Xerox ANS COBOL (for BPM/CP-V)

Xerox 550 560 and Sigma 5-9 Computers

Language
Reference Manual

90 15 00 D

XEROX

Xerox ANS COBOL (for BPM/CP-V)

Xerox 550/560 and Sigma 5-9 Computers

Language
Reference Manual

90 15 00D

May 1976

REVISION

This edition merely incorporates the 90 15 00C-1 revision package into the manual. It documents the E07 version of the software.

RELATED PUBLICATIONS

Title	Publication No.
Xerox 550 Computer/Reference Manual	90 30 77
Xerox 560 Computer/Reference Manual	90 30 76
Xerox Sigma 5 Computer/Reference Manual	90 09 59
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 8 Computer/Reference Manual	90 17 49
Xerox Sigma 9 Computer/Reference Manual	90 17 33
Xerox Batch Processing Monitor (BPM)/BP, RT Reference Manual	90 09 54
Xerox Batch Processing Monitor (BPM)/OPS Reference Manual	90 11 98
Xerox Control Program-Five (CP-V)/TS Reference Manual	90 09 07
Xerox Control Program-Five (CP-V)/OPS Reference Manual	90 16 75
Xerox Control Program-Five (CP-V)/TS User's Guide	90 16 92
Xerox Control Program-Five (CP-V)/SM Reference Manual	90 16 74
Xerox ANS COBOL (for CP-V/BPM)/OPS Reference Manual	90 15 01
Xerox ANS COBOL On-Line Debugger Reference Manual	90 30 60
Xerox Sort and Merge (for CP-V/BPM)/Reference Manual	90 11 99
Xerox Data Management System (DMS) (for BPM/CP-V)/Reference Manual	90 17 38
Xerox Extended Data Management System (EDMS)/Reference Manual	90 30 12

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RP - remote processing, RT - real-time, SM - system management, SP - system programming, TP - transaction processing, TS - time-sharing, UT - utilities.

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their Xerox sales representative for details.

CONTENTS

APPENDIXES

FIGURES

TABLES

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 1965.

"Any organization interested in reproducing the COBOL report and specifications[t], 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[tt] 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."

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

[tt] 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.

PREFACE

Xerox ANS 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 - 1968. Certain features from the X3.23 1974 American National Standard Specifications for COBOL which were not a part of the 1968 standard are also included.

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 Xerox ANS COBOL, which is compatible with American National Standard COBOL and includes a number of Xerox extensions to it as well. The compiler supports the processing modules defined in the standard. These processing modules include the following ten:

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 and indexes. A convenient method for searching a table is provided.

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.

SORT provides the capability of sorting files in ascending and/or descending order according to a set of user-specified keys within each record. Optionally, a user may apply some special processing which may consist of addition, deletion, creation, altering, editing or other modification of the individual records by input or output procedures.

REPORT WRITER allows the programmer to describe the format of a report in the DATA DIVISION, thereby minimizing the amount of PROCEDURE DIVISION coding necessary.

SEGMENTATION allows the programmer to specify program overlay requirements. This enables large problem programs to be split into segments that can then be designated as permanent or overlayable core storage, which assures more efficient use of core storage at object time.

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.

DEBUG allows the programmer to describe a debugging algorithm including the conditions under which data items or procedures are to be monitored during the execution of the object program.

INTER-PROGRAM COMMUNICATION provides a facility by which a program can communicate with one or more separately compiled programs.

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, Xerox 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 is a detailed description of the Report Writer, which has associated information that is necessarily contained in more than one division.
- Chapter 8 describes the Sort feature. This feature also requires information contained in several divisions for its implementation.
- Chapter 9 contains a description of the statements that affect the COBOL library.
- Chapter 10 presents information concerning Inter-Program Communication.
- Chapter 11 describes the COBOL Priority Segmentation feature.
- Chapter 12 is a discussion of the facilities provided for debugging COBOL programs.

The appendixes contain supplementary information: a list of Xerox ANS COBOL reserved words; a sample Xerox ANS COBOL problem; a description of slack bytes; an evaluation of arithmetic expressions; sample programs illustrating the use of the Report Writer and Sort feature; a list of Xerox ANS COBOL compiler diagnostics.

Xerox Extensions to the ANS COBOL Standard

Listed below are Xerox extensions to the ANS COBOL Standard (both the 1968 and the 1974 versions). 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 generally accepted industry usage.

Language Concepts

Apostrophe is used as a default value for quotation mark.

Identification Division

None.

Environment Division

Use of "integer" in the ASSIGN clause of the SELECT statement.

Data Division

Use of COMPUTATIONAL-1, -2, and -3 formats.
COMMON-STORAGE SECTION.

Procedure Division

ENTER statement formats 2 and 3.

Declaratives

PROCEDURE option of the USE FOR DEBUGGING statement.

Report Writer

Overflow headings and footings.

Debug Features

On-line debugger.
TRACE and EXHIBIT debugging statements.

COMMAND SYNTAX NOTATION

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

Notation	Description
lowercase letters	Lowercase letters identify an element that must be replaced with a user-selected value. CRndd could be entered as CRA03.
CAPITAL LETTERS	Capital letters must be entered as shown for input, and will be printed as shown in output. DPndd means "enter DP followed by the values for ndd".
[ ]	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. [KEYM] means the term "KEYM" may be entered.
{ }	Elements placed one under the other inside a pair of braces identify a required choice. $\begin{Bmatrix} \text{A} \\ \text{id} \end{Bmatrix}$ means that either the letter A or the value of id must be entered.
. . .	The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted. name[, name]. . . means that one or more name values may be entered, with a comma inserted between each name value.
⋮	The vertical ellipsis indicates that commands or instructions have been omitted. MASK2 DATA, 2 X'1EF' ⋮ BYTE DATA, 3 BA(L(59)) means that there are one or more statements omitted between the two DATA directives.
Numbers and special characters	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. (value) means that the proper value must be entered enclosed in parentheses; e.g., (234).
Subscripts	Sybscripts indicate a first, second, etc., representation of a parameter that has a different value for each occurrence. sysid$_1$, sysid$_2$, sysid$_3$ means that three successive values for sysid should be entered, separated by commas.

1. COBOL LANGUAGE STRUCTURE

Introduction

COBOL (the COmmon Business Oriented Programming Language) 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 Xerox ANS COBOL consists of the 51 characters listed below.

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

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. A space must follow a period, comma, or semicolon when one of these punctuation characters is used to terminate a word.
2. A space must not immediately follow a left parenthesis or immediately precede a right parenthesis.
3. 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, "Xerox ANS COBOL Reserved Words".)

Condition-Name

A condition-name is one assigned to a specific value, set of values, or range of values within the complete set of values that a data-name may assume. The condition-name must contain at least one alphabetic character and must be unique or made unique through qualification. The data-name itself is called a conditional variable. A conditional variable may be used as a qualifier for any of its condition-names. If references to a conditional variable require indexing, subscripting, or qualifying, references to any of its condition-names also require the same combination of indexing, subscripting, or qualifying. (Refer to "Uniqueness of Data Reference" later in this chapter.)

In addition to being described in the DATA DIVISION, condition-names may also be defined in the SPECIAL-NAMES paragraph within the ENVIRONMENT DIVISION, where a condition-name can be associated with either or both the ON or OFF status of hardware devices.

A condition-name is also used in place of the "equal" relational operator in forming conditional tests. Such tests determine whether a given conditional variable is equal to one of the set of values to which the condition-name is assigned.

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 Extended Binary Coded Decimal Interchange Code (EBCDIC) characters (including reserved words but excluding the quotation mark character) up to 254 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 30 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. If the literal is used as a numeric operand in an arithmetic operation, it may not exceed 18 digits in length. 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 are:

ZERO
ZEROS
ZEROES — Represents the value 0, or one or more of the character 0, depending on context.

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 Sigma collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more characters that have the lowest value in the Sigma 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, the only figurative-constant permitted is ZERO (ZEROS, ZEROES).

Special Registers

DEBUG-ITEM

DEBUG-ITEM, is the name for a special register that is generated automatically by the COBOL run-time package that supports the debugging facility.

LINE-COUNTER

LINE-COUNTER is the fixed data-name for a line-counter used by the REPORT SECTION in the DATA DIVISION to generate automatically PAGE HEADING and PAGE FOOTING report groups. One line-counter is automatically supplied for each report described in the REPORT SECTION if a PAGE LIMIT clause is written in the Report Description entry. LINE-COUNTER is represented as one binary word.

PAGE-COUNTER

PAGE-COUNTER is a fixed data-name for a page-counter generated by the REPORT SECTION for use as a source data item to present page numbers within a report group. One page-counter is supplied for each report by the REPORT SECTION if the word PAGE-COUNTER is given as a source data item in the Report Group Description entry. PAGE-COUNTER is represented as one binary word.

Special Names

Special names provide a means of relating implementor-names with problem-oriented names and the status of hardware switches with condition-names. (Refer to "SPECIAL-NAMES Paragraph" in Chapter 4.)

Reserved Words

Reserved words are used for syntactical purposes and cannot appear as user-defined words. (See Appendix A, "Xerox 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, ENTER.
2. Required words (in statement and entry formats) such as TO and GIVING.
3. Words that have a specific functional meaning such as NEGATIVE, SECTION, TALLY, 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 and common storage. Thus, working-storage and common-storage 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 49. Special level-numbers 66, 77, and 88 are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. The level-number of an item (either 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.

Three types of data exist for which there is no true concept of level. These are

1. Names of elementary items introduced by a RENAMES clause.

2. Noncontiguous working-storage and common-storage items.

3. Entries that specify condition-names.

Data-names described by a RENAMES clause for the purpose of regrouping data items are assigned the special level-number 66.

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

Entries that specify condition-names to be associated with particular values of a conditional variable are assigned the special level-number 88.

Initial Values of Tables

In the WORKING-STORAGE and COMMON-STORAGE SECTIONs, initial values of elements within tables are specified in one of the following ways:

1. 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 due to synchronization, usage, etc. 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.

2. When the elements of the table do not require separate handling, the value of the entire table may be given in the entry defining the entire table. The lower level entries show the hierarchical structure of the table; they must not contain VALUE clauses.

Initial values may not be specified for items appearing in a LINKAGE SECTION.

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

Qualification

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because the name exists within a hierarchy of names so that it can be made unique by mentioning one or more of the higher hierarchical levels. The higher levels are called qualifiers; the process that specifies uniqueness is called qualification. Enough qualification must be specified, if needed, to make the name unique; however, it may not be necessary to mention all levels of the hierarchy.

The name associated with a level indicator (FD, SD, and RD) is the highest level qualifier available for a data-name. A section-name is the highest (and only) qualifier available for a paragraph-name. Thus, level indicator names and section-names must be unique in themselves, as they cannot be qualified. In the WORKING-STORAGE, COMMON-STORAGE and LINKAGE SECTIONs, data-names associated with 77 levels and 01 levels must be unique. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, can be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and procedure-name.

Qualification is accomplished by following a data-name or a paragraph-name with one or more phrases composed of a qualifier preceded by IN or OF. (IN and OF are logically equivalent.) Rules for qualification are

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

2. The same name cannot appear at two levels in a hierarchy so that the name appears to qualify itself.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referenced in the PROCEDURE, ENVIRONMENT, and DATA DIVISIONs (except REDEFINES where, by definition, qualification is unnecessary).

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referenced within the same section.

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

6. A name can be qualified even though it does not need qualification. If there is more than one combination of qualifiers that ensures uniqueness, any set can be used.

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

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

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

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

Restrictions on Indexing, Subscripting, and Qualification

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, or qualifier.

When qualification, subscripting, or indexing is required for a given data item, the indexes or subscripts are stated after all necessary qualification is given.

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, page x.

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. Refer to the example below.
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 ($\overset{\pm}{n}$) 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 $\boxed{A}\boxed{B}\boxed{C}\boxed{D}$. 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.

$$\underline{\text{SUBTRACT}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \left[\begin{matrix}\text{, identifier-2}\\ \text{, literal-2}\end{matrix}\right] \ldots \ \underline{\text{FROM}}\ \text{identifier-m}\ [\underline{\text{ROUNDED}}]$$

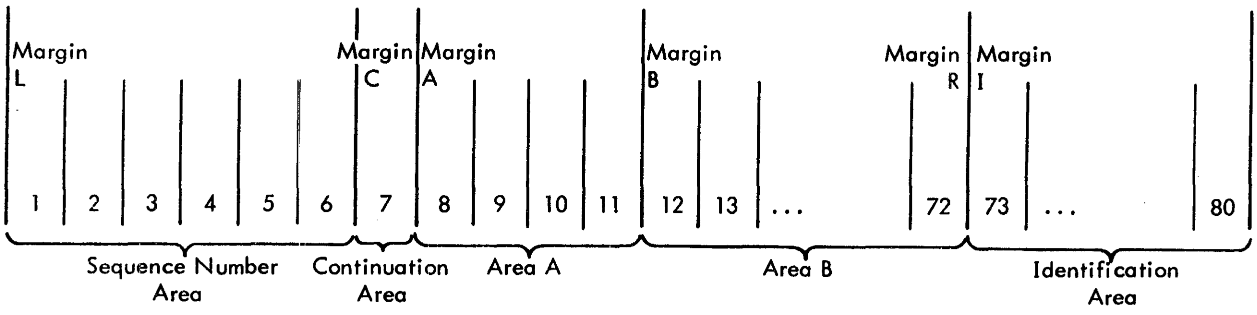
Reference Format

General Description

The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions or columns on an 80-column card. 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.

Reference Format Representation



where

Margin L designates the leftmost character position of a line, column 1.

Margin C designates the seventh character position relative to Margin L, column 7.

Margin A designates the eighth character position relative to Margin L, column 8.

Margin B designates the twelfth character position relative to Margin L, column 12.

Margin R designates the rightmost character position of a line, column 72.

Margin I designates the seventy-third character position relative to Margin L, column 73.

Sequence Number Area occupies the six character positions beginning at Margin L, columns 1 through 6.

Continuation Area occupies one character position beginning at Margin C, column 7.

Area A occupies four character positions beginning at Margin A, columns 8 through 11.

Area B occupies sixty-one character positions beginning at Margin B, columns 12 through 72.

Identification Area occupies eight character positions beginning at Margin I, columns 73 through 80.

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

Format Control. Margin C (column 7) may be used for format control.

* means to treat the line as comments.

/ means to start printing the source listing on a new page.

