negative or zero, the procedure has been executed by the specified number of times and control transfers to the statement following the PERFORM statement.

UNTIL Option

In Format 3, the number of times the procedure is executed is dependent on the truth or falsity of a condition (condition-1) rather than a stated value. Condition-1 can be any simple or compound conditional expression that is evaluated before the specified procedure is executed. If it is found to be false, the procedure is executed and the expression is evaluated again (values of the items may be altered by execution of the procedure) and tested for truth or falsity; this process is repeated until the conditional expression is found to be true, at which point control transfers to the statement following the PERFORM statement. If the conditional expression is found to be true when the PERFORM statement is first encountered, the specified procedure is not executed. (Refer to "Conditional Statements" at the beginning of this chapter).

VARYING Option

In Format 4 the VARYING option makes it possible to PERFORM a procedure repetitively, increasing or decreasing the value of one to three data items once for each execution until one to three conditional expressions are satisfied.

The flowcharts in Figure 6-6 illustrate the logic of the PERFORM statement when one, two, or three identifiers are varied.

- Let:
1. Each d. represent an identifier or index-name.
1
 2. Each l. represent a literal.
1
 3. Each c. represent a condition.
1
 4. Each p. represent a procedure-name.
1

Example:

To help clarify use of the VARYING subscript-name option, assume that a rate table is employed in a billing procedure and that the table requires periodic updating. This hypothetical rate table is three-dimensional: divided into five regions, each of which includes ten states, each of which contains rates for twelve cities. It is assumed further that an appropriate rate-updating procedure is available elsewhere in the program. Such a procedure might appear as

RATE-UPDATING. MULTIPLY RATE (REGION, STATE, CITY) BY ADJUST-FACTOR
GIVING RATE (REGION, STATE, CITY).

It is desired to execute this RATE-UPDATING procedure once for each city of each state in each region, using the current rate for a given city and producing an adjusted rate for that city. Accordingly, the programmer employs a PERFORM statement varying these items:

PERFORM RATE-UPDATING VARYING REGION FROM 1 BY 1 UNTIL REGION IS
GREATER THAN 5 AFTER STATE FROM 1 BY 1 UNTIL STATE EQUALS 11 AFTER

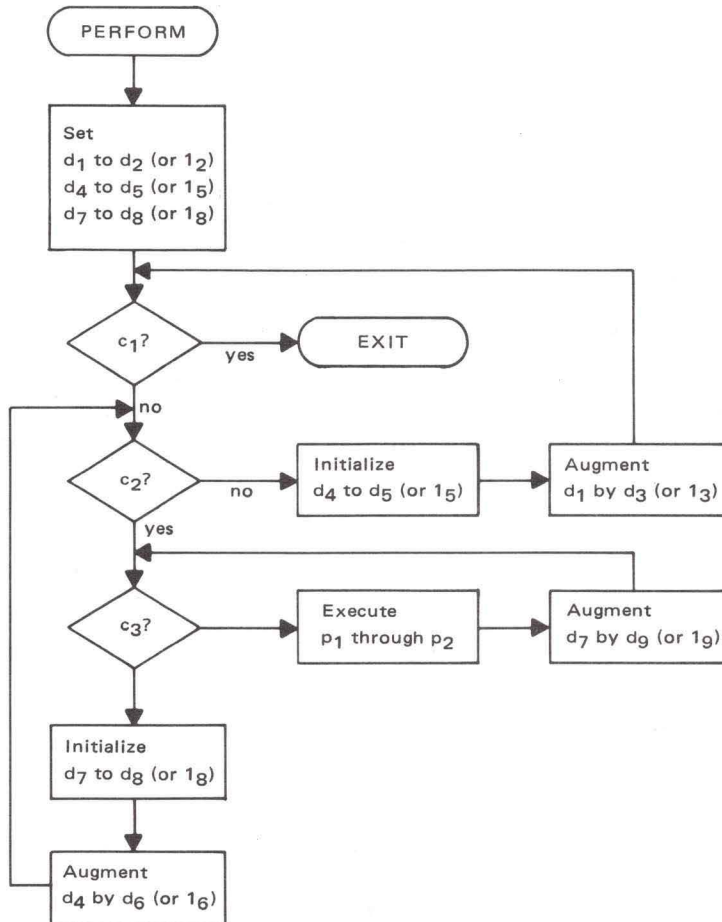
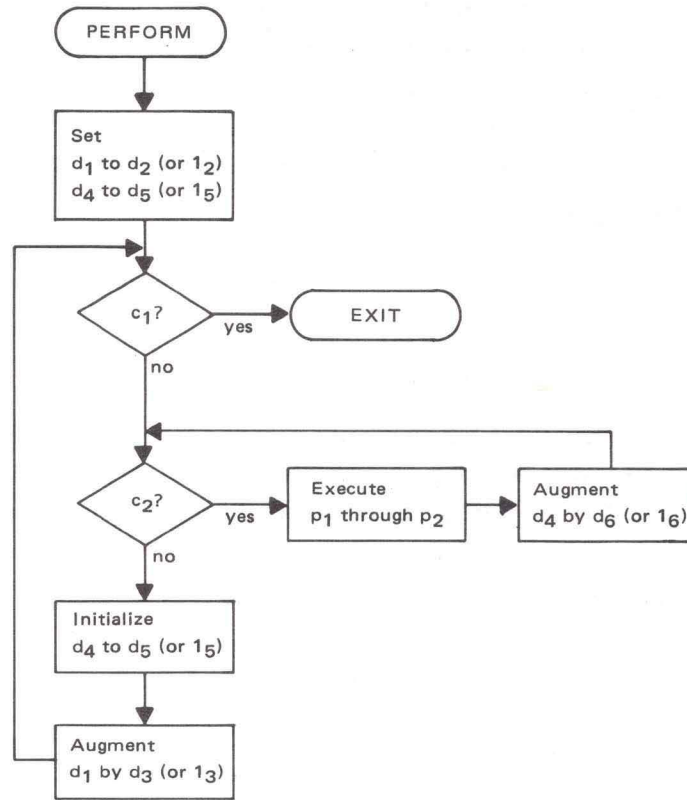
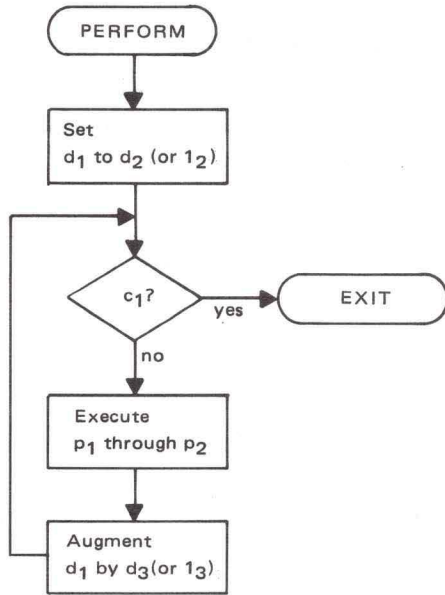


FIGURE 6-6. PERFORM Statement (VARYING Option)

CITY FROM 1 BY 1 UNTIL CITY IS GREATER THAN 12.