Thus, an asterisk in column 7 allows interspersing comments throughout the COBOL program. Its function is similar to that of the NOTE statement in the Procedure Division. A slash in column 7 forces a page ejection on the source listing.

Continuation of Lines. Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B, thus forming continuation line(s). The line being continued is called the continued line. Any word or literal may be broken so that part of it appears on a continuation line.

A hyphen in the Continuation Area of the continuation line indicates that the first nonblank character in Area B of the continuation line is the continuation of the last word written in the continued line. If a hyphen does not appear in the Continuation Area and the sentence or entry starts in Area B, it is assumed that the last character in the continued line is followed by a space.

Continuation of Non-Numeric Literals. When a non-numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area 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.

Continuation of Words and Numeric Literals. When a word or numeric literal is continued from one line to another, a hyphen in the Continuation Area of the continuation line indicates that the first nonblank character in Area B of the continuation line is to follow the last nonblank character on the continued line without an intervening space.

Program Identification. Columns 73 through 80 can be used to identify the program. Any character may be used. The program identification code has no effect on the compiler or the object program.

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 Area 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, a space followed by a priority number (optional), and a period followed by a space. No other text may appear on the same line as the section header except in the DECLARATIVES portion of the PROCEDURE DIVISION, in which case USE and INCLUDE sentences may begin 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 as described in "Continuation of Lines" above.

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 (except in the REPORT SECTION), 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, SD, and RD are level indicators. In DATA DIVISION entries that begin with a level indicator, the level indicator begins in Area A, followed by its associated file-name or report-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 49, 66, 77, 88. 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.

PROCEDURE DIVISION Declaratives. The key words DECLARATIVES and END DECLARATIVES that precede and follow, respectively, the DECLARATIVES portion of the PROCEDURE DIVISION must each appear on a line by itself. Each begins in Area A followed by a period and a space. Refer to "PROCEDURE DIVISION Structure" in Chapter 6.

2. COBOL INPUT/OUTPUT PROCESSING

Monitor-Formatted Files

Xerox ANS COBOL supports the file organizations, record formats, and access methods provided by Xerox Batch Processing Monitor (BPM) and Control Program-Five (CP-V) operating systems.

The COBOL term "file" can be considered equivalent in meaning to the use of the word "file" in the operating system publications. A file is created upon being opened for output. Each file defined in a COBOL program is described to the operating system by a Data Control Block (DCB) which consists of a group of contiguous fields that supply information about the file to the operating system, for the purpose of scheduling and executing input/output operations. These fields describe the characteristics of the file (e.g., its organization) and its processing requirements (e.g., input or output). The COBOL compiler creates a "skeletal" DCB for each file defined in the source program and places in it pertinent information provided by the ENVIRONMENT DIVISION and the File Description (FD) entry. The name of the DCB is the file-name specified in the FD entry, prefixed by the two characters "F:", denoting a user DCB to the operating system. The DCBs for the files are part of the object program module produced by the compilation of the COBOL program.

At execution time, the ASSIGN control command and the file label contribute additional information to the DCB. This information may replace and override information originallyprovided by the compiler. System control commands are not required for COBOL files assigned to card punch, card reader, or printer (see ASSIGN Clause, Chapter 4), but are required for all other files.

File Organization

There are three types of COBOL file organizations: sequential, relative, and indexed. In the operating system reference manuals, these organizations are referred to as consecutive, random, and keyed, respectively.

Sequential File Organization

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

Relative File Organization

A relative file is one in which records are accessed by using a one-word binary key containing the logical ordinal position of the record in the file. Relative file organization is indicated in the COBOL language by the use of DISC-R in the ASSIGN clause of the SELECT statement. Only random access is allowed with this type of file organization.

Indexed File Organization

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

File Access

The two methods of accessing files are (1) sequential and (2) random access. In the operating system reference manuals, these access methods are termed sequential and direct, respectively.

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. 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 and the ACTUAL KEY clause specifies the key.

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.

Access Mode is Sequential

1. OPEN OUTPUT. This combination creates a sequential file.

2. OPEN INPUT. If the file organization is sequential, READ statements obtain 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 the order in which they were written).

3. OPEN INPUT-OUTPUT. This OPEN statement allows a WRITE statement to follow a READ statement. This replaces the record just read if the file organization is indexed. If the file organization is sequential, the remainder of the file is deleted.

Access Mode is Random

1. OPEN OUTPUT. This combination creates an indexed file. An ACTUAL KEY must be specified and its contents consulted upon each WRITE statement.

2. OPEN INPUT. Organization of the file must be indexed. An ACTUAL 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.

File Labels

All Xerox monitor-formatted disk and tape files are labeled. The Xerox monitor label contains such information as which password account numbers are permitted to access the file, whether the file is keyed or consecutive, etc. ANS standard tapes also have labels.

In addition to the standard label the user may, at his own discretion, write additional tape file labels that he himself specifies. The format of the user label is specified through the LABEL RECORD clause in the FD entry. The label is created (when writing the file) or checked (when reading) through a DECLARATIVES section associated with the appropriate USE statement.

Input/Output Processing Summary

Table 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 1. File Manipulation Statements

File Organization	ACCESS MODE IS	Type of OPEN Statement	Permissible I/O Statement	ACTUAL KEY Required	Required Clauses	Optional Clauses
Sequential	SEQUENTIAL (or unspecified)	INPUT [REVERSED]	READ... AT END	No	LABEL RECORD	DATA RECORDS ASSIGN RESERVE SAME RECORD AREA RERUN MULTIPLE FILE TAPE USE (Format 1) USE (Format 2) (if file is labeled) USE (Format 3) (if a report file) BLOCK CONTAINS
		OUTPUT	WRITE... [{BEFORE / AFTER} [ADVANCING]] WRITE... INVALID KEY	No	REPORTS ARE (if a report file) LABEL RECORD	
Indexed	SEQUENTIAL (or unspecified)	INPUT [REVERSED]	READ... AT END	No	LABEL RECORD	ASSIGN SAME RECORD AREA RERUN PROCESSING MODE USE (Format 1) USE (Format 2) (if file is labeled)
Indexed/ Relative	RANDOM	INPUT	READ... INVALID KEY	Yes	LABEL RECORD	ASSIGN SAME RECORD AREA PROCESSING MODE USE (Format 1) USE (Format 2) (if file is labeled) BLOCK CONTAINS
		INPUT-OUTPUT	READ... INVALID KEY WRITE... INVALID KEY	Yes		
		OUTPUT	WRITE... INVALID KEY	Yes		

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 provides the compilation date in the source program listing. During program compilation, the current date replaces the comment-sentences in the paragraph.

Example:

The IDENTIFICATION DIVISION of a typical program might be written

```
IDENTIFICATION DIVISION.

PROGRAM-ID. GOOD-NEWS.

AUTHOR. XEROX CORPORATION.

DATE-WRITTEN. MAY 6 1975.

DATE-COMPILED. JUNE 21 1975.

REMARKS. LISTING IS PRINTED USING FIRST RECORD DIGIT AS PAGE CONTROL.
```

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.
[SPECIAL-NAMES. special-names 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 DIVISONs 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 three paragraphs: the SOURCE-COMPUTER paragraph describing the computer on which the COBOL compiler is to run; the OBJECT-COMPUTER paragraph defining the computer on which the translated program is to run; and the SPECIAL-NAMES paragraph relating implementor-names used by the compiler to mnemonic-names used by the source program.

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 logical points at which the object program should checkpoint itself and 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 [WITH DEBUGGING MODE].

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 9 for a complete description of the COBOL library. If the WITH DEBUGGING MODE clause is specified, all USE FOR DEBUGGING statements and debugging lines are compiled.

OBJECT-COMPUTER Paragraph

The formats of this paragraph are

Format 1

```
    OBJECT-COMPUTER.  copy-statement.
```

Format 2

```
    OBJECT-COMPUTER.
                   ┌                    ┌          ┌WORDS     ┐                                                   ┐┐
                   │                    │  integer │CHARACTERS│                                                   ││
    computer-name  │ , MEMORY SIZE      │          └MODULES   ┘                                                   ││
                   │                    │                                                                       ││
                   └                    └ ADDRESS literal-2 THRU literal-3 [, literal-4 THRU literal-5]...      ┘┘
                   [, [literal-6] implementor-name-1] ...

                   [,  SEGMENT-LIMIT IS priority-number].
```

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

The ADDRESS clause of the OBJECT-COMPUTER paragraph is not significant to the compiler and is treated as commentary. The MEMORY SIZE clause can be used to communicate to Sort the amount of core available for its use; WORDS specifies core size in terms of words, CHARACTERS specifies core size in terms of bytes, and MODULES specifies core size in terms of 2048-byte pages.

The bulk of 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. In the case above, however, the paragraph is examined by the compiler so that the single item of pertinent information, the SEGMENT-LIMIT clause, can be located. Therefore, the programmer must observe the language format shown.

The computer-name may be any nonreserved word that conforms to the rules for data-name formation: for example, XEROX-SIGMA-9. Each literal may be either numeric or nonnumeric, and any implementor-names written here may be represented by any nonreserved words.

The priority-number introduced by the SEGMENT-LIMIT clause must be an unsigned numeric literal with an integral value greater than 0 and less than 50. PROCEDURE DIVISION section bearing priority-numbers in the range 50 through 99 are considered independent overlayable program segments at execution time. The SEGMENT-LIMIT clause enables the programmer to extend this range, thus making a greater proportion of the object program overlayable if he so desires. A detailed description of priority segmentation and the significance of the SEGMENT-LIMIT clause are presented in Chapter 11, "Priority Segmentation".

SPECIAL-NAMES Paragraph

The formats of this paragraph are

Format 1

```
    SPECIAL-NAMES.  copy-statement.
```

Format 2

```
    SPECIAL-NAMES.
   ┌                  ┌IS mnemonic-name[ON STATUS IS condition-name-1[OFF STATUS IS condition-name-2]]┐┐
   │                  │IS mnemonic-name[OFF STATUS IS condition-name-2[ON STATUS IS condition-name-1]]││
   │implementor-name ┤ON STATUS IS condition-name-1[OFF STATUS IS condition-name-2]                  ├│ . . .
   └                  └OFF STATUS IS condition-name-2[ON STATUS IS condition-name-1]                  ┘┘

          [CURRENCY SIGN IS literal][DECIMAL-POINT IS COMMA].
```

Note: In repetition, a comma may precede implementor-name.

The SPECIAL-NAMES paragraph relates implementor-names or codes to programmer-specified mnemonic-names and defines status conditions.

Format 1 is used when there is an element in the current COBOL source library that contains the special-names declarations desired for this compilation. For additional information see Chapter 9, "COBOL Library".

Implementor-name Clause. Implementor-names recognized by the Xerox ANS COBOL Compiler are

1. "character"
2. CONSOLE
3. PRINTER
4. SWITCH-1 through SWITCH-6

"Character" may be any single-character, nonnumeric literal (i.e., any EBCDIC character except the quotation mark) enclosed by quotation marks. The associated mnemonic-name may be mentioned in the Report Writer CODE clause or in the ADVANCING option of a WRITE statement. The six switches are provided so that sense switches can be simulated. The status of a switch is specified by condition-names and interrogated by testing the condition-names. Mnemonic names associated with the implementor-names CONSOLE and PRINTER may be used in ACCEPT and DISPLAY statements. ON and OFF status applies only to switches.

CURRENCY SIGN Clause. The literal that appears in the CURRENCY SIGN clause is used to represent the currency symbol in the PICTURE clause. The literal is limited to a single-character, nonnumeric literal and may not be one of the characters that is assigned significance in the definition of the PICTURE clause character string. Thus the characters may not be

1. Digits 0 through 9.
2. Alphabetic characters A, B, C, D, P, R, S, V, X, or Z.
3. Special characters asterisk, plus, hyphen (minus), comma, period, left parenthesis, right parenthesis, quotation mark, space (blank), or semicolon.

DECIMAL-POINT Clause. The DECIMAL-POINT IS COMMA clause states that in the current compilation the functions of the comma and period are interchanged in the PICTURE clause character-string and in numeric literals.

The purpose of both the CURRENCY SIGN and DECIMAL-POINT clauses is to render COBOL more acceptable in countries where other conventions are observed.

INPUT-OUTPUT SECTION

FILE-CONTROL Paragraph

The formats of this paragraph are

Format 1

FILE-CONTROL. copy statement.

Format 2

FILE-CONTROL.

{SELECT [OPTIONAL] file-name-1 [ASSIGN-clause] [MULTIPLE-clause]

[RESERVE-clause] [FILE-LIMITS-clause] [ACCESS-clause] [PROCESSING-clause] [ACTUAL 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 Chapter 9, "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 selected file must have a File Description entry in the DATA DIVISION. File-name-2 must not be a Sort-File. The word OPTIONAL indicates an input file that is not necessarily present each time the object program is executed.

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 clause is

[ASSIGN TO [integer]implementor-name-1[, implementor-name-2]...[OR implementor-name-n]]

The ASSIGN clause permits a file to be associated with a particular type of hardware device. For program execution, the actual device assignment is made via the Monitor ASSIGN control command.

Acceptable implementor-names are

1. CARD-PUNCH
2. CARD-READER
3. DISC
4. DRUM
5. MAGNETIC-TAPE
6. PAPER-TAPE-PUNCH
7. PAPER-TAPE-READER
8. PRINTER
9. TYPEWRITER
10. DISC-R

Where multiple implementor-names are specified, only the last name is recorded by the compiler.

"Integer", if specified, refers to removable volumes, e.g., magnetic tapes. The compiler provides space for "integer" number of Volume Serial numbers (SNs) in the file Data Control Block (DCB). If not specified, the default value is three. For Sort-Files, this indicates the number of Sort work files to be used (must be in the range 3 to 17 for tape sorts, 6 to 17 for disk sorts).

MULTIPLE Clause. The format of this clause is

$$\left[\text{FOR MULTIPLE}\begin{Bmatrix}\text{REEL}\\\text{UNIT}\end{Bmatrix}\right]$$

The MULTIPLE REEL clause enables the programmer to instruct the compiler when the number of tape units assigned might be less than the number of reels in the file. Similarly, the MULTIPLE UNIT clause indicates when the number of mass storage devices assigned might be less than the number of mass storage units in the file.