When the PERFORM is executed at object time, the RATE-UPDATING procedure is executed for the first city of the first state in the first region, then for the next city, etc. The PERFORM is complete when the procedure is executed for the twelfth city of the tenth state of the fifth region, by which time the procedure has been executed 600 times.

STOP Statement

The format of this statement is:

STOP { literal }
 { RUN }

The STOP statement permanently suspends execution of the object program. STOP RUN generates an end-of-program exit to the Monitor that terminates program execution permanently. If STOP is followed by a literal, the literal is typed out and execution is suspended. Any literal may be used.

EXIT Statement

The format of this statement is:

paragraph-name. EXIT

The EXIT statement ends a procedure to be executed by a PERFORM statement. EXIT must be the only statement in a paragraph; it is equivalent to a paragraph with no sentences and generates no code.

IF Statement

The format of this statement is:

IF condition THEN { statement-1 } [ELSE { statement-2 }]
 { NEXT SENTENCE } { NEXT SENTENCE }

The IF statement causes alternate sequences of operations to be followed, depending on whether the description of a data condition is found to be true or false when the data is evaluated. IF is followed by the description of the condition, then by the actions to be taken if the description of the condition is true. The word ELSE may be used, followed by the operations to be performed if the description of the condition is false.

The condition may be a simple condition as presented by the format below or a compound condition as described under "Conditional Statements" at the beginning of this chapter. The format of a simple condition is

{	{ identifier-1 literal-1 formula-1 }	{	IS [<u>NOT</u>]	{	<u>GREATER THAN</u> ≥ <u>LESS THAN</u> ≤ <u>EQUAL TO</u> =	}	}	{ identifier-2 literal-2 formula-2 }
	{ identifier-3 formula-3 }		IS [<u>NOT</u>]		{ <u>POSITIVE</u> <u>NEGATIVE</u> <u>ZERO</u> }			
	[identifier-4]		IS [<u>NOT</u>]		{ <u>NUMERIC</u> <u>ALPHABETIC</u> }			

Evaluation of the Condition

The condition is evaluated before any action is taken. If the condition is true, either statement-1 or NEXT SENTENCE is executed. When NEXT SENTENCE is specified, control is transferred to the next sentence, and the ELSE part of the statement is ignored. If the condition is false, either statement-2 or NEXT SENTENCE is executed. Control is transferred to the succeeding sentence when NEXT SENTENCE is specified.

Statement-1 or statement-2 may be a series of statements and each may be terminated by a period or ELSE.

Nested Conditional Statements

Statements-1 and -2 can be imperative-statements or imperative-statements followed by a conditional statement. When either statement-1 or statement-2 or both contains a conditional statement, the conditional statement becomes nested. Nested conditional statements may also contain conditional statements. Nested conditional statements are analogous to the use of parentheses for combining subordinate arithmetic-expressions so that the expressions become part of a larger arithmetic unit.

Evaluation of Nested IF Statements

Conditional statements contained within conditional statements (IFs within IFs) must be considered as paired IF and ELSE combinations, proceeding from left to right. Therefore, any ELSE encountered applies to the immediately preceding IF that is not already paired with an ELSE.

In essence, the number of occurrences of ELSE in any conditional statement must be equal to the number of occurrences of IF, regardless of the complexity caused by nesting, with the following exception: when ELSE or NEXT SENTENCE directly precedes the terminal period of a sentence, the entire phrase may be omitted and the period specified at the end of the previous phrase. This rule is extended to resulting sentences, etc. For each ELSE, the associated statement is executed only when the conditional expression in the corresponding IF is found to be false. If there are more IFs than ELSEs in a statement, it is assumed that ELSE NEXT SENTENCE phrases at the end of the sentence are omitted.

Example:

The sentence in the following paragraph contains two independent nests of conditional statements. The first nest ends after the statement PERFORM procedure-name-2; the second nest consists of the remainder of the sentence and has an implied ELSE NEXT SENTENCE before the period. Each upper-case letter of the alphabet corresponds to a conditional expression.

```
IF A IF B PERFORM procedure-name-1 ELSE NEXT SENTENCE ELSE  
IF C NEXT SENTENCE ELSE PERFORM  
procedure-name-2 IF D PERFORM procedure-name-3 IF E PERFORM  
procedure-name-4 IF F PERFORM procedure-name-5 ELSE PERFORM  
procedure-name-6 ELSE STOP RUN.
```

TABLE-HANDLING STATEMENTS

The structure of a table is defined by the use of an OCCURS clause (refer to "OCCURS Clause" under "Data Description Entries" in Chapter 5). Entries in a table may be referenced by a subscript, which contains a number indicating a particular occurrence of the elements within a table. Location of the particular item desired is obtained by multiplying the

value of the subscript by the length of the previous element and adding the product to the address of the table base. The programmer provides for execution of statements ensuring that subscripts contain the proper values to permit current table elements to be referenced.

Indexing is a technique similar to subscripting but has the advantage in efficiency that no address computation is involved; an index contains a direct pointer to an individual element in a table rather than a mere occurrence number. Two statements, SEARCH and SET, facilitate the correct setting of indexes.

SEARCH Statement

The format of this statement is

```

SEARCH identifier-1 [ VARYING { index-name-1 } ] [; AT END imperative-
                    { identifier-2 } statement-2]
;WHEN condition-1 { imperative-statement-2 }
                  { NEXT SEQUENCE }
[ ;WHEN condition-2 { imperative-statement-3 }
                   { NEXT SEQUENCE } ]

```

The SEARCH statement searches a table for a table element that satisfies the specified condition and adjusts the associated index-name to indicate that table element. Identifier-1 may not be subscripted or indexed, and its description must also contain an OCCURS and an INDEXED BY clause. Identifier-2, when specified, must be described as USAGE IS INDEX or as the name of a numeric elementary item described without any positions to the right of the assumed decimal point. Identifier-2 is incremented by the same amount and at the same time as the occurrence number represented by the index-name associated with identifier-1.

Condition-1, condition-2, etc., may be any condition described under "Conditional Statements" at the beginning of this chapter.

When the SEARCH statement is executed, a serial search operation takes place starting with the current index setting and following either of two procedures:

1. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number greater than the highest permissible occurrence number for identifier-1, the SEARCH is immediately terminated. If the AT END clause is specified, imperative-statement-1 is executed; if not, control passes to the next sentence.
2. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number less than the highest permissible occurrence number for identifier-1, the SEARCH statement operates by evaluating the conditions sequentially as written, making use of index settings (wherever specified) to determine the occurrence of those items to be tested. If none of the conditions is satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings, unless the new value of the index-name settings for identifier-1 corresponds to a table element exceeding the last element of the table by one or more occurrences, whereby the search terminates as indicated in 1. above. If one of the conditions is satisfied upon

evaluation, the search immediately terminates and the imperative-statement associated with that condition is executed; the index-name remains set at the occurrence that caused the condition to be satisfied.

If any of the specified imperative-statements do not terminate with a GO TO statement, control passes to the next sentence after execution of the imperative-statement.

In the VARYING option, if index-name-1 appears in the INDEXED BY clause of identifier-1, that index-name is used for this search; otherwise, the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used. If index-name-1 appears in the INDEXED BY clause of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount and at the same time as the occurrence number represented by the index-name associated with identifier-1.

If identifier-1 is an item in a group or a hierarchy of groups each of whose description contains an OCCURS clause, each of those groups must also have an index-name associated with it; the settings of these index-names are used throughout the execution of the SEARCH statement to refer to identifier-1 or items therein. These index settings are not modified by the execution of the SEARCH statement (unless stated as index-name-1); only the index-name associated with identifier-1 (and the item identifier-2 or index-name-1) is incremented by the SEARCH.

A diagram of SEARCH operation containing two WHEN phrases is shown in Figure 6-7.

SET Statement

The formats of this statement are:

Format 1

$$\underline{\text{SET}} \quad \left\{ \begin{array}{l} \text{index-name-1} \\ \text{identifier-1} \end{array} \right\} \quad \text{TO} \quad \left\{ \begin{array}{l} \text{index-name-2} \\ \text{identifier-2} \\ \text{literal-1} \end{array} \right\}$$

Format 2

$$\underline{\text{SET}} \text{ index-name-3} \quad \left\{ \begin{array}{l} \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{array} \right\} \quad \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-2} \end{array} \right\}$$

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

All identifiers must be either index data items or numeric elementary items described without any positions to the right of the assumed decimal point, except that identifier-3 must not be 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 specification in the INDEXED BY clause.

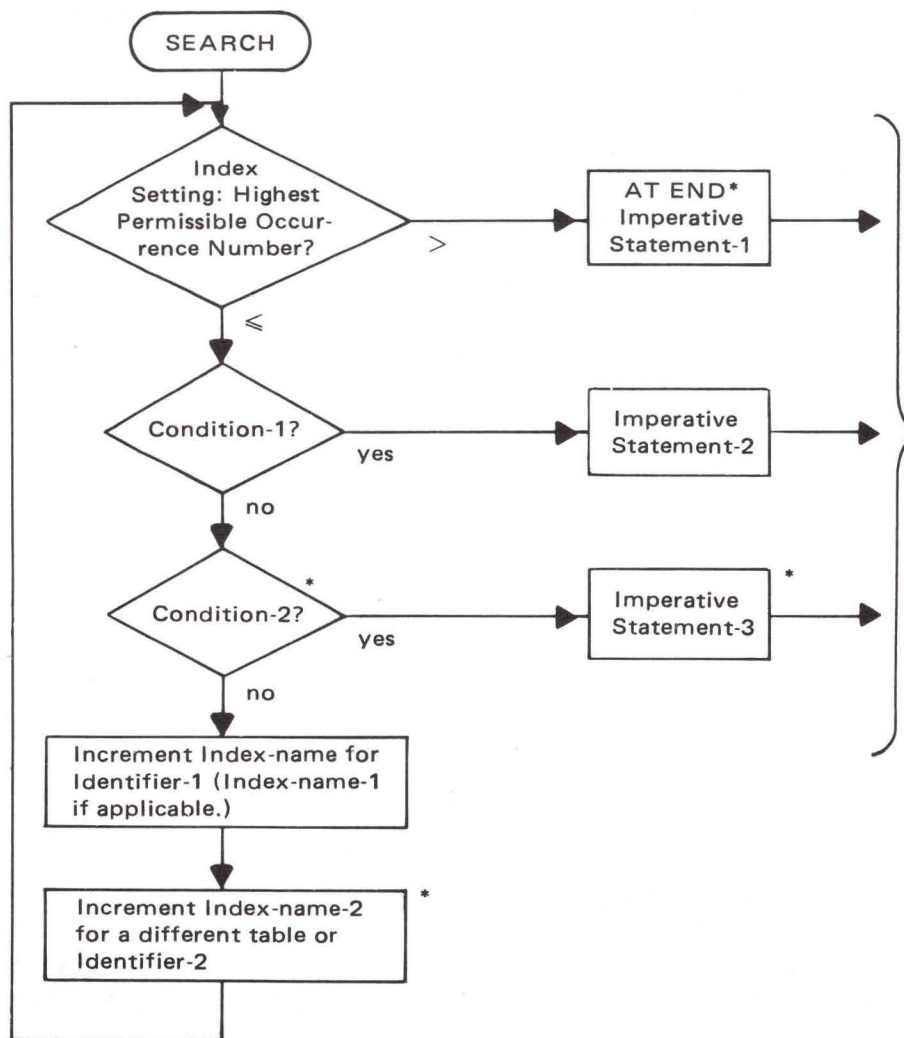
In Format 1 the following action occurs:

1. Index-name-1 is set to a value corresponding to the same occurrence number to which either index-name-2, identifier-2 or literal-1 corresponds. If identifier-2 is an index data item or if index-name-2 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-2 or identifier-2 where the latter is also an index data item; literal-1 cannot be used.