RESERVE Clause. The format of this clause is

$$\left[, \ \underline{\text{RESERVE}} \begin{Bmatrix}\text{integer}\\ \underline{\text{NO}}\end{Bmatrix} \ \text{ALTERNATE} \begin{bmatrix}\text{AREA}\\ \text{AREAS}\end{bmatrix}\right]$$

The RESERVE clause permits the programmer to modify the number of input/output buffer areas to be allocated by the compiler. The integer option specifies that a number (integer) of areas in addition to the standard number are to be reserved for the file.

If "integer" is specified, one alternate area is reserved to provide double buffering. If single buffering is desired, the NO option must be specified so that this alternate area will not be reserved.

FILE-LIMITS Clause. The format of this clause is

$$\left[, \begin{Bmatrix}\underline{\text{FILE-LIMIT}} \ \underline{\text{IS}}\\ \underline{\text{FILE-LIMITS}} \ \underline{\text{ARE}}\end{Bmatrix} \begin{Bmatrix}\text{data-name-1}\\ \text{literal-1}\end{Bmatrix} \ \underline{\text{THRU}} \begin{Bmatrix}\text{data-name-2}\\ \text{literal-2}\end{Bmatrix} \left[, \begin{Bmatrix}\text{data-name-3}\\ \text{literal-3}\end{Bmatrix} \ \underline{\text{THRU}} \begin{Bmatrix}\text{data-name-4}\\ \text{literal-4}\end{Bmatrix}\right] \ldots\right]$$

Each pair of operands associated with the word THRU represents a logical segment of the file. The logical end of a mass storage file is that address represented by the last operand of the clause.

Allocation of hardware resources is solely a monitor system responsibility.

ACCESS Clause. The format of this clause is

$$\left[, \ \underline{\text{ACCESS}} \ \text{MODE} \ \underline{\text{IS}} \begin{Bmatrix}\underline{\text{SEQUENTIAL}}\\ \underline{\text{RANDOM}}\end{Bmatrix}\right]$$

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 is specified the ACTUAL KEY entry (see below) must also be specified, and at execution time the file must be assigned to a direct-access device. In this case, the specified logical record (located using ACTUAL 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 ACTUAL KEY data-name contents) on a WRITE statement execution.

Sequential access is assumed when this clause is omitted.

PROCESSING Clause. The format of this clause is

$$\left[, \ \underline{\text{PROCESSING}} \ \text{MODE} \ \underline{\text{IS}} \ \underline{\text{SEQUENTIAL}} \ \right]$$

The PROCESSING clause permits the programmer to signify whether synchronous or asynchronous processing of a mass storage file is desired. The fixed, read/write head-per-track design of the XDS Rapid-Access Data (RAD) units eliminates the time delays associated with movable-head disc files. Thus, asynchronous processing is not implemented in this COBOL compiler; only sequential processing exists.

ACTUAL KEY Clause. The format of this clause is

$$\left[, \ \underline{\text{ACTUAL}} \ \text{KEY} \ \underline{\text{IS}} \ \text{data-name}\right]$$

The ACTUAL KEY clause must be specified if RANDOM access is specified; it is not meaningful to SEQUENTIAL access. For files assigned to DISC-R, a key defined as COMPUTATIONAL (binary) must be specified.

The contents of data-name are used by the SEEK, 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. This symbolic identify is transmitted to the Monitor system and is used to determine the physical location from which the record is to be read or into which it is to be written.

The Monitor system employs a key consisting of a character-string preceded by a byte containing the length of the string. Data-name may be any data item: it need not exist in the COBOL program in the form required by the Monitor, since the compiler generates code that places the contents of data-name into the desired form before presenting it to the Monitor system for execution of the input/output operation. However, overall length of the data item is limited to 31 characters. Data-name may not be dynamically subscripted.

I-O-CONTROL Paragraph

The formats of this paragraph are

Format 1

$$\underline{\text{I-O-CONTROL.}}\ \text{copy-statement.}$$

Format 2

$$\underline{\text{I-O-CONTROL.}}\ [\text{RERUN-clause}]\ldots[\text{SAME AREA-clause}]\ldots[\text{MULTIPLE FILE-clause}]\ldots .$$

Format 1 causes the library element to be retrieved from the current COBOL source library and inserted into the source prgoram at this point. A discussion of Format 2 follows. For additional information see Chapter 9, "COBOL Library".

RERUN Clause

The format of this clause is

$$\left[\underline{\text{RERUN}} \left[\underline{\text{ON}} \text{ data-name} \right] \text{ EVERY } \left\{ \begin{array}{l} \left\{ \begin{array}{l} \underline{\text{END}} \text{ OF } \left\{ \begin{array}{l} \underline{\text{REEL}} \\ \underline{\text{UNIT}} \end{array} \right\} \\ \text{integer-1 } \underline{\text{RECORDS}} \end{array} \right\} \text{ OF file-name} \\ \text{integer-2 } \underline{\text{CLOCK-UNITS}} \\ \text{condition-name} \end{array} \right\} \right] \ldots$$

The RERUN clause of the I-O-CONTROL paragraph is not significant to the compiler and is treated as commentary.

SAME AREA Clause

The format of this clause is

$$\left[;\ \underline{\text{SAME}} \left[\begin{array}{l} \underline{\text{RECORD}} \\ \underline{\text{SORT}} \end{array} \right] \text{ AREA FOR file-name-1 } \{, \text{ file-name-2}\} \ldots \right] \ldots$$

When SAME RECORD AREA is written, the logical record areas for all of the files mentioned overlap. Thus, although several of the files may be open at the same time, the logical record of only one of the files can exist in the record area at one time. More than one SAME RECORD AREA clause may appear in a COBOL program, but no one file-name may appear in more than one such clause. Sort-Files may not be referenced in a SAME clause. Without the RECORD option, this clause is treated as comment.

MULTIPLE FILE Clause

The format of this clause is

```
    [; MULTIPLE FILE TAPE CONTAINS file-name-1 [POSITION integer-1]
            [, file-name-2 [POSITION integer-2], ...] ...
```

The MULTIPLE FILE clause enables the programmer to indicate the order and positioning of each file on a reel when more than one file shares the same physical magnetic tape reel.

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 falls 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, WORKING-STORAGE, LINKAGE, COMMON-STORAGE, and REPORT 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. The LINKAGE SECTION and the COMMON-STORAGE SECTION contain records and noncontiguous data items that may be used in common by two or more independently compiled programs operating together as a single executable object-time program. Like the WORKING-STORAGE SECTION, the LINKAGE and COMMON-STORAGE SECTIONs may specify both logical records and noncontiguous items. The REPORT SECTION describes the content and format of reports to be generated.

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 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

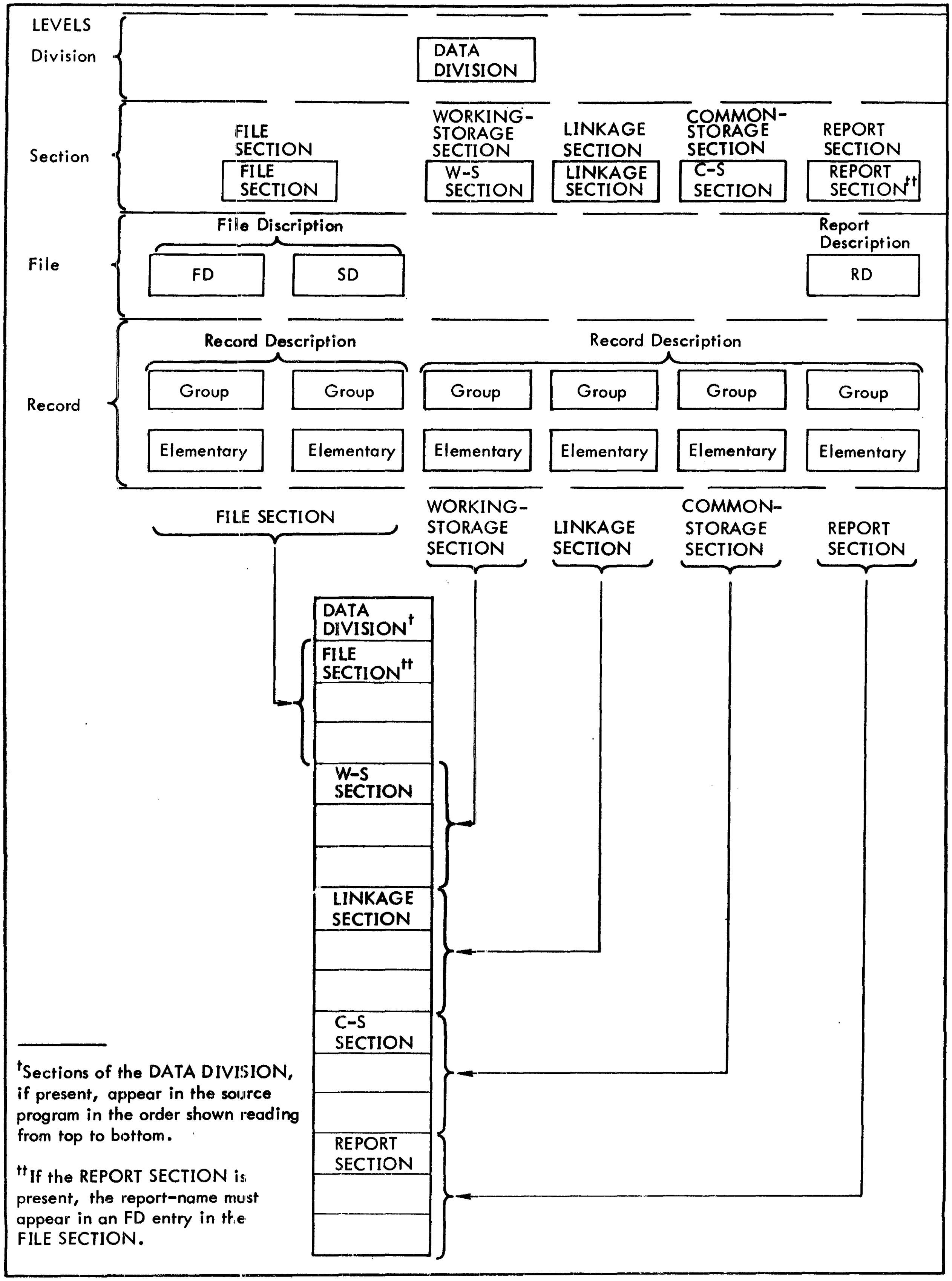
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
COMMON-STORAGE SECTION.
REPORT SECTION.

Section headers for the FILE SECTION and the REPORT SECTION are followed by one or more sets of entries composed of file clauses, followed by associated Record Description entries. WORKING-STORAGE, LINKAGE, and COMMON-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) and Sort-File Description (SD) entries represent 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 or SD), a data-name, and a series of independent clauses. These clauses specify the size of the physical records, the name of the user label record contained in the file, and the names of the data records and reports that compose the file. The entry itself is terminated by a period.

The Sort-File Description (SD) is a special type of file description that gives the names of data records in the Sort-File. See Chapter 8, "Sort Feature" and Appendix E, "Sort Feature Sample Program".



[t] Sections of the DATA DIVISION, if present, appear in the source program in the order shown reading from top to bottom.

[tt] If the REPORT SECTION is present, the report-name must appear in an FD entry in the FILE SECTION.

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). Each WORKING-STORAGE record-name (01 level) and noncontiguous data-name must be unique, since it cannot be qualified; subordinate data-names need not be unique if they can be made unique by qualification.

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 and/or the USAGE 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.

LINKAGE SECTION

The LINKAGE SECTION defines records and noncontiguous data items used in common by independent compilations when they are combined to form a single run-time program.

The LINKAGE SECTION is organized exactly the same way as the WORKING-STORAGE SECTION, beginning with a section header LINKAGE SECTION and a period, followed by Data Description entries for noncontiguous constants and Data Description entries for contiguous constants, in that order. Each LINKAGE SECTION record-name must be unique, since it cannot be qualified; subordinate data-names need not be unique if they can be made unique by qualification. Initial values cannot be specified for items appearing in a LINKAGE SECTION.

COMMON-STORAGE SECTION

The COMMON-STORAGE SECTION provides an alternate means of defining records and noncontiguous data items used in common by independent compilations when they are combined to form a single run-time program.

The COMMON-STORAGE SECTION is organized exactly the same way as the WORKING-STORAGE and LINKAGE SECTIONs, beginning with a section header COMMON-STORAGE SECTION and a period, followed by Data Description entries for noncontiguous constants and Data Description entries for contiguous constants, in that order. Each COMMON-STORAGE SECTION record-name must be unique, since it cannot be qualified; subordinate data-names need not be unique if they can be made unique by qualification. Initial values may be specified for items appearing in a COMMON-STORAGE SECTION.

REPORT SECTION

The REPORT SECTION consists of two types of entries for each report:

1. Report Description (RD) entry.
2. Report Group Description entries.

The RD entry describes physical aspects of the report format. Report Group Description entries describe conceptual characteristics of the items comprising the report and their relation to the report format.

Report Description Entry. The Report Description entry contains information pertaining to the overall format of a report named in the FILE SECTION and is uniquely identified in the REPORT SECTION by the level indicator RD. Characteristics of the report page are provided by describing the number of physical lines per page and limits for specified headings, footings, and details within page structure. Data items that act as contro' factors during presentation of the report are specified in the RD entry. Each report named in an FD entry in the FILE SECTION must be defined by an RD entry.

Report Group Description Entry. A report may be divided into report groups. A report group is a set of data items that are to be presented as an individual unit, irrespective of physical format structure. This unit may consist of several report lines containing many data items or of one report line containing a single data item. Three categories of report group definitions are provided: Heading Groups, Footing Groups, and Detail Groups.

The data items comprising a report group must be identified by the level-number 01 and a TYPE clause. Report group names are required when reference is made in the PROCEDURE DIVISION to

1. A TYPE DETAIL report group by a GENERATE statement.
2. A TYPE HEADING or FOOTING report group by a USE statement.

Description of the report group, analogous to that of the data record, consists of a set of entries defining characteristics of the elements: placement of an item in relation to the entire report group and to the overall report format, format description of all items, and any control factors associated with the group.

File Description - Complete Entry Skeleton

The general formats of this entry are

Format 1

```
FD file-name copy-statement.
```

Format 2

```
FD file-name
[; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]
                                           {CHARACTERS}

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

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

[; VALUE OF data-name-3 IS {data-name-4} [, data-name-5 IS {data-name-6}]...]
                           {literal-1  }                   {literal-2  }

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

[; {REPORT IS  } report-name-1[, report-name-2]...] .
   {REPORTS ARE}
```

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 9, "COBOL Library". A description of Format 2 follows.