3. If identifier-1 is not an index data item, it may be set only to an occurrence number corresponding to the value of index-name-2; neither identifier-2 nor literal-1 can be used.

In Format 2 the value of index-name-3 is incremented (UP BY) or decremented (DOWN BY) by a value corresponding to the number of occurrences represented by the value of literal-2 or identifier-3.

FIGURE 6-7



* These operations are only included if called for in the statement.

** Each of these operations transfers control to the next sentence unless the statement ends with a GO TO statement.

COMPILER-DIRECTING STATEMENTS

COPY Statement

The format of this statement is:

$\left. \begin{array}{l} \text{\{paragraph-name\}} \\ \text{\{section-name\}} \end{array} \right\} \text{ SECTION copy-statement.}$

The COPY statement incorporates library routines into the PROCEDURE DIVISION of the source program. A library routine, composed of either one paragraph or one section, is a procedure that is stored in a library. The routine is copied from the library during compilation, and the result is the same as if the routine were actually a part of the source program. See Chapter 7, "COBOL Library," for a more detailed description.

When the library routine is composed of one paragraph it is copied into the source program in place of the COPY statement, with the procedure-name of the routine.

When the library routine is composed of one section it is copied into the source program in place of the COPY section, with the section-name of the COPY section automatically replacing the section-name of the section being copied from the library.

CHAPTER 7

COBOL LIBRARY

INTRODUCTION

The COBOL library contains groups of source program card images that are available for inclusion in a COBOL program at compile time. The effect of the compilation of library text is the same as if the text were actually written as part of the source program. The library facility enables standard files, record descriptions, and procedures to be created and made readily accessible to multiple users, thus avoiding duplication of effort and possibilities of error.

Each group of lines, or elements, in the library is a file (in the MDOS sense rather than in COBOL terms) residing on in the user diskette. A library element is incorporated into a source program by the compiler in response to a COPY statement.

The COPY statement causes a search of the same disk containing the COBOL program for a file named "library-name." The file is expected to contain a series of card images that are inserted into the input stream to the compiler immediately following the line containing the COPY request.

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

COPY Statement

The format of this statement is:

COPY library-name

The COPY statement may be written in any of the following forms:

1. In the ENVIRONMENT DIVISION
SOURCE-COMPUTER. Copy-statement.
OBJECT-COMPUTER. Copy-statement.
FILE-CONTROL. Copy-statement.
I-O-CONTROL. Copy-statement.
2. In the FILE SECTION
FD file-name copy-statement.
01 data-name copy-statement.
01 data-name copy-statement.
3. In the WORKING-STORAGE SECTION
01 data-name copy-statement.
4. In the PROCEDURE DIVISION
paragraph-name. SECTION COPY-STATEMENT.
section-name

In case 1 above, the COPY statement is replaced by the information identified by library-name. This information should constitute the entire contents of the appropriate paragraph. In the remaining cases, the entire entry is replaced by the source lines identified by library-name, except that information preceding the COPY statement is not overridden. Thus the original level indicator and (when applicable) data-name, CODE and REDEFINES information are retained.

Examples:

1. FD MASTER-FILE COPY FILEA.

FILEA is the library-name of the COBOL source library element containing a complete File Description entry to be copied into the source program as the description of the file named MASTER-FILE.

2. 01 SUM-DATA COPY SUMMARY-A.

If SUMMARY-A is the name of a library element whose sole contents is a Record Description entry of the form

02 COUNT	PICTURE 9(3).
02 G-TOTAL	PICTURE 9(5)V99.
02 O-TOTAL	PICTURE 9(6)V99.
02 G-DEVIATION	PICTURE 9(4)V99.
02 O-DEVIATION	PICTURE 9(4)V99.

then the data description copied into the source program in place of the line bearing the COPY clause is

01	SUM-DATA.	
02	COUNT	PICTURE 9(3).
02	G-TOTAL	PICTURE 9(5)V99.
02	O-TOTAL	PICTURE 9(6)V99.
02	G-DEVIATION	PICTURE 9(4)V99.
02	O-DEVIATION	PICTURE 9(4)V99.

CHAPTER 8 DEBUGGING

M6800 COBOL supports two debugging features: a paragraph name trace and the EXHIBIT statement. The paragraph name trace is enabled by compiling the program with the trace option. The trace option is described along with the other compiler options in the M6800 COBOL-Operations Reference Manual.

EXHIBIT Statement

The format of this statement is:

```
EXHIBIT [NAMED]      { identifier-1 }      [ identifier-2      ... ]
                    { literal-1   }      [ literal-2      ... ]
```

The execution of the EXHIBIT statement causes the data or literals to be displayed on the printer. The items in the EXHIBIT statement will be separated by blanks when printed. If the NAMED option is used, the data for identifier-n will be preceded by the name of the identifier.

Examples:

EXHIBIT VALUE-A, VALUE-B.

EXHIBIT NAMED VALUE-A, VALUE-B.

EXHIBIT 'VALUES ARE,' VALUE-A, VALUE-B.

EXHIBIT NAMED 'VALUES ARE,' VALUE-A, VALUE-B.

Assuming that VALUE-A contains 123 and VALUE-B contains 'XYZ,' then the data printed by the above statements will be:

123 XYZ VALUE-A 123 VALUE-B XYZ

VALUES ARE 123 XYZ

VALUES ARE VALUE-A 123 VALUE-B XYZ

APPENDIX A. ANS COBOL RESERVED WORDS

ACCEPT	COMPUTATIONAL-3	EVERY
ACCESS	COMPUTE	EXHIBIT
ACTUAL	CONFIGURATION	EXIT
ADD	CONTAINS	EXTEND
ADDRESS	CONTROL	
ADVANCING	CONTROLS	FD
AFTER	COPY	FILE
ALL	CORR	FILE-CONTROL
ALPHABETIC	CORRESPONDING	FILE-LIMIT
ALTER	COUNT	FILE-LIMITS
ALTERNATE	CURRENCY	FILLER
AND		FINAL
ARE	DATA	FIRST
AREA	DATA-COMPILED	FOOTING
AREAS	DATE	FOR
ASCENDING	DATE-WRITTEN	FROM
ASSIGN	DE	
AT	DEBUG-CONTENTS	GENERATE
AUTHOR	DEBUG-ITEM	GIVING
	DEBUG-LINE	GO
BEFORE	DEBUG-NAME	GREATER
BEGINNING	DEBUG-SUB1	GROUP
BLANK	DEBUG-SUB3	
BLOCK	DEBUGGING	HEADING
BOTTOM	DECIMAL-POINT	HIGH-VALUE
BREAK-KEY	DECLARATIVES	HIGH-VALUES
BY	DELIMITED	
	DELIMITER	I-O
CALL	DEPENDING	I-O-CONTROL
CANCEL	DESCENDING	IDENTIFICATION
CF	DETAIL	IF
CH	DISPLAY	IN
CHARACTERS	DIVIDE	INDEX
CLOCK-UNITS	DIVISION	INDEXED
CLOSE	DOWN	INDICATE
COBOL	DUPLICATES	INITIAL
CODE	DYNAMIC	INITIATE
COLUMN		INPUT
COMMA	ELSE	INPUT-OUTPUT
COMP	END	INSPECT
COMP-1	ENDING	INSTALLATION
COMP-2	ENTER	INTO
COMP-3	ENVIRONMENT	INVALID
COMPUTATIONAL	EQUAL	IS
COMPUTATIONAL-1	EQUALS	
COMPUTATIONAL-2	ERROR	JUST

JUSTIFIED	OV	RIGHT
KEY	OVERFLOW	ROUNDED
KEYS		RUN
LABEL	PAGE	SAME
LAST	PAGE-COUNTER	SD
LEADING	PERFORM	SEARCH
LEFT	PF	SECTION
LESS	PH	SECURITY
LIBRARY	PIC	SEEK
LIMIT	PICTURE	SEGMENT-LIMIT
LIMITS	PLUS	SELECT
LINAGE	POINTER	SELECTED
LINAGE-COUNTER	POSITION	SENTENCE
LINE	POSITIVE	SEQUENTIAL
LINE-COUNTER	PROCEDURE	SET
LINES	PROCEDURES	SIGN
LINKAGE	PROCEED	SIZE
LOCK	PROCESSING	SORT
LOW-VALUE	PROGRAM	SOURCE
LOW-VALUES	PROGRAM-ID	SOURCE-COMPUTER
	QUOTE	SPACE
	QUOTES	SPACES
MEMORY		SPECIAL-NAMES
MODE	RANDOM	STANDARD
MODULES	RD	STATUS
MOVE	READ	STOP
MULTIPLE	RECORD	STRING
MULTIPLY	RECORDS	SUBTRACT
	REDEFINES	SUM
NAMED	REEL	SYNC
NEGATIVE	REFERENCES	SYNCHRONIZED
NEXT	RELEASE	
NO	REMAINDER	TALLY
NOT	REMARKS	TALLYING
NUMBER	RENAMES	TAPE
NUMERIC	REPLACING	TERMINAGE
	REPORT	THAN
OBJECT-COMPUTER	REPORTING	THEN
OCCURS	REPORTS	THROUGH
OF	RERUN	THRU
OFF	RESERVE	TIMES
OH	RESET	TO
OMITTED	RETURN	TOP
ON	REVERSED	TYPE
OPEN	REWIND	
OPTIONAL	REWRITE	UNIT
OR	RF	UNSTRING
OUTPUT	RH	UNTIL