BLOCK CONTAINS Clause. The format of this clause is

```
[; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]
                                           {CHARACTERS}
```

The BLOCK CONTAINS clause specifies to the compiler the block size of the file. This information is conveyed either directly in CHARACTERS or indirectly by providing the blocking factor (i.e., the number of logical records per block) via the RECORDS option. If neither CHARACTERS nor RECORDS is written, CHARACTERS is assumed. "Integer-1" and "integer-2" refer to the minimum and maximum size of the block, respectively.

RECORD CONTAINS Clause. The format of this clause is

$$\left[;\ \underline{\text{RECORD}}\text{ CONTAINS }\left[\text{integer-3 }\underline{\text{TO}}\ \right]\text{integer-4 CHARACTERS}\right]$$

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.

Variable length records may be described by the OCCURS DEPENDING ON clause. The Record Description may contain up to 15 variable groups. These variable groups must follow the fixed portion of the record. No fixed items or groups are permitted following the variable groups. The length of each variable group is determined at execution time. For further information, see "OCCURS Clause", later in this chapter.

LABEL RECORDS Clause. The format of this clause is

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

This clause is required in every File Description 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.

The data-name option specifies that user labels exist. Data-name-1, data-name-2, etc., are the names of user label records that (1) must not appear in the DATA RECORDS clause, and (2) must be the subject of a Record Description associated with the file. The contents of user labels can be accessed via USE statements in the DECLARATIVES SECTION. All Procedure Division references to data-name-1, or to any items subordinate to data-name-1, must appear within USE procedures.

VALUE OF Clause. The format of this clause is

$$\left[;\ \underline{\text{VALUE}}\ \underline{\text{OF}}\text{ data-name-3 IS }\left\{\begin{matrix}\text{data-name-4}\\ \text{literal-1}\end{matrix}\right\}\left[,\text{ data-name-5 IS }\left\{\begin{matrix}\text{data-name-6}\\ \text{literal-2}\end{matrix}\right\}\right]...\right]$$

Sigma Monitor systems perform label-checking services, hence this clause is treated as documentation.

DATA RECORDS Clause. The format of this clause is

$$\left[;\ \underline{\text{DATA}}\left\{\begin{matrix}\underline{\text{RECORD}}\text{ IS}\\ \underline{\text{RECORDS}}\text{ ARE}\end{matrix}\right\}\text{data-name-7}\left[,\text{ data-name-8}\right]...\right]$$

The DATA RECORDS clause cross-references the description of data records with their associated file description. Each logical record in the file must be named in this clause; the order of listing the names is not significant.

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.

REPORT Clause. The format of this clause is

$$\left[;\left\{\begin{matrix}\underline{\text{REPORT}}\ \text{IS}\\ \underline{\text{REPORTS}}\ \text{ARE}\end{matrix}\right\}\ \text{report-name-1}[,\ \text{report-name-2}]\ldots\right]$$

The REPORT clause cross-references the description of Report Description entries with their associated File Description entry. Each report-name listed in the FD entry must be the subject of a Report Description (RD) entry in the REPORT SECTION.

The REPORT clause is required in the File Description entry if the file is an output report file or is to contain output report records. The presence of more than one report-name indicates that the file contains more than one report; the order of presentation is not significant.

Data Description Entries

General Format:

$$\begin{array}{ll}\text{level-number}\left\{\begin{matrix}\text{data-name}\\ \underline{\text{FILLER}}\end{matrix}\right\}[\text{REDEFINES-clause}]\ [\text{COPY statement}]\\ \qquad [\text{PICTURE-clause}]\ [\text{USAGE-clause}]\ [\text{SYNCHRONIZED-clause}]\\ \qquad [\text{BLANK-clause}]\ [\text{JUSTIFIED-clause}]\ [\text{VALUE-clause}]\\ \qquad [\text{OCCURS-clause}]\ [\text{RENAMES-clause}].\end{array}$$

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 a 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

$$\begin{array}{ll}\text{level-number}\left\{\begin{matrix}\text{data-name}\\ \underline{\text{FILLER}}\end{matrix}\right\}[\text{REDEFINES-clause}]\ [\text{OCCURS-clause}]\\ \qquad [\text{USAGE-clause}]\\ \qquad [\underline{\text{VALUE}}\ \text{is non-numeric-literal}].\end{array}$$

Example:

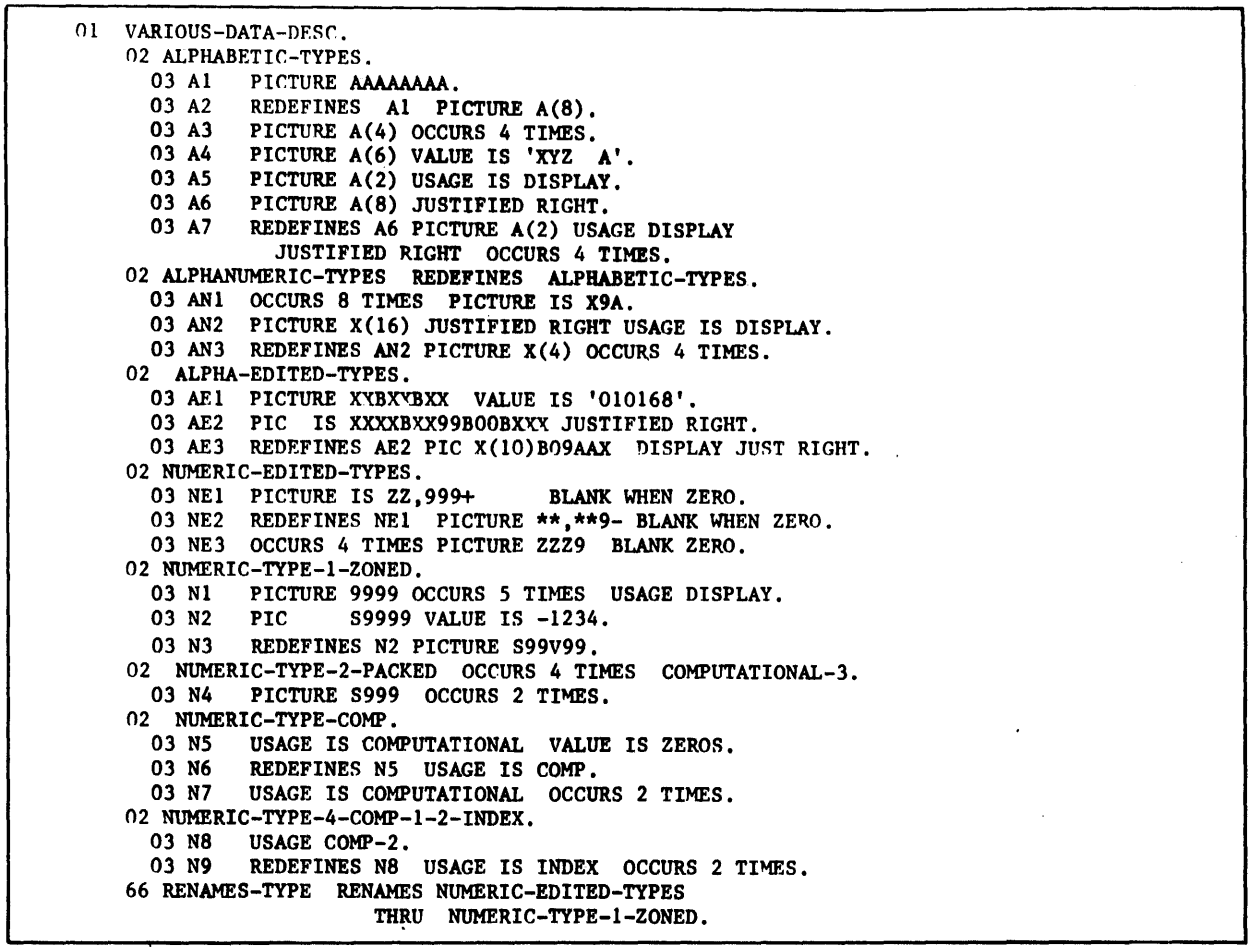
```
01  GROUP-ITEM.
    02  FIELD-1 PICTURE X.
    02  FIELD-2 PICTURE X.
```

Alphabetic Elementary Item

$$\begin{array}{ll}\text{level-number}\left\{\begin{matrix}\text{data-name}\\ \underline{\text{FILLER}}\end{matrix}\right\}[\text{REDEFINES-clause}]\ [\text{OCCURS-clause}]\\ \qquad \left\{\begin{matrix}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{matrix}\right\}\ \text{IS alpha-type}\ \ [\text{USAGE IS}\ \underline{\text{DISPLAY}}]\\ \qquad [\underline{\text{VALUE}}\ \text{IS non-numeric-literal}]\left[\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\}\underline{\text{RIGHT}}\right].\end{array}$$

Example:

```
05      CLIENT-NAME PICTURE A(35) USAGE IS DISPLAY.
```



```
01  VARIOUS-DATA-DESC.
    02 ALPHABETIC-TYPES.
      03 A1    PICTURE AAAAAAAA.
      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) JUSTIFIED RIGHT.
      03 A7    REDEFINES A6 PICTURE A(2) USAGE DISPLAY
                 JUSTIFIED RIGHT  OCCURS 4 TIMES.
    02 ALPHANUMERIC-TYPES  REDEFINES  ALPHABETIC-TYPES.
      03 AN1   OCCURS 8 TIMES  PICTURE IS X9A.
      03 AN2   PICTURE X(16) JUSTIFIED RIGHT USAGE IS DISPLAY.
      03 AN3   REDEFINES AN2 PICTURE X(4) OCCURS 4 TIMES.
    02  ALPHA-EDITED-TYPES.
      03 AE1   PICTURE XXBXXBXX  VALUE IS '010168'.
      03 AE2   PIC  IS XXXXBXX99B00BXXX JUSTIFIED RIGHT.
      03 AE3   REDEFINES AE2 PIC X(10)B09AAX  DISPLAY JUST RIGHT.
    02 NUMERIC-EDITED-TYPES.
      03 NE1   PICTURE IS ZZ,999+      BLANK WHEN ZERO.
      03 NE2   REDEFINES NE1  PICTURE **,**9- BLANK WHEN ZERO.
      03 NE3   OCCURS 4 TIMES PICTURE ZZZ9  BLANK ZERO.
    02 NUMERIC-TYPE-1-ZONED.
      03 N1    PICTURE 9999 OCCURS 5 TIMES  USAGE DISPLAY.
      03 N2    PIC     S9999 VALUE IS -1234.
      03 N3    REDEFINES N2 PICTURE S99V99.
    02  NUMERIC-TYPE-2-PACKED  OCCURS 4 TIMES  COMPUTATIONAL-3.
      03 N4    PICTURE S999  OCCURS 2 TIMES.
    02  NUMERIC-TYPE-COMP.
      03 N5    USAGE IS COMPUTATIONAL  VALUE IS ZEROS.
      03 N6    REDEFINES N5  USAGE IS COMP.
      03 N7    USAGE IS COMPUTATIONAL  OCCURS 2 TIMES.
    02 NUMERIC-TYPE-4-COMP-1-2-INDEX.
      03 N8    USAGE COMP-2.
      03 N9    REDEFINES N8  USAGE IS INDEX  OCCURS 2 TIMES.
    66 RENAMES-TYPE  RENAMES NUMERIC-EDITED-TYPES
                       THRU  NUMERIC-TYPE-1-ZONED.
```

Figure 2. Various Data Description Entries Listing

Alphanumeric Elementary Item

level-number $\left\{\begin{array}{l}\text{data-name}\\ \underline{\text{FILLER}}\end{array}\right\}$ [REDEFINES-clause] [OCCURS-clause]

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ IS an-type [USAGE IS $\underline{\text{DISPLAY}}$]

[$\underline{\text{VALUE}}$ IS non-numeric-literal] $\left[\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\ \underline{\text{RIGHT}}\right]$.

Example:

04 U-NAME PICTURE X(21) DISPLAY.

04 ERROR-DATA PIC X(45) JUST RIGHT.

Alphanumeric Edited Elementary Item

level-number $\left\{\begin{array}{l}\text{data-name}\\ \underline{\text{FILLER}}\end{array}\right\}$ [REDEFINES-clause] [OCCURS-clause]

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ IS ae-type [USAGE IS $\underline{\text{DISPLAY}}$]

[$\underline{\text{VALUE}}$ IS non-numeric-literal] $\left[\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\ \underline{\text{RIGHT}}\right]$.

Example:

```
     05     DATE PICTURE XXBXXXBXXXX VALUE '19 AUG 1968'.
```

Numeric Edited Elementary Item

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

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

             [USAGE IS DISPLAY].
```

Example:

```
     02     W-RECORD-COUNT PIC 999 BLANK ZERO.
     02     W-INVENTORY-VALUE PICTURE $Z, ZZZ, ZZZ, ZZZ. 99-.
```

Zoned 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     UNIT-PRICE PICTURE 9(4)V99 VALUE 1280.25.
```

Packed Decimal Elementary Item

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

             {PICTURE}IS numeric-type  USAGE IS {COMPUTATIONAL-3}
             {PIC    }                          {COMP-3         }

             [VALUE IS numeric-literal].
```

Example:

```
     02     INVENTORY-VALUE PICTURE 9(10)V99 COMPUTATIONAL-3 OCCURS 5 TIMES.
```

Binary Elementary Item

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

             USAGE IS {COMPUTATIONAL}
                      {COMP         }

             [VALUE IS numeric-type].
```

Example:

```
     77      SUBSCRIPT-1 COMP VALUE 8.
```

Floating-Point Elementary Item

$$\text{level-number} \left\{\begin{array}{l}\text{data-name}\\ \underline{\text{FILLER}}\end{array}\right\} [\text{REDEFINES-clause}]\ [\text{OCCURS-clause}]$$

$$\text{USAGE IS} \left\{\begin{array}{l}\underline{\text{COMPUTATIONAL-1}}\\ \underline{\text{COMP-1}}\\ \underline{\text{COMPUTATIONAL-2}}\\ \underline{\text{COMP-2}}\end{array}\right\}$$

$$[\underline{\text{VALUE}}\ \text{IS numeric-type}].$$

Example:

```
     06      FP-FIELD-1 COMPUTATIONAL-1 VALUE -36.2584.
```

Index Elementary Item

$$\text{level-number} \left\{\begin{array}{l}\text{data-name}\\ \underline{\text{FILLER}}\end{array}\right\} [\text{REDEFINES-clause}]\ [\text{OCCURS-clause}]$$

$$\text{USAGE IS } \underline{\text{INDEX}}$$

$$[\underline{\text{VALUE}}\ \text{IS numeric-type}].$$

Example:

```
     77      IND USAGE IS INDEX VALUE IS 16.
```

Condition Name Item

$$\text{88 condition-name;} \left\{\begin{array}{l}\underline{\text{VALUE IS}}\\ \underline{\text{VALUES ARE}}\end{array}\right\} \text{literal-1} \left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\} \text{literal-2}\right]$$

$$\left[\text{, literal-3} \left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\} \text{literal-4}\right]\right] \ldots \quad .$$

Example:

```
     01      GROUP-ITEM
             02      FIELD-1 PICTURE XX.
                     88      COND VALUE IS 'ON'.
```

REDEFINES Clause. The format of this clause is

$$\text{level-number data-name-1 } \underline{\text{REDEFINES}} \text{ data-name-2}$$

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 but must not be 66 or 88.