UP
UPON
USAGE
USE
USING

VALUE

VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE

WRITE

ZERO
ZEROES
ZEROS

APPENDIX B. SAMPLE M6800 COBOL PROGRAM

PAGE 001 ADD SA:0 APPENDIX B - SAMPLE M6800 COBOL PROGRAM.

```
0010 IDENTIFICATION DIVISION.
0020 *
0030 PROGRAM-ID. ADD
0040 AUTHOR. MOTOROLA MICROSYSTEMS.
0050 DATE-WRITTEN. 03/18/78.
0060 DATE-COMPILED. 03/20/78
0070 REMARKS. THIS IS THE ADD TO INVENTORY MASTER FILE COMMAND.
0080 *
0090 ENVIRONMENT DIVISION.
0100 *
0110 CONFIGURATION SECTION.
0120 SOURCE-COMPUTER. M6800.
0130 OBJECT-COMPUTER. M6800.
0140 INPUT-OUTPUT SECTION.
0150 FILE-CONTROL.
0160     SELECT DATAFILE
0170     ASSIGN TO DISK INV:1
0180     ORGANIZATION IS INDEXED
0190     ACCESS IS RANDOM
0200     RECORD KEY IS PART-NUMBER.
0210 *
0220 DATA DIVISION.
0230 *
0240 FILE SECTION.
0250 FD DATAFILE
0260     LABEL RECORDS ARE OMITTED
0270     DATA RECORD IS MASTER-REC.
0280 COPY MASREC.
0290 *
0300 WORKING-STORAGE SECTION.
0310 77 ERRCK PIC 9 VALUE ZERO.
0320 77 E PIC 99 VALUE ZERO.
0330 77 X PIC 99 VALUE ZERO.
0340 77 Y PIC 99.
0350 77 X1 PIC 99.
0360 77 X2 PIC 99.
0370 77 TAB PIC X VALUE #09.
0380 77 WORK-5 PIC S9(5).
0390 01 FUNMSG LINE IS NEXT PAGE.
0400 02 FILLER PIC X(3) COLUMN 37 VALUE 'ADD'.
0410 *
0420 * DISPLAY ITEM NUMBER REQUEST ON LINE 2 OF SCREEN
0430 01 ITEM-NUM-LINE.
0440 02 FILLER PIC X(11) LINE 2 VALUE 'ITEM NUMBER'.
0450 02 FILLER PIC X(8) COLUMN 15 VALUE SPACES.
0460 01 ANS.
0470 02 ANS-ITEM PIC X(8).
0480 * ERRROW & ERRCOL ARE TABLE POS OF ROW & COL FOR
0490 * CURSOR FOR ERRORS
0500 01 ERRROW.
0510 02 FILLER PIC 99 VALUE 06.
0520 02 FILLER PIC 99 VALUE 08.
0530 02 FILLER PIC 99 VALUE 08.
0540 02 FILLER PIC 99 VALUE 08.
0550 02 FILLER PIC 99 VALUE 10.
0560 02 FILLER PIC 99 VALUE 10.
0570 02 FILLER PIC 99 VALUE 10.
0580 02 FILLER PIC 99 VALUE 12.
```

```

0590 02 FILLER PIC 99 VALUE 12.
0600 02 FILLER PIC 99 VALUE 12.
0610 02 FILLER PIC 99 VALUE 14.
0620 02 FILLER PIC 99 VALUE 14.
0630 01 ROW REDEFINES ERRROW PIC 99 OCCURS 12 TIMES.
0640 01 ERRCOL.
0650 02 FILLER PIC 99 VALUE 73.
0660 02 FILLER PIC 99 VALUE 16.
0670 02 FILLER PIC 99 VALUE 41.
0680 02 FILLER PIC 99 VALUE 71.
0690 02 FILLER PIC 99 VALUE 18.
0700 02 FILLER PIC 99 VALUE 43.
0710 02 FILLER PIC 99 VALUE 72.
0720 02 FILLER PIC 99 VALUE 18.
0730 02 FILLER PIC 99 VALUE 43.
0740 02 FILLER PIC 99 VALUE 73.
0750 02 FILLER PIC 99 VALUE 20.
0760 02 FILLER PIC 99 VALUE 43.
0770 01 COL REDEFINES ERRCOL PIC 99 OCCURS 12 TIMES.
0780 01 ERR-TABLE.
0790 02 FILLER PIC X(20) VALUE 'INVALID ENTRY' <.
0800 02 FILLER PIC X(20) VALUE 'ITEM ALREADY EXISTS' <.
0810 02 FILLER PIC X(20) VALUE ' SUCCESSFUL' <.
0820 02 FILLER PIC X(20) VALUE ' DISK I/O ERROR' <.
0830 01 ERR-MSG REDEFINES ERR-TABLE PIC X(20) OCCURS 4 TIMES.
0840 01 MASTER-AT-SCREEN.
0850 02 FILLER PIC X LINE 2, COLUMN 14; VALUE $EA.
0860 02 FILLER PIC X(11) LINE 5; VALUE 'DESCRIPTION' <.
0870 02 FILLER PIC X(16) COLUMN 18; VALUE SPACES.
0880 02 FILLER PIC X(16) COLUMN 57; VALUE 'LOCATION/BIN' <.
0890 02 FILLER PIC X(5) VALUE SPACES.
0900 02 FILLER PIC X(14) LINE 8; VALUE 'COST' <.
0910 02 FILLER PIC X(7) VALUE SPACES.
0920 02 FILLER PIC X(16) VALUE ' LIST PRICE' <.
0930 02 FILLER PIC X(7) VALUE SPACES.
0940 02 FILLER PIC X(21) VALUE ' TRADE PRICE' <.
0950 02 FILLER PIC X(7) VALUE SPACES.
0960 02 FILLER PIC X(16) LINE 10; VALUE 'QTY ON HAND' <.
0970 02 FILLER PIC X(5) VALUE SPACES.
0980 02 FILLER PIC X(18) VALUE ' QTY ON ORDER' <.
0990 02 FILLER PIC X(5) VALUE SPACES.
1000 02 FILLER PIC X(14) VALUE ' DATE ORDERED' <.
1010 02 FILLER PIC X(8) VALUE '(MMDDYY)' <.
1020 02 FILLER PIC X(6) VALUE SPACES.
1030 02 FILLER PIC X(16) LINE 12; VALUE 'REORDER POINT' <.
1040 02 FILLER PIC X(5) VALUE SPACES.
1050 02 FILLER PIC X(18) VALUE ' STOCKING QTY' <.
1060 02 FILLER PIC X(5) VALUE SPACES.
1070 02 FILLER PIC X(16) VALUE ' QTY/PKG FOR' <.
1080 02 FILLER PIC X(7) VALUE 'REORDER' <.
1090 02 FILLER PIC XXX VALUE SPACES.
1100 02 FILLER PIC X(18) LINE 14; VALUE 'VENDOR/TYPE CODE' <.
1110 02 FILLER PIC XXX VALUE SPACES.
1120 02 FILLER PIC X(18) VALUE ' LEAD TIME' <.
1130 02 FILLER PIC XXX VALUE SPACES.
1140 02 FILLER PIC X(23) COLUMN 52; VALUE 'BACK ORDER FLAG' <.
1150 02 FILLER PIC X VALUE SPACE.
1160 02 FILLER PIC X(13) LINE 17; VALUE 'COMMENT' <.

```

```

1170 02 FILLER PIC X(8) VALUE SPACES.
1180 * DATA FROM SCREEN
1190 01 DATAIN.
1200 02 SDESC PIC X(16).
1210 02 SLOC PIC X(5).
1220 02 SCOST PIC X(7).
1230 02 SLIST PIC X(7).
1240 02 STRADE PIC X(7).
1250 02 SQTY-HAND PIC X(5).
1260 02 SQTY-ORDER PIC X(5).
1270 02 SDATE.
1280 03 SDATE-MO PIC XX.
1290 03 SDATE-DAY PIC XX.
1300 03 SDATE-YR PIC XX.
1310 02 SREORDER PIC X(5).
1320 02 SSTOCKING PIC X(5).
1330 02 SQTY-PER-PK PIC XXX.
1340 02 SVEND PIC XXX.
1350 02 SLEAD PIC XXX.
1360 02 SBACK-O-FLAG PIC X.
1370 02 SCOMMENT PIC X(8).
1380 01 BLINK-OFF.
1390 * LOC
1400 02 FILLER PIC X LINE 6, COLUMN 73; VALUE #E3.
1410 * COST
1420 02 FILLER PIC X LINE 8 COLUMN 16; VALUE #E3.
1430 * LIST PRICE
1440 02 FILLER PIC X COLUMN 41; VALUE #E3.
1450 * TRADE PRICE
1460 02 FILLER PIC X COLUMN 71; VALUE #E3.
1470 * QTY ON HAND
1480 02 FILLER PIC X LINE 10, COLUMN 18; VALUE #E3.
1490 * QTY ON ORDER
1500 02 FILLER PIC X COLUMN 43; VALUE #E3.
1510 * DATE
1520 02 FILLER PIC X COLUMN 72; VALUE #E3.
1530 * QTY FOR RE-ORDER
1540 02 FILLER PIC X LINE 12, COLUMN 18; VALUE #E3.
1550 * STOCKING QTY
1560 02 FILLER PIC X COLUMN 43; VALUE #E3.
1570 * QTY PER PACK
1580 02 FILLER PIC X COLUMN 73; VALUE #E3.
1590 02 FILLER PIC X LINE 14 COLUMN 20 VALUE #E3.
1600 02 FILLER PIC X COLUMN 43 VALUE #E3.
1610 * ERASE LINE 24
1620 02 FILLER PIC X LINE 24 COLUMN 3; VALUE #D5.
1630 *****
1640 PROCEDURE DIVISION.
1650 AA-DRIVER-SECTION.
1660 *
1670 * THIS SECTION WILL DO INITIALIZATION/OPEN/CLOSE
1680 * PERFORM PROCESSING ROUTINES & WRAP UP
1690 *
1700 OPEN I-O DATAFILE.
1710 * FUNCTION TYPE MSG
1720 DISPLAY FUNMSG.
1730 * GET PART NUM
1740 GET-ITEM.