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 9, "COBOL Library".

PICTURE Clause. The format of this clause is

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ 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, P(10)9(2) is equivalent to PPPPPPPPPP99.

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, PPPPPPPPPP99, or by a PICTURE containing only 9 characters, P(10)9(2). In either case, the actual size of the item is only 2 characters. An item containing 75 alphabetic characters may be specified by the PICTURE A(75), which only uses 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 65,535 characters; other classes of data items are limited to 255 characters except for numeric display items, which are limited to 31 digits. However, numeric operands in arithmetic operations are limited to a maximum of 18 digits only. The size of an entire Group Item is also limited to 65,535 characters.

2. Categories of Data

a. Alphabetic (alpha-type)

The PICTURE of an alphabetic item contains only the character A. The number of As 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 EBCDIC 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, P, S, and V.

The character 9 represents a digit position containing a numeral and is counted in the size of the item.

The character P indicates an assumed decimal scaling position and specifies the location of an assumed decimal point when the point is not within the number that appears in the data item. The character P is not counted in the size of the data item and can appear only to the left or right as a continuous string of Ps. Since the scaling position character P implies an assumed decimal point (to the left of the Ps if Ps are leftmost PICTURE characters, and to the right of the Ps if Ps are rightmost PICTURE characters), the character V is redundant as either the leftmost or rightmost character with such a PICTURE (see below).

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 he can specify arithmetic statements 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 P V $ + - . , 0 B CR DB Z *

The characters 9, P, 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. A special currency sign may replace the $ sign, but must be defined in the SPECIAL-NAMES paragraph.

+ 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.

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
48	$99	$48
48↑34	$99.99	$48.34
4834	9,999	4,834
292	+999	+292
29$\overset{+}{2}$	+999	+292
29$\bar{2}$	+999	-292
29$\bar{2}$	999-	292-
29$\overset{+}{2}$	-999	△292
292	999-	292△
243↑21	$BB999.99	$△△243.21
243↑21	$00999.99	$00243.21
11↑3$\bar{4}$	99.99CR	11.34CR
11↑34	99.99CR	11.34△△
23↑7$\bar{6}$	99.99DB	23.76DB
23↑76	99.99DB	23.76△△

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. Zs 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 nonzero 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 Zs, 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 Zs, the $ + or - is replaced by a blank.

If a 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. A special currency sign may replace the $ but must be defined in the SPECIAL-NAMES paragraph.

+ 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.ZZ	△ △ △ △ △ △ △
0 0 9↑2 3	$***.99	$ * * 9 . 2 3
0 0 0 8↑2 4	$$$$9.99	△ △ $ 8 . 2 4
0 0 5↑2 $\bar{6}$	---9.99	△ △ - 5 . 2 6
3 2↑6 5	$$$.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↑	$***,***.99	$ 1 2 3 , 4 5 6 . 0 0
1↑2 3 4 5 6	$***,***.99	$ * * * * * * 1 . 2 3
0 0 0 0 1 $\overset{+}{2}$↑	+999,999	+ 0 0 0 , 0 1 2
0 0 0 0 1 $\bar{2}$↑	-ZZZ,ZZZ	- △ △ △ △ △ 1 2

Examples of PICTURE Editing (cont.):

Data to Be Edited	PICTURE of Report Item	Edited Item
$1\ 2\ 3\ 4\ 5\ \bar{6}_{\uparrow}$	$ZZZ,ZZ9.99CR	$ 1 2 3 , 4 5 6 . 0 0 C R
$0\ 0\ 0\ 1_{\uparrow}2\ \overset{+}{3}$	$ZZZ,ZZ9.99DB	$ △ △ △ △ △ △ 1 . 2 3 △ △
$0\ 0\ 1\ 2\ 3_{\uparrow}4$	$(4),$$9.99	△ △ △ △ $ 1 2 3 . 4 0
$0\ 0\ 0\ 0_{\uparrow}0\ 0$	$(4),$$$.99	△ △ △ △ △ △ △ $. 0 0
$0\ 0\ 0\ 0_{\uparrow}1\ \bar{2}$	----,---.99	△ △ △ △ △ △ △ - . 1 2
$0\ 0\ 0\ 0_{\uparrow}1\ \overset{+}{2}$	----,---.99	△ △ △ △ △ △ △ △ . 1 2
$0\ 0\ 0\ 0_{\uparrow}0\ 1$	$$$$,$ZZ.99	Illegal PICTURE

5. Summary

   a. 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.

   b. 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.

   c. A plus sign and a minus sign may not be included in the same PICTURE.

   d. A leftmost plus sign and a dollar sign may not be included in the same PICTURE.

   e. A leftmost minus sign and a dollar sign may not be included in the same PICTURE.

   f. The character 9 may not be specified to the left of a replacement character.

   g. Symbols that may appear only once are V S . CR and DB.

   h. The decimal point may not be the rightmost character in a PICTURE.

Table 2 shows the possible legal combinations of characters in a PICTURE clause. Diagnostic tests are made for most of the conditions represented in the table. Refer to the COBOL Operations manual for a listing of diagnostics.

USAGE Clause. The format of this clause is

$$[\underline{\text{USAGE}}\ \text{IS}]\left\{\begin{array}{l}\underline{\text{DISPLAY}}\\ \underline{\text{INDEX}}\\ \underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\\ \underline{\text{COMPUTATIONAL-1}}\\ \underline{\text{COMP-1}}\\ \underline{\text{COMPUTATIONAL-2}}\\ \underline{\text{COMP-2}}\\ \underline{\text{COMPUTATIONAL-3}}\\ \underline{\text{COMP-3}}\end{array}\right\}$$

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.

Table 2. Combination of Characters in a PICTURE Clause

Given Character in PICTURE	Character Legal Anywhere to Right																				
	A	X	9	S	V	P	Z	*	$	,	.	B	0	-	+	C	D	R	Repeated $	Repeated -	Repeated +
A	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No
X	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No
9	Yes	Yes	Yes	No	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
S	No	No	Yes	No	Yes	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No
V	No	No	Yes	No	No	Yes	2	2	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	2	2	2
P	No	No	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Z	No	No	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
*	No	No	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
$	No	No	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
,	No	No	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
.	No	No	Yes	No	No	No	2	2	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	2	2	2
B	Yes	Yes	Yes	No	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
0	Yes	Yes	Yes	No	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Leading -	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	Yes	Yes	No
Leading +	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	Yes	No	Yes
C	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No
D	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
R	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Trailing -	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Trailing +	No	No	No	No	1	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No

[1] Permissible if all digits to the right are the same.

[2] Permissible if all digits both to the right and left are the same.

DISPLAY denotes that the item is carried in the EBCDIC 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 rightmost byte can contain an operational sign in addition to a digit, i.e., zoned decimal format.

An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that corresponds to an occurrence number of a table element. Like a COMPUTATIONAL item, an index data item is carried in binary in a computer word. Index data items are discussed fully under "Table-Handling Statements" in Chapter 6.

COMPUTATIONAL defines a one-word (32-bit) binary item. The item may contain integral values only, ranging from -2^{31} to $+2^{31}-1$.

COMPUTATIONAL-1 defines a single precision, floating-point data item occupying one 32-bit word.

COMPUTATIONAL-2 defines a double precision, floating-point data item that resides in a doubleword.

COMPUTATIONAL-3 identifies a packed decimal data item whose length is specified by the accompanying PICTURE clause.

<u>SYNCHRONIZED Clause.</u> The format of this clause is

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

Since the Xerox ANS COBOL compiler automatically aligns those data items (COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, and INDEX) requiring alignment at appropriate machine storage boundaries, the ability to specify such alignment (synchronization) is not extended into the language.

<u>BLANK WHEN ZERO Clause.</u> The format of this clause is

$$\underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}$$

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. If a combination of check protection and BLANK WHEN ZERO is specified, the latter is overridden.

<u>JUSTIFIED Clause.</u> The format of this clause is

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

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 is aligned at the rightmost character position in the data item with other characters space-filled.

<u>VALUE Clause.</u> The formats of this clause are

<u>Format 1</u>

$$\underline{\text{VALUE}}\ \text{IS literal}$$

<u>Format 2</u>

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

<u>Note:</u> A figurative-constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

The VALUE clause defines the value of constants, the initial value of working-storage items, or the values associated with a condition-name. 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, all literals in the VALUE clause must be numeric literals. The literal is aligned according to the alignment rules except that the literal must not have a value requiring truncation of nonzero digits.

   b. If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned according to the alignment rules (see "JUSTIFIED Clause" above) except that the number of characters in the literal must not exceed the size of the item.

   c. All numeric literals in a VALUE clause of an item must have a value within the range of values indicated by the USAGE or PICTURE clause; for example, for PICTURE PPP99 the literal must be within the range 0.00000 through 0.00099.

   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. Condition-Name Rules

   a. The VALUE clause is required in a condition-name entry and is the only clause permitted in that entry. The characteristics of a condition-name are implicitly those of its conditional variable.

   b. Format 2 can be used only in connection with condition-names. Wherever the THRU entry is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

3. Data Description Entries Other than Condition-Names

   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 meaningful only in condition-name entries, and is considered merely documentation in other Data Description entries.

      (2) In the WORKING-STORAGE and COMMON-STORAGE SECTIONs, the VALUE clause may be used in condition-name entries and also 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.

      (3) Use of the VALUE clause is not permitted in the LINKAGE SECTION.

      (4) In the REPORT SECTION, the VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at the elementary level in the REPORT SECTION.

   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. This rule does not apply to condition-name entries.

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

OCCURS Clause. The formats of this clause are

Format 1

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

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

Format 2

```
OCCURS integer-1 TO integer-2 TIMES [DEPENDING ON data-name-1]

                 [{ASCENDING } KEY IS data-name-2 [, data-name-3]...]...
                 [{DESCENDING}                                       ]

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

The OCCURS clause eliminates the need for separate entries of repeated data and supplies information required for the application of subscripts or indexes. This clause cannot be specified in a Data Description entry that has a 01 or a 77 level-number.

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 or indexed whenever it is referenced in a statement other than SEARCH. Furthermore, if the subject of this entry is the name of a group item, all data-names belonging to the group must be subscripted or indexed whenever they are used as operands.

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

The OCCURS clause may not be specified in a data description entry that

1. Has an 01 or a 77 level-number.

2. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains an OCCURS clause with the DEPENDING ON option. Therefore, nesting of variable items is not permitted.

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. Condition-name entries, however, are permitted.

In Format 1, integer-2 represents the exact number of occurrences of the data item. Format 2 specifies that the subject of this entry has a variable number of occurrences. Integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. The value of integer-1 must be less than integer-2. The value of integer-1 may be zero, but integer-2 may not be zero.

For the DEPENDING ON option, the data description of data-name-1 must describe a positive integer. Data-name-1 may be qualified, when necessary, but it must not be subscripted. The value of data-name-1 is the count of the actual number of occurrences of the subject at any given time during execution; therefore, its value must not exceed integer-2. Reducing the value of data-name-1 makes inaccessible the contents of data items whose occurrence numbers now exceed the value of data-name-1.

If data-name-1 is an entry in the same record as the current data description entry, the data description entry for data-name-1 must be before a data description entry containing the OCCURS clause in which data-name-1 appears. A record description may contain up to 15 variable groups, and these variable groups must follow the fixed portion of the record. No fixed items or groups are permitted following the variable groups (see Example 3 below).

Unused character positions resulting from use of the DEPENDING ON option will not appear in the external media.

If the DEPENDING ON option is specified for a table referenced by a SEARCH statement, the contents of data-name-1 is interrogated at execution time in order to determine the extent of the table.

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.

The KEY IS option is used to indicate that repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. Data-names are listed in descending order of significance. If data-name-2 is not the subject of this entry.

1. All of the entries identified by the data-names in the KEY IS phrase must be within the group item that is the subject of this entry.

2. None of the data-names in the KEY IS phrase can contain an entry containing an OCCURS clause or be subordinate to an entry containing an OCCURS clause, other than the subject of this entry.

If the DEPENDING ON option of the OCCURS clause is used, certain rules must be observed. If the value of data-name-1 changes during program execution, the following results occur:

1. The size of any group described by or containing the related OCCURS clause will be recomputed to reflect the new value of data-name-1.

2. Compilation of the location of any items following the item described with the OCCURS clause will be affected by the new value of data-name-1. If the user wishes to refer to these following (nonsubordinate) items, he should provide procedural statements to accomplish the proper adjustments:

   a. Before the change in data-name-1, the nonsubordinate items should be moved to a work area.

   b. After the change in data-name-1, these items should be moved back. Therefore, the user is responsible for protecting nonsubordinate items when a variable item is added, and "squeezing" them down when a variable item is deleted.

Example 1:

Assume that the Data Division of a program contains the following record description:

```
01  RECORD-1.
    02  FIXED-PORTION.
        .
        .
        .
    02  A PICTURE S999 COMP-3.
    02  B PICTURE S999 COMP-3.
    02  VAR-GROUP-A PICTURE X(4) OCCURS 1 TO 20 TIMES DEPENDING ON A.
    02  VAR-GROUP-B PICTURE X(5) OCCURS 5 to 10 TIMES DEPENDING ON B.
        .
        .
        .
```

VAR-GROUP-A is a variable item since it contains the DEPENDING ON option. Since VAR-GROUP-B is not subordinate to VAR-GROUP-A, the location of VAR-GROUP-B will be affected by a change in the value of A. Assuming that work-b is a work area with the same data structure as VAR-GROUP-B, the following procedural coding should be used to preserve the contents of VAR-GROUP-B:

1. MOVE VAR-GROUP-B TO work-b.

2. Change the value of A.

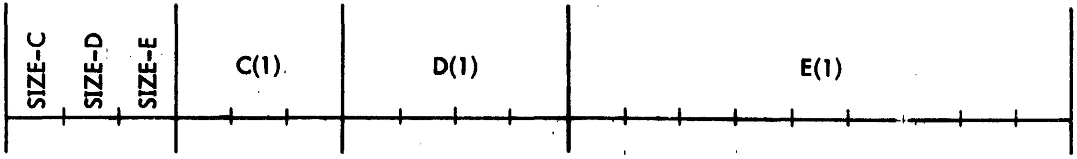
3. MOVE work-b TO VAR-GROUP-B.

Example 2:

Assume the following record description:

```
01  RECORD-2.
    02  SIZE-C  PIC 9.
    02  SIZE-D  PIC 9.
    02  SIZE-E  PIC 9.
    02  C  PIC  X(3)  OCCURS 1 TO 4 DEPENDING ON SIZE-C.
    02  D  PIC  X(4)  OCCURS 1 TO 2 DEPENDING ON SIZE-D.
    02  E  PIC  X(9)  OCCURS 1 TO 3 DEPENDING ON SIZE-E.
```

Assume that the values of SIZE-C, SIZE-D, and SIZE-E are all equal to 1. The record will then be packed in memory as follows:



If the record is output onto external media (e.g., WRITE RECORD-2), a 19-character record will be output since the unused character positions will not appear in the external media.

Example 3:

Assume the following record description:

```
01  RECORD-3.
    02  FIXED-PORTION.
         .
         .
         .
    02  A PICTURE S999 COMP-3.
    02  VAR-GROUP-A OCCURS 10 TIMES.
    03  ITEM-B PICTURE X(4) OCCURS 1 TO 20 TIMES DEPENDING ON A.
    03  ITEM-C PICTURE X(4).
02  GROUP-D.
    03  ITEM-E PICTURE X(4).
```

This description is illegal because it violates Rule 2 which states that the OCCURS clause may not be specified in a data description entry that describes an item whose size is variable. In the example above, ITEM-B is a variable item; thus, VAR-GROUP-A may not contain an OCCURS clause. Also, another violation has occurred in that both ITEM-C and GROUP-D are fixed items or groups. This is illegal because fixed items or groups may not follow a variable group. All variable groups must be at the end of the record.

Example 4:

The data description below is valid since GROUP-A, GROUP-B, and GROUP-C are not variable items.

```
01  RECORD-4.
    02  FIXED-PORTION.
         .
         .
         .
    02  A PICTURE S999 COMP-3.
    02  VAR-GROUP-A OCCURS 1 TO 10 TIMES DEPENDING ON A.
        03  GROUP-A PICTURE X(10) OCCURS  5 TIMES.
        03  GROUP-B  PICTURE X(20) OCCURS 10 TIMES.
        03  GROUP-C  PICTURE X(30) OCCURS 15 TIMES.
```

RENAMES Clause. The format of this clause is

66 data-name-1; RENAMES data-name-2 THRU data-name-3 .

The RENAMES clause permits alternative – possible overlapping – groupings of elementary items. All RENAMES entries associated with a given logical record must immediately follow the last Data Description entry of that record.

Data-name-2 and data-name-3 must be names of elementrary items or groups of elementary items in the associated logical record, and cannot be the same data-name. A 66-level entry cannot rename another 66-level entry nor can it rename a 77-, or 01-level entry.

Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the 01- or FD-level entries. Data-name-2 and data-name-3 may not have an OCCURS clause in a Data Description entry or be subordinate to an item that has an OCCURS clause in its Data Description entry. Data-name-2 must have a smaller displacement from the base of the logical record than data-name-3.

When data-name 3 is specified, data-name-1 is treated as a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

When data-name-3 is not specified, data-name-1 is treated either as a group item or as an elementary item. It is treated as a group item if data-name-2 is a group item, or as an elementary item if data-name-2 is an elementary item.

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, USE, and NOTE.

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. SEARCH with AT END option.
4. Arithmetic statements with the SIZE ERROR option.
5. 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, an optional priority-number, 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.

[DECLARATIVES.                                                  ]
[{section-name SECTION. declarative-sentence.                   ]
[{paragraph-name. {sentence.} ...} ...} ...                     ]
[END DECLARATIVES.                                              ]

{section-name SECTION [priority].

{paragraph-name.{sentence.} ...} ...} ...
```

Format 2

```
PROCEDURE DIVISION.

{paragraph-name.{sentence.} ...} ...
```

If the program contains declaratives, the DECLARATIVES sections must be grouped at the beginning of the PROCEDURE DIVISION preceded by the key word DECLARATIVES and followed by the key phrase END DECLARATIVES.

The sections in the DECLARATIVES portion are executed when exceptional conditions are encountered during input/output operations or in the course of writing a report. A USE statement must identify the conditions under which each section is executed. For further discussion refer to "USE Statement" under "Compiler-Directing Statements" at the end of this chapter.

Execution of the program begins at the first statement of the first nondeclarative 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
**	Exponentiation

Also, the operator '-' may be used as a unary – to indicate logical negation. In this case, the unary operator must not be preceded by a space when it follows a left parenthesis.

The following are examples of arithmetic-expressions:

1. (HOURS + (OVERTIME * 1.5)) * WAGE-RATE

2. PI * RADIUS ** 2 * HEIGHT / 3

3. WEEKLY-SALES * +.05

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. Thus, if WEEKLY-SALES has the value 574.20, the third example would reduce to the value of 28.71. An arithmetic-expression may be used in the COMPUTE statement or in conditional expressions (see below). 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, exponentiation is performed.

3. Then, multiplication and division are performed.

4. Finally, addition and subtraction are performed.

5. In each of the four 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.

6. 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 3.

Table 3. Rules for Constructing Arithmetic-Expressions

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

1 This is permitted when + or - indicates the sign of a numeric literal, in which case no space is allowed between the sign and the number.

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. Exact rules covering treatement of this situation and conversion from one COMPUTATIONAL form to another are discussed separately in Appendix D, "Evaluation of Arithmetic-Expressions".

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, PERFORM, and SEARCH, in which the condition is explicitly stated.

Conditions

Several kinds of condition tests are at the disposal of the programmer, and COBOL allows combination of these tests using logical operators AND, OR, and NOT. Condition tests allowed are

1. Class tests
2. Switch-status tests
3. Relation tests between arithmetic-expressions
4. Condition-name tests (effectively equivalent to relation tests)
5. Sign tests (a special case of relation tests)

Relations

Relational-operators in the COBOL language are

$$\text{IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{GREATER}}\ \text{THAN} \\ \quad\underline{>} \end{array} \right\}$$

$$\text{IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{LESS}}\ \text{THAN} \\ \qquad\underline{<} \end{array} \right\}$$

$$\text{IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{EQUAL}}\ \underline{\text{TO}} \\ \qquad\underline{=} \end{array} \right\}$$

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, literals, or arithmetic-expressions to create relation conditions. The general format is

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

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 MARRIED 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 NOT MARRIED 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 a diagnostic message at compile time. See Appendix G, "Xerox ANS COBOL Compiler Diagnostics".

Order of Computation in Arithmetic-Expressions

The way in which a compound condition is evaluated can be specified by parentheses. Thus, the compound condition

A + 2 = B OR NOT A - C = 3 AND D = 4

might be considered ambiguous because

(A + 2 = B OR NOT A - C = 3) AND D = 4   or   A + 2 = B OR NOT (A - C = 3 AND D = 4)

are both among possible interpretations.

If parentheses are not written to specify the order of computation, COBOL evaluates a compound condition using the following rules:

1. Arithmetic-expressions are evaluated first.
2. Relationals and other condition tests are evaluated next.
3. NOT, AND, and OR are then evaluated, in that order.

Evaluation begins with the innermost pair of parentheses and proceeds to the outermost. If the order of evaluation is not specified by parentheses, the expression is evaluated in the following way: the conditions surrounding all ANDs are evaluated first, starting at the left of the expression and proceeding to the right; then the ORs are evaluated, also working from left to right. Thus the correct interpretation of the compound condition above is neither of those given but is

A + 2 = B OR ((NOT A - C = 3) AND D = 4).

Table 4 indicates the allowable relationships between logical connectors and simple conditions.

Table 4. Rules for Constructing Compound Conditions

	Second Symbol					
First Symbol	(	OR	AND	NOT	Simple Condition	) or End of Compound Condition
(or Beginning of Compound Condition	P	-	-	P	P	-
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P	-	-	-	P	-
Simple Condition	-	P	P	-	-	P
)	-	P	P	-	-	P

P   A permitted relationship.

-     A relationship that is not permitted.

Other Conditions Tests

Condition-Name Test

The format of this test is

$[\underline{\text{NOT}}]$ condition-name

A condition-name test determines whether or not the value of a conditional variable is equal to the value specified for a condition-name associated with it. It is, in effect, a special case of relation testing wherein the relation is equality, and the data item and value to be compared are specified by a level-88 entry in the DATA DIVISION rather than being explicitly stated in the PROCEDURE DIVISION.

A condition-name is a name given to a value or range of values of a data-name. In the DATA DIVISION a condition-name is assigned to a particular value or range of values of a particular data-name. For example, the data item MARITAL-STATUS might be a code indicating whether an employee is married, divorced, or single. Assume that if MARITAL-STATUS has the value 1, the employee is single; if it has the value 2, he is married; and if it equals 3, he is divorced. To determine whether or not an employee is married, the condition might be tested by using a relational test in a condition statement such as IF MARITAL-STATUS = 2 SUBTRACT MARRIED-DEDUCTION FROM GROSS.

However, in the DATA DIVISION a condition-name can be associated with each value that MARITAL-STATUS might assume; thus, the condition-name SINGLE might be associated with the "condition" that the data-name MARITAL-STATUS has a value of 1. MARRIED might be similarly associated with 2, and DIVORCED with 3.

For example,

```
02  MARITAL-STATUS PICTURE 9.
    88  SINGLE     VALUE 1.
    88  MARRIED    VALUE 2.
    88  DIVORCED VALUE 3.
```

Then, as a shorthand form of the simple relational condition MARITAL-STATUS = 2, the programmer could write the single condition-name MARRIED. Therefore, identical results would be produced by

```
IF MARITAL-STATUS = 2 SUBTRACT MARRIED-DEDUCTION FROM GROSS.

IF MARRIED SUBTRACT MARRIED-DEDUCTION FROM GROSS.
```

Thus the condition-name is another form of a relation. It is an alternative way of expressing the same tests. Details of specifying condition-names in the DATA DIVISION are given in the "VALUE Clause" paragraph under "Data Description Entries" in Chapter 5.

Sign Test

The format of this test is

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

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

$$\underline{\text{IF}}\ \text{data-name IS}\,[\underline{\text{NOT}}]\left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{array}\right\}$$

The data-name must be defined in the DATA DIVISION as USAGE DISPLAY.

Table 5 lists cases where the class test is valid and meaning of the results.

Switch-Status Test

The format of this test is

[NOT] switch-status-name

With the switch-status test it is possible to test the two positions (ON and OFF) of a pseudoswitch by associating a switch-status-name with either the ON or the OFF position. Switch-status-names are assigned in the ENVIRONMENT DIVISION using the SPECIAL-NAMES paragraph. Refer to "CONFIGURATION SECTION" in Chapter 4.

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 non-numeric 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 Sigma EBCDIC set.

If the non-numeric 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 indicates characteristics of the compared items and the type of comparison made.

Table 5. Valid Class Tests

PICTURE				
Must Contain	May Contain	Allowable Characters	Valid Tests	Meaning
A	B	Alphabetic (A-Z and space)	[NOT] ALPHABETIC	(Not) Only characters A-Z and space appear.
A 9 X	X B 0 A 9 B 0	Alphanumeric (any character)	[NOT] ALPHABETIC [NOT] NUMERIC	(Not) Only characters A-Z and space appear. (Not) Only characters 0-9 appear.
S 9	0 V P	Zoned decimal with operational sign	[NOT] NUMERIC	(Not) Only characters 0-9 appear in all positions except the low-order position, which can contain zone bit.
9	0 V P	Zoned decimal without sign	[NOT] NUMERIC	(Not) Only characters 0-9 appear.

Table 6. Permissible Comparisons

Item Characteristics		GR	X	ND	C	FC
Group Item	GR	A	A	A	A	A
Alphabetic, Alphanumeric, and Edited	X	A	A	A		A^1
Numeric Display	ND	A	A	9	9	A^2
Computational, Index	C	A		9	9	9^1
Figurative-Constants	FC	A	A^1	A^2	9^1	

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

A^1 Alphanumeric comparison permitting any figurative-constants and ALL 'character'.

A^2 Alphanumeric comparison permitting the figurative-constants ZERO and ALL 'digit', where digit is a numeric character.

9 Numeric comparison.

9^1 Numeric comparison permitting the figurative-constant ZERO.

Note: A blank box indicates the test is not permitted.

Abbreviated Relations

In compound conditions consisting of relation conditions, it is permissible to omit the subject (the first identifier, literal, or arithmetic-expression) or the relational-operator according to the following situations.

Implied Subjects

A conditional expression may contain several, consecutive, simple relational conditions that may have common subjects. For example, the conditional expression AGE GREATER THAN 21 AND AGE LESS THAN 65 contains the common subject AGE. AGE can be "implied", that is, stated in the first simple relational condition and omitted in the second. Thus, if the second occurrence of AGE were implied, the expression would appear as AGE GREATER THAN 21 AND LESS THAN 65. The expression AGE GREATER THAN 21 AND LESS THAN 65 OR = 16 OR = 18 would be interpreted as AGE GREATER THAN 21 AND AGE LESS THAN 65 OR AGE = 16 OR AGE = 18.