```

```

1750     MOVE ZERO TO ERRCK Y.
1760     DISPLAY ITEM-NUM-LINE.
1770     ACCEPT ANS.
1780 *
1790     PERFORM BA-READ-RECORD THRU BA-READ-RECORD-EXIT.
1800     IF ERRCK EQUAL 1 GO TO GET-ITEM.
1810 *     DISPLAY BUILD REC SCREEN
1820     DISPLAY MASTER-AT-SCREEN.
1830     PERFORM BB-DATA-EDIT THRU BB-DATA-EDIT-EXIT.
1840 *
1850     PERFORM BC-DATA-UPDATE THRU BC-DATA-UPDATE-EXIT.
1860 *
1870     CLOSE DATAFILE.
1880     STOP RUN.
1890 AA-DRIVER-EXIT.     EXIT.
1900 *
1910 BA-READ-RECORD.
1920 *
1930 *     THIS ROUTINE WILL VALIDATE THAT THE PART IS IN THE FILE
1940 *
1950     DISPLAY @(24,2) #D5.
1960     MOVE ANS-ITEM TO PART-NUMBER.
1970     READ DATAFILE
1980         INVALID KEY GO TO BA-READ-RECORD-EXIT.
1990 * RECORD ALREADY PRESENT
2000     MOVE 2 TO E.
2010     MOVE 1 TO ERRCK.
2020     PERFORM MSG-PRT THRU MSG-PRT-EXIT.
2030 BA-READ-RECORD-EXIT.     EXIT.
2040 *
2050 *
2060 BB-DATA-EDIT.
2070 *
2080 *     THIS ROUTINE WILL READ SCREEN & VALIDATE DATA FIELDS
2090 *
2100     ACCEPT DATAIN.
2110     DISPLAY BLINK-OFF.
2120     MOVE ZERO TO ERRCK Y.
2130 * EDIT LOCATION
2140     MOVE 1 TO X.
2150     MOVE SLOC TO WORK-5 ON SIZE ERROR
2160         PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2170     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2180 * EDIT COST
2190 CK-COST.
2200     MOVE SCOST TO COST
2210         ON SIZE ERROR GO TO COST-ERR.
2220     IF COST IS NUMERIC GO TO CK-LIST.
2230 COST-ERR.
2240     MOVE 2 TO X.
2250     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2260 CK-LIST.
2270 * VALIDATE LIST PRICE
2280     MOVE SLIST TO LIST-PRICE
2290         ON SIZE ERROR GO TO LIST-ERR.
2300     IF LIST-PRICE IS NUMERIC GO TO CK-TRADE.
2310 LIST-ERR.
2320     MOVE 3 TO X.

```

```
2330     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2340 CK-TRADE.
2350 *     VALIDATE TRADE PRICE
2360     MOVE STRADE TO TRADE-PRICE
2370     ON SIZE ERROR GO TO TRADE-ERR.
2380     IF TRADE-PRICE IS NUMERIC GO TO CK-QTY-HAND.
2390 TRADE-ERR.
2400     MOVE 4 TO X.
2410     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2420 CK-QTY-HAND.
2430 *     VALIDATE QTY ON HAND
2440     MOVE 5 TO X.
2450     MOVE SQTY-HAND TO WORK-5 ON SIZE ERROR
2460     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2470     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2480 *     VALIDATE QTY-ON-ORDER
2490 CK-ORD.
2500     MOVE 6 TO X.
2510     MOVE SQTY-ORDER TO WORK-5 ON SIZE ERROR
2520     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2530     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2540 *     DATE EDIT
2550 CK-DATE.
2560     IF SDATE EQUAL SPACES
2570     MOVE ZERO TO ORDER-MONTH ORDER-DAY ORDER-YEAR
2580     GO TO CK-REORD.
2590     IF SDATE IS NOT NUMERIC GO TO DATE-ERR.
2600     MOVE SDATE TO ORDER-DATE.
2610     IF ORDER-MONTH GREATER THAN 12 OR LESS THAN ZERO
2620     GO TO DATE-ERR.
2630     IF ORDER-DAY IS GREATER THAN 31 GO TO DATE-ERR.
2640     IF ORDER-YEAR IS GREATER THAN 76 GO TO CK-REORD.
2650 DATE-ERR.
2660     MOVE 7 TO X.
2670     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2680 CK-REORD.
2690 *     EDIT REORDER POINT
2700     MOVE 8 TO X.
2710     MOVE SREORDER TO WORK-5 ON SIZE ERROR
2720     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2730     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2740 *     EDIT STOCKING QTY
2750 CK-STOCK.
2760     MOVE 9 TO X.
2770     MOVE SSTOCKING TO WORK-5 ON SIZE ERROR
2780     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2790     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2800 *     QTY PER PACKAGE
2810 CK-PACK.
2820     MOVE 10 TO X.
2830     MOVE SQTY-PER-PK TO WORK-5 ON SIZE ERROR
2840     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2850     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2860 *     EDIT VENDOR CODE
2870 CK-VEND.
2880     MOVE 11 TO X.
2890     MOVE SVEND TO WORK-5 ON SIZE ERROR
2900     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
```



```

2910     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2920 *   EDIT LEAD TIME
2930 CK-LEAD.
2940     MOVE 12 TO X.
2950     MOVE SLEAD TO WORK-5 ON SIZE ERROR.
2960     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2970     PERFORM EDIT-CK THRU EDIT-CK-EXIT.
2980 *
2990 CK-ANY-ERR.
3000     IF ERRCK EQUAL 1
3010         MOVE 1 TO E
3020         PERFORM MSG-PRT THRU MSG-PRT-EXIT
3030         PERFORM TAB-IT THRU TAB-IT-EXIT Y TIMES
3040     ELSE
3050         MOVE SLOC TO LOCATION-BIN
3060         MOVE SQTY-HAND TO QTY-ON-HAND
3070         MOVE SQTY-ORDER TO QTY-ON-ORDER
3080         MOVE SREORDER TO REORDER-POINT
3090         MOVE SSTOCKING TO STOCKING-QTY
3100         MOVE SQTY-PER-PK TO QTY-PER-PACK
3110         MOVE SVEND TO VENDOR-CODE
3120         MOVE SLEAD TO LEAD-TIME
3130         MOVE SDESC TO DESCRIPTION
3140         MOVE SBACK-O-FLAG TO BACK-ORDER-IND
3150         MOVE ZERO TO ISS-MONTH-1 ISS-MONTH-2 ISS-MONTH-3
3160         MOVE ZERO TO ISS-QUARTER-1 ISS-QUARTER-2 ISS-QUARTER-3
3170         MOVE SCOMMENT TO COMMENT.
3180 BB-DATA-EDIT-EXIT.     EXIT.
3190 *
3200 EDIT-CK.
3210     IF WORK-5 EQUAL ZERO GO TO EDIT-CK-EXIT.
3220     IF WORK-5 NOT NUMERIC GO TO EDIT-ERR.
3230     IF WORK-5 POSITIVE GO TO EDIT-CK-EXIT.
3240 EDIT-ERR.
3250     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
3260 EDIT-CK-EXIT.     EXIT.
3270 *
3280 ERR-BLINK.
3290 *   ROUTINE TO BLINK FIELD IN ERROR
3300 *
3310     DISPLAY @(ROW(X),COL(X)) #E2.
3320     IF Y EQUAL ZERO MOVE X TO Y.
3330     MOVE 1 TO ERRCK.
3340 ERR-BLINK-EXIT.     EXIT.
3350 *
3360 *
3370 BC-DATA-UPDATE.
3380 *
3390 *   THIS ROUTINE WILL WRITE THE MASTER TO DISK
3400 *
3410     WRITE MASTER-REC INVALID KEY
3420         MOVE 4 TO E
3430         PERFORM MSG-PRT THRU MSG-PRT-EXIT
3440         GO TO BC-DATA-UPDATE-EXIT.
3450 *   SUCCESSFUL MSG
3460     MOVE 3 TO E.
3470     PERFORM MSG-PRT THRU MSG-PRT-EXIT.
3480 BC-DATA-UPDATE-EXIT.     EXIT.

```

PAGE 007 ADD SA:0 APPENDIX B - SAMPLE M6800 COBOL PROGRAM.

```
3490 MSG-PRT.  
3500 *  
3510 * ROUTINE TO PRINT MESSAGES ON SCREEN  
3520 *  
3530 DISPLAY @(24,4) $EAE0C4 ERR-MSG(E).  
3540 MSG-PRT-EXIT. EXIT.  
3550 *  
3560 TAB-IT.  
3570 DISPLAY TAB.  
3580 TAB-IT-EXIT. EXIT.
```

PAGE 001 MASREC SA:0 APPENDIX B - SAMPLE M6800 COBOL PROGRAM.

```
0010 01 MASTER-REC.  
0015 02 PART-NUMBER PIC X(8).  
0020 02 DESCRIPTION PIC X(16).  
0030 02 LOCATION-BIN PIC 9(5).  
0040 02 COST PIC 9(4)V99.  
0050 02 LIST-PRICE PIC 9(4)V99.  
0060 02 TRADE-PRICE PIC 9(4)V99.  
0070 02 QTY-ON-HAND PIC 9(5).  
0080 02 QTY-ON-ORDER PIC 9(5).  
0090 02 QTY-PER-PACK PIC 999.  
0100 02 REORDER-POINT PIC 9(5).  
0110 02 STOCKING-QTY PIC 9(5).  
0120 02 VENDOR-CODE PIC 999.  
0130 02 LEAD-TIME PIC 999.  
0140 02 ORDER-DATE.  
0150 03 ORDER-MONTH PIC 99.  
0160 03 ORDER-DAY PIC 99.  
0170 03 ORDER-YEAR PIC 99.  
0180 02 ISSUE-COUNT.  
0190 03 ISS-MONTH-1 PIC 9(5).  
0200 03 ISS-MONTH-2 PIC 9(4).  
0210 03 ISS-MONTH-3 PIC 9(4).  
0220 03 ISS-QUARTER-1 PIC 9(4).  
0230 03 ISS-QUARTER-2 PIC 9(4).  
0240 03 ISS-QUARTER-3 PIC 9(4).  
0250 02 BACK-ORDER-IND PIC X.  
0260 02 COMMENT PIC X(8).
```


-COBOL SOFTWARE PROBLEM REPORT-

DATE:

NAME:

ADDRESS:

PHONE NUMBER:

PROBLEM DESCRIPTION:

RETURN THIS FORM TO MOTOROLA MICROSYSTEMS; PHOENIX, ARIZONA.

INCLUDE A LISTING OF THE PROGRAM (S OR L OPTION)
ALONG WITH A DUMP OF THE GENERATED OBJECT (LO FILE) MODULE.

ALSO INCLUDE ANY OTHER INFORMATION THAT MAY BE APPROPRIATE
TO THE SOLUTION OF THE PROBLEM.

APPENDIX C. INTERNAL DATA TYPES

1. PICTURE X—data is stored as a series of 8-bit ASCII coded bytes. Normally, the most significant is zero, but the full 8 bits may contain information when the data is initialized with a hexadecimal constant.
2. PICTURE 9 USAGE DISPLAY—data is stored as a series of ASCII coded numbers up to a maximum of 15 digits. A negative value will have the sign bit (most significant bit) set on the left-most digit.
3. PICTURE 9 USAGE COMPUTATIONAL—data is stored in packed decimal; two digits per byte. A signed number will have a sign as the first half byte. A zero represents a positive number and a hexadecimal “F” represents a negative number. If necessary, the number of digits specified by the picture clause will be increased by one to force an integral number of bytes to be allocated for the data.