The following rules specify the use of implied subjects:

1. Only relational conditions may have implied subjects. Sign conditions and class conditions can never have implied subjects.
2. Within a compound condition, the first of a series of simple relational conditions must always consist of a subject, relational-operator, and object, all of which must be explicitly stated.
3. Subjects may be implied only in a series of consecutive simple relational conditions connected by AND and/or OR.
4. When the subject of a simple relational condition is implied, the subject used is the first subject to the left that is explicitly stated. For example, A = B OR = C OR D = E AND = F is interpreted as A = B OR A = C OR D = E AND D = F, since D is the first stated subject to the left of = F.

Implied Relational-Operators

In some cases relational-operators may also be implied in a series of consecutive simple relational conditions in much the same way as subjects can be implied. Thus the expression AGE = 16 OR AGE = 18 OR AGE = 21 could be written AGE = 16 OR 18 OR 21. Not only is the subject AGE implied in the last two conditions, but the relational-operator = is also implied.

The following rules specify the use of implied operators:

1. A relational-operator can be implied only in a simple conditional relation where the subject is also implied.
2. When an operator is implied, the operator of the nearest simple relational condition to the left is taken.

Implied Logical Connectors

If in a consecutive sequence of relation conditions separated by logical operators the subjects are identical, the relational-operators are identical, and the logical connectors are identical, the sequence may be abbreviated. For example, A GREATER B OR A GREATER C OR A GREATER D can be abbreviated A GREATER B C OR D. Only the first occurrence of the subject and relational-operator is written. The logical operator is written only once and appears immediately preceding the last object.

Conditional Statements with Exception Branches

The format of these statements is

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

The READ, RETURN, WRITE, SEARCH, ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE 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

$$\begin{array}{ll}\underline{\text{OPEN}} & \left[\underline{\text{INPUT}}\left\{\text{file-name}\left[\begin{array}{l}\underline{\text{REVERSED}}\\ \text{WITH }\underline{\text{NO}}\text{ }\underline{\text{REWIND}}\end{array}\right]\right\}\ldots\right]\\ & \left[\underline{\text{OUTPUT}}\ \{\text{file-name [WITH }\underline{\text{NO}}\text{ }\underline{\text{REWIND}}]\}\ldots\right]\\ & \left[\left\{\begin{array}{l}\underline{\text{I-O}}\\ \underline{\text{INPUT-OUTPUT}}\end{array}\right\}\ \{\text{file-name}\}\ldots\right]\end{array}$$

The OPEN statement initiates processing of the files named in the statement. When applicable during the execution of an OPEN statement, standard label checking is performed for input and output files and label creation is done for output files. In addition, when a user label is specified, the associated DECLARATIVES section is executed as a subroutine.

At least one of the INPUT, OUTPUT, or INPUT-OUTPUT (I-O) options must be specified; options may appear in any order, but each must be used only once. The INPUT-OUTPUT or I-O option pertains only to files on random access media when ACCESS IS RANDOM is specified.

The OPEN statement must not be applied to Sort-Files. The OPEN statement may be circumvented for those FDs which contain RDs if the report generation is to be suppressed or partially suppressed (report restart); refer to Chapter 1, "Programming Hints", in the ANS COBOL OPS Reference Manual (90 15 01). For all other files, 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 be executed to obtain or release, respectively, the first data record.

The INPUT option institutes standard label-checking procedures upon ANS and Xerox monitor-fomatted tape files; permits label-checking in the DECLARATIVES section if a user label is specified; and permits subsequent reading of the file. The OUTPUT option initiates creation of files. Execution of an OPEN OUTPUT statement causes the ANS or Xerox monitor label to be written, and when a user label is specified, the DECLARATIVES section that creates that label is called as a subroutine. Label checking procedures are performed by the monitors on private disk packs but the action is transparent to the user. Conventional label checking is not applicable to public disk storage.

The REVERSED option should be specified only for input files with sequential access that are assigned to devices on which backward reading is implemented. This option causes the file to be positioned at the end-of-file point during the execution of the OPEN statement, so that the first READ statement execution obtains the last record in the file. The NO REWIND option should be specified only in connection with files assigned to devices on which rewinding is implemented.

READ Statement

The formats of this statement are

Format 1

READ file-name 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 access (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 access (Format 2) makes available a specific record from a relative or indexed file and allows execution of a specified series of imperative-statements if the contents of the associated ACTUAL KEY data item are found to be invalid.

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 without the CORRESPONDING option. 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, at which time subsequent CLOSE and OPEN statements for that file must be executed before any further attempt to READ can be honored.

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 ACTUAL KEY cannot be located in the file. The contents of the ACTUAL 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

$$\underline{\text{WRITE}}\ \text{record-name}\ [\underline{\text{FROM}}\ \text{identifier-1}]\ \left[\left\{\begin{matrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{matrix}\right\}\ \text{ADVANCING}\ \left\{\begin{matrix}\text{identifier-2 LINES}\\ \text{integer LINES}\\ \text{mnemonic-name}\end{matrix}\right\}\right]$$

Format 2

$$\underline{\text{WRITE}}\ \text{record-name}\ [\underline{\text{FROM}}\ \text{identifier-1}]\ ;\ \underline{\text{INVALID}}\ \text{KEY imperative-statement}$$

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

An OPEN OUTPUT 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 without the CORRESPONDING option.

Format 1 relates to files opened for sequential access. The ADVANCING option applies to files containing output destined to be printed. It is assumed that the first byte of the logical record is a carriage-control character, and the ADVANCING option causes information to be stored in this byte that instructs the printer to space the indicated number of lines. If the ADVANCING option is not specified in a WRITE statement addressed to a print file, the programmer is presumed to have produced the desired carriage-control character employing other source language statements and placed it in the first byte. Integer should be an unsigned integer, and identifier-2, similarly, should contain a non-negative integer. The line is printed BEFORE or AFTER the specified number of lines is spaced. Mnemonic-name should represent a single character that is the desired print order code to be supplied with the line. (The mnemonic-name is defined in "SPECIAL-NAMES Paragraph" under "CONFIGURATION SECTION" in Chapter 4.)

Format 2 is used for relative and indexed files. Statements following the INVALID KEY clause are executed when

1. No space exists on the storage media to accommodate the record.

2. The file is open for output and a record corresponding to the contents of the ACTUAL KEY already exists in the file.

When the INVALID KEY condition arises, no data transfer takes place, and the information in the record area remains available.

CLOSE Statement

The format of this statement is

$$\underline{\text{CLOSE}}\left\{\text{file-name}\left[\begin{matrix}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{matrix}\right]\left[\text{WITH}\left\{\begin{matrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\right]\right\}\ldots$$

The CLOSE statement terminates the processing of reels, units, and files. Execution of a CLOSE statement causes the standard monitor file 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.

The REEL or UNIT option instructs the monitor system to advance to the next volume of the file immediately instead of doing so automatically upon encountering the end of the current volume. The NO REWIND and LOCK options, although specifiable for language standard compatibility reasons, are ineffective in conjunction with the REEL/UNIT option and are treated as comments.

The LOCK option specifies that the processing of this file is complete and that the file is to be added to the monitor master file and the file-name is to be added to the system file directory. Normally, when a magnetic tape file is closed, the tape is rewound and a message is typed requesting the operator to dismount the reel. The NO REWIND option specifies that these actions are not to be taken, and the file is not repositioned.

SEEK Statement

The format of this statement is

$$\underline{\text{SEEK}}\ \text{file-name RECORD}$$

The SEEK statement initiates the access of a record in a relative or indexed file for subsequent reading or writing. However, Xerox monitor systems combine the seek and read or write operations into one continuous function, therefore the SEEK statement is treated as a comment and has no effect on the operation of the users program.

ACCEPT Statement

The format of this statement is

$$\underline{\text{ACCEPT}}\ \text{identifier}\ \left[\underline{\text{FROM}}\ \begin{Bmatrix}\underline{\text{CONSOLE}}\\ \text{mnemonic-name}\end{Bmatrix}\right]$$

The ACCEPT statement specifies acceptance of data from the console keyboard; it allows the entering of low-volume data such as information needed to initialize program switches, balance totals, or serial numbers. Mnemonic-name may be used to identify the hardware device. This identification is made in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. If mnemonic-name is specified, it must refer to CONSOLE.

Identifier must be an unedited DISPLAY data item whose length is not greater than 255 characters. If the keyed-in message is shorter than the length of the identifier, the remaining character positions on the right are space filled.

DISPLAY Statement

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

The DISPLAY statement enables low-volume data such as exception records or messages to be written on the operator console or the printer. If mnemonic-name is specified, it must refer to either CONSOLE or PRINTER in the SPECIAL-NAMES paragraph. Literal-1 or identifier-1 may be one of the figurative-constants; if so, only a single occurrence of the constant is displayed. 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 they are specified characters. Any remaining positions on a line at the end of the data transfer are left blank.

The maximum number of characters that may be displayed at any one time (the sum of the characters in the specified data items, literals, and figurative-constants in one DISPLAY statement) is 254 characters. Any number of literals and figurative-constants may be specified, up to one line. 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 non-numeric.

Data items with USAGE DISPLAY, literals, and figurative-constants are displayed without modification. Data items with other USAGEs are converted to EBCDIC representation and displayed as

1. COMPUTATIONAL and INDEX: sign plus 10 digits, i.e., ±9999999999
2. COMPUTATIONAL-1: sign, 8-digit number with decimal point, E (exponent), 2-digit signed exponent, i.e., ±9.9999999E±99
3. COMPUTATIONAL-2: same as COMPUTATIONAL-1, but with 16-digit number
4. COMPUTATIONAL-3: 1 to 31 zoned decimal digits

The programmer is cautioned not to use DISPLAY UPON PRINTER to request operator action. CONSOLE should be specified for this purpose.

Arithmetic Statements

The basic arithmetic operations are specified by the four verbs ADD, SUBTRACT, MULTIPLY, and DIVIDE. Using the fifth verb, COMPUTE, the user may specify any of the basic arithmetic operations with arithmetic-expressions.

Rules for Arithmetic Verbs

The following general rules apply to all five arithmetic verbs:

1. All literals specified in arithmetic statements must be numeric. Wherever it is legal to specify a literal, the figurative-constant ZERO(S)(ES) may be used. Other figurative-constants (including ALL any-literal) may not be used, since they are considered alphanumeric.

   An identifier used in an arithmetic statement must be an elementary item and must be numeric except when it is the operand of a CORRESPONDING option.

2. The maximum size of an operand is 18 decimal digits. If the entry for an operand in the DATA DIVISION specifies a size greater than 18 digits or if a literal contains more than 18 nonzero digits, an error is indicated at compilation time.

3. The items in an arithmetic statement may be mixed in usage as long as they are all numeric. Any necessary conversions are performed by the compiler, and 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 and SIZE ERROR options apply to all arithmetic statements. The GIVING option applies to all arithmetic statements except COMPUTE.

GIVING Option

If the GIVING option is written, the value of each identifier that follows the word GIVING is made equal to the calculated result of the arithmetic operation. Each identifier that follows GIVING is not used in the computation, and may contain editing symbols.

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 7 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

Table 7. 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

$$\underline{\text{ADD}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \left[\begin{array}{l}\text{, identifier-2}\\ \text{, literal-2}\end{array}\right] \ldots \text{, identifier-n } [\underline{\text{ROUNDED}}]\ [\text{; ON } \underline{\text{SIZE}}\ \underline{\text{ERROR}} \text{ imperative-statement}]$$

Format 2

$$\underline{\text{ADD}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \left[\begin{array}{l}\text{, identifier-2}\\ \text{, literal-2}\end{array}\right] \ldots\ \underline{\text{TO}} \text{ identifier-m } [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-n } [\underline{\text{ROUNDED}}]\right] \ldots [\text{; ON } \underline{\text{SIZE}}\ \underline{\text{ERROR}} \text{ imperative-statement}]$$

Format 3

$$\underline{\text{ADD}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} , \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\} \left[\begin{array}{l}\text{, identifier-3}\\ \text{, literal-3}\end{array}\right] \ldots\ \underline{\text{GIVING}} \text{ identifier-m } [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-n } [\underline{\text{ROUNDED}}]\right] \ldots [\text{; ON } \underline{\text{SIZE}}\ \underline{\text{ERROR}} \text{ imperative-statement}]$$

Format 4

$$\underline{\text{ADD}} \left[\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right] \text{ identifier-1 } \underline{\text{TO}} \text{ identifier-2 } [\underline{\text{ROUNDED}}]\ [\text{; ON } \underline{\text{SIZE}}\ \underline{\text{ERROR}} \text{ imperative-statement}]$$

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 value 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, A, the values of W, X, Y, Z, and A before and after execution are

	W	X	Y	Z	A
Before	2	7	8	12	10
After	2	7	8	29	27

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.

Format 4 adds the values of one or more selected elementary items within a group to the values of one or more selected elementary items within another group. For the pair of groups, the values of a selected pair of elementary items are added, and the result is placed in the selected item in the second group (identifier-2).

Selection of Pairs of Groups

A pair of items from a pair of groups is selected for addition if

1. Their names are identical.
2. The names of all higher level items in each group (the qualifier for each item), up to but not including the names of the groups, are identical.
3. Both items are numeric elementary items.
4. Both level-numbers are less than 50.
5. Neither of two otherwise matching items contains a REDEFINES or OCCURS clause.
6. When either of the groups from the pair of groups is described with an OCCURS clause, it is subscripted or indexed by the compiler.

Addition of Matched Pairs

After a pair meets the above requirements, operation is identical to that for Format 2. The rules that apply to the addition of items specified directly, as in Formats 1, 2, and 3 above, apply also to the addition of selected pairs.

Example:

Consider the PROCEDURE DIVISION statement

```
    ADD CORRESPONDING CAL TO NY ON SIZE ERROR PERFORM ERROR-ROUTINE
```

where the DATA DIVISION contains the following entries:

```
01  CAL.                                        01  NY.
    02  LUX.                                        02  LUX.
        03  COSM   PIC 9V99  VALUE 0.04.                03  JEWEL  PIC 9V99  VALUE 0.05.
        03  JEWEL  PIC 9V99  VALUE 0.07.                03  COSM   PIC 9V99  VALUE 0.02.
    02  PROP       PIC 9V99  VALUE 0.03.            02  PROP       PIC 9V99  VALUE 0.08.
    02  TYPEX      PIC X(5)  VALUE 'LARGE'.         02  TYPEX      PIC X(5)  VALUE 'SMALL'.
```

The value of the elementary items in CAL and NY before and after execution would be

	COSM OF CAL	JEWEL OF CAL	PROP OF CAL	TYPE OF CAL	COSM OF NY	JEWEL OF NY	PROP OF NY	TYPE OF NY
Before	0.04	0.07	0.03	LARGE	0.02	0.05	0.08	SMALL
After	0.04	0.07	0.03	LARGE	0.06	0.12	0.11	SMALL

SIZE ERROR and ROUNDED Options

When SIZE ERROR is used in conjunction with CORRESPONDING, the size error test is made only after the completion of all ADD operations. If any of the additions produce a size error condition, the resultant field for that addition remains unchanged and the imperative-statements specified in the SIZE ERROR clause are executed. When the ROUNDED option is used in conjunction with CORRESPONDING, it applies to all ADD operations.

SUBTRACT Statement

The formats of this statement are

Format 1

$$\underline{\text{SUBTRACT}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \begin{bmatrix}\text{, identifier-2}\\ \text{, literal-2}\end{bmatrix} \ldots \ \underline{\text{FROM}} \text{ identifier-m } [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-n } [\underline{\text{ROUNDED}}]\right] \ldots \ [\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement}]$$

Format 2

$$\underline{\text{SUBTRACT}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \begin{bmatrix}\text{, identifier-2}\\ \text{, literal-2}\end{bmatrix} \ldots \ \underline{\text{FROM}} \begin{Bmatrix}\text{identifier-m}\\ \text{literal-m}\end{Bmatrix} \underline{\text{GIVING}} \text{ identifier-n } [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-o } [\underline{\text{ROUNDED}}]\right] \ldots [\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement}]$$

Format 3

$$\underline{\text{SUBTRACT}}\ \left[\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right]\ \text{identifier-1}\ \underline{\text{FROM}}\ \text{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

The SUBTRACT statement subtracts the value of one or the sum of two or more numeric items from one or more items and stores the result(s) in one or more items.

Format 1 subtracts the sum of the operands preceding the word FROM from identifier-m, placing the result in identifier-m: then from identifier-n, placing the result in identifier-n, etc.

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

Format 3 (SUBTRACT CORRESPONDING) allows subtraction of corresponding items in different groups in one operation, similar to the ADD statement with a CORRESPONDING option.

Example:

Given the statement SUBTRACT A, 20 FROM B GIVING C, D, the values of the operands before and after execution are

	A	B	C	D
Before	10	80	90	100
After	10	80	50	50

MULTIPLY Statement

The formats of this statement are

Format 1

$$\underline{\text{MULTIPLY}}\ \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\}\ \underline{\text{BY}}\ \text{identifier-2}\ [\underline{\text{ROUNDED}}]\ \left[\text{, identifier-3}\ [\underline{\text{ROUNDED}}]\right]\ldots$$

$$[\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

Format 2

$$\underline{\text{MULTIPLY}}\ \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\}\ \underline{\text{BY}}\ \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\}\ \underline{\text{GIVING}}\ \text{identifier-3}\ [\underline{\text{ROUNDED}}]$$

$$\left[\text{, identifier-4}\ [\underline{\text{ROUNDED}}]\right]\ldots[\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

The MULTIPLY statement can be used to specify one or more multiplications per statement or multiplication with the value of several items set to the same 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. Another product is formed by multiplying the multiplicand (identifier-1 or literal-1) by the multiplier (identifier-3), and the value of identifier-3 is set to the product (and so on for every item named in the statement). A literal cannot be used in place of identifier-2 or -3.

Example:

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

	A	B	C
Before	10	20	30
After	10	200	300

Note that the values of operands B and C 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) and the value of identifier-3 and identifier-4 (and any other following identifiers) to be set to the product. Identifiers to the right of the reserved word GIVING may be numeric edited items.

Example:

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

	A	B	C	D
Before	5	10	20	30
After	5	10	50	50

Note that the values of operands A and B remain the same, while values of operands C and D change.

DIVIDE Statement

The formats of this statement are

Format 1

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED][, identifier-3 [ROUNDED]]...
       {literal-1   }

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

Format 2

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

            [, identifier-4 [ROUNDED]]...[; ON SIZE ERROR imperative-statement]
```

Format 3

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

            [, identifier-4 [ROUNDED]]...[; ON SIZE ERROR imperative-statement]
```

Format 4

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

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

Format 5

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} \ \underline{\text{BY}} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\} \ \underline{\text{GIVING}} \text{ identifier-3 } [\, \underline{\text{ROUNDED}} \,]$$

$$\underline{\text{REMAINDER}} \text{ identifier-4} [\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement}]$$

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 specifice rules applying only to this individual statement. Use of the SIZE ERROR and ROUNDED options is also discussed in the referenced paragraph.

Format 1 allows one or more different divisions, with different quotients stored as the values of the different items following INTO. The dividends (identifier-2, identifier-3, etc.) are each divided by the divisor (identifier-1 or literal-1) and the values of each of the dividends set to the value of the associated quotient. Literals cannot be used in place of identifiers-2, -3, etc. The size error condition results when the divisor is zero or the quotient contains more integer positions than are available.

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

Example:

Given the statement DIVIDE A INTO B, C, D, E, the values of the operands before and after execution are

	A	B	C	D	E
Before	5	10	15	20	25
After	5	2	3	4	5

Formats 2 and 3 allow the single quotient resulting from a division to be stored in one or more items. 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, -4, etc. If Format 3 is used, the dividend (identifier-1 or literal-1) is divided by the divisor (identifier-2 or literal-2), and the value of the resultant quotient becomes the new value of identifiers-3, -4, etc. Literals cannot be used in place of identifiers-3, -4, etc. Identifiers to the right of the reserved word GIVING may be numeric edited items.

Example:

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

	A	B	C	D	E
Before	5	10	15	20	25
After	5	10	2	2	2

COMPUTE Statement

The format of this statement is

$$\underline{\text{COMPUTE}} \text{ identifier-1 } [\underline{\text{ROUNDED}}] \left[\text{, identifier-2 } [\underline{\text{ROUNDED}}]\right] \ldots \underline{=} \left\{\begin{array}{l}\text{identifier-n}\\ \text{literal-1}\\ \text{arithmetic-expression}\end{array}\right\}$$

$$[\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ imperative-statement}]$$

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, editing the results to conform with the receiving item PICTURE. 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. (Refer to "Arithmetic Expressions" at the beginning of this chapter.) The arithmetic-expression is evaluated and the resulting numeric value replaces the contents of identifier-1 and any other items named before the =. 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:

1. COMPUTE QTY-ON-HAND = STOCK + RECEIPTS + RETURNS - ORDERS-FILLED.

2. COMPUTE D = A + B + C ON SIZE ERROR GO TO EXCESS-D.

3. COMPUTE VOLUME = 4 / 3 * PI * R ** 3.

ROUNDED and SIZE ERROR options are discussed under "Rules for Arithmetic Verbs" above.

Data Manipulation Statements

EXAMINE Statement

The format of this statement is

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

The EXAMINE statement replaces certain occurrences of a given character and counts the number of such occurrences in the specified data item. Each literal in the EXAMINE statement must consist of only one character. The USAGE of the identifier must be DISPLAY. EXAMINE is a subset of the INSPECT statement.

When the TALLYING option of this statement is used, a count is made at object time of the number of occurrences of the specified character in identifier-1; this count is placed in the special data-name TALLY, which may then be used as a data-name in other procedural statements. (Refer to "Special Registers" in Chapter 1.)

The count at object time depends on which TALLYING option is specified:

1. ALL counts all occurrences of literal-1 in the data item (identifier-1).

2. LEADING counts the number of occurrences of literal-1 prior to encountering a character other than literal-1 in the data item. Examination proceeds from left to right.

3. UNTIL FIRST counts the number of characters other than literal-1 encountered prior to the first occurrence of literal-1 in the data item. Examination proceeds from left to right.

Replacement of characters depends on which REPLACING BY option is specified:

1. ALL substitutes literal-2 for each occurrence of literal-1.

2. LEADING terminates the substitution of literal-2 for literal-1 when a character other than literal-1 is encountered, or when the right-hand boundary of the data item is reached. Examination proceeds from left to right.

3. UNTIL FIRST terminates the substitution of literal-2 when the first literal-1 is encountered or when the right-hand boundary is reached. Examination proceeds from left to right.

4. FIRST replaces only the first occurrence of literal-1 by literal-2. Examination proceeds from left to right.

INSPECT Statement

The INSPECT statement provides more comprehensive capabilities than EXAMINE. It has three formats: format 1 provides the ability to count, format 2 to replace, and format 3 to count and replace occurrences of single characters or groups of characters in a data item.

Format 1

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{TALLYING}}$$

$$\left\{ \text{, identifier-2}\ \underline{\text{FOR}} \left\{ \text{,} \left\{ \begin{array}{l} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-1} \end{array} \right\} \\ \underline{\text{CHARACTERS}} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \right] \right\} \ldots \right\} \ldots$$

Format 2

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{REPLACING}}$$

$$\left\{ \begin{array}{l} \underline{\text{CHARACTERS}}\ \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \\ \left\{ \text{,} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{FIRST}} \end{array} \right\} \left\{ \text{,} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \right\} \ldots \right\} \ldots \end{array} \right\}$$

Format 3

$$\underline{\text{INSPECT}}\ \text{identifier-1}\ \underline{\text{TALLYING}}$$

$$\left\{ \text{, identifier-2}\ \underline{\text{FOR}} \left\{ \text{,} \left\{ \begin{array}{l} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-1} \end{array} \right\} \\ \underline{\text{CHARACTERS}} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \right] \right\} \ldots \right\} \ldots$$

$$\underline{\text{REPLACING}}$$

$$\left\{ \begin{array}{l} \underline{\text{CHARACTERS}}\ \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\} \right] \\ \\ \left\{ \text{,} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{FIRST}} \end{array} \right\} \left\{ \text{,} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-3} \end{array} \right\} \underline{\text{BY}} \left\{ \begin{array}{l} \text{identifier-6} \\ \text{literal-4} \end{array} \right\} \left[\left\{ \begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array} \right\} \text{INITIAL} \left\{ \begin{array}{l} \text{identifier-7} \\ \text{literal-5} \end{array} \right\} \right] \right\} \ldots \right\} \ldots \end{array} \right\}$$

Identifier-1 must reference either a group item or any category of an elementary item, described implicitly or explicitly as USAGE IS DISPLAY. Identifier-3 through identifier-n must reference an elementary alphabetic, alphanumeric, or numeric item described implicitly or explicitly as USAGE IS DISPLAY. Literals must be non-numeric and may be any figurative constant except ALL.

In level 1, literal-1 through literal-5 and the data items referenced by identifier-3 through identifier-7 must be one character in length. Identifier-2 (formats 1 and 3 only) must reference an elementary numeric data item. The size of the data referenced by literal-4 or identifier-6 (in formats 2 and 3 only) must be equal to the size of the data referenced by literal-3 or identifier-5.

When the CHARACTERS phrase is used, literal-4, literal-5, or the data item referenced by identifier-6 or identifier-7 must be one character in length. Likewise, when a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length. In the context of the following rules, a figurative constant refers to an implicit one-character data item whose USAGE IS DISPLAY.

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, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 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 (see also MOVE statement, below).

Rules Applicable to Format 1

1. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.
2. The rules stated below for the ALL, LEADING, CHARACTERS, and BEFORE/AFTER phrases options apply to each iteration of the TALLYING phrase and to each permissible iteration of the phrase therein.
3. If ALL is specified, the data referenced by identifier-2 is increased by 1 for each nonoverlapping occurrence of literal-1 or the data referenced by identifier-3 within the data referenced by identifier-1.
4. If LEADING literal-1 or LEADING identifier-3 is used, the data referenced by identifier-2 is increased by 1 for each contiguous nonoverlapping occurrence of literal-1 or the data referenced by identifier-3 within the data referenced by identifier-1, provided that the leftmost such occurrence is at the point where counting begins (see Rule 5).
5. If the CHARACTERS phrase is used, the data referenced by identifier-2 is increased by 1 for each character in the data referenced by identifier-1.
6. If the BEFORE/AFTER phrase is absent, the process described in Rules 2 to 4 above begins at the leftmost character position of the data referenced by identifier-1 and proceeds to the rightmost character position of the data referenced by identifier-1.

7. If the BEFORE phrase is used, the counting begins at the leftmost character position and terminates at, but not including, the first occurrence of literal-2 or the data referenced by identifier-4 that occurs within the data referenced by identifier-1. If no occurrence of literal-2 or the data referenced by identifier-4 is encountered within the data referenced by identifier-1, then counting terminates at the rightmost character position of the data referenced by identifier-1.

8. If the AFTER phrase is used, the counting begins at, but not including, the first occurrence of literal-2 or the data referenced by identifier-4 that occurs within the data referenced by identifier-1, and continues to the rightmost character position.

Rules Applicable to Format 2

1. Rules governing format 2 apply as appropriate to each phrase in the INSPECT statement following the key word REPLACING, and to each iteration of those phrases.

2. The rules for replacement are as follows:

   a. When literal-3 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-4 or identifier-6.

   b. When literal-4 is a figurative constant, each character in the data referenced by identifier-1 that is equal to the character referenced by literal-3 or identifier-5 is replaced by the character referenced by the figurative constant.

   c. When literal-3, literal-4 or the data referenced by identifier-5, identifier-6 refer to character-strings, each nonoverlapping occurrence of the character-string referenced by literal-3 or identifier-5 is replaced by the character-string referenced by literal-4 or identifier-6.

   d. When the CHARACTERS phrase is used, any character encountered in the data item referenced by identifier-1 will be replaced by literal-4 or the character referenced by identifier-6.

   e. Once replacement has occurred in a given character position of the data referenced by identifier-1, no further replacement may be made in that character position by the same INSPECT statement.

3. If the CHARACTERS phrase is used, each character encountered will be replaced by literal-4 or the character referenced by identifier-6 from the point where replacement begins to the point where replacement ends.

4. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears:

   a. If ALL identifier-5/literal-3s are to be replaced, this is done according to the replacement rules specified in paragraph 2, above, and as described in paragraph 5, which follows.

   b. If the adjective LEADING is used, all contiguous, nonoverlapping occurrences of the character string referenced by literal-3 or identifier-5 are replaced by the character string referenced by literal-4 or identifier-6, provided that the leftmost such occurrence, in the data referenced by identifier-1, is at the point where replacement begins. See paragraph 5, below, for the BEFORE/AFTER case.

   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-3 or identifier-5 begins (see paragraph 5, below), is replaced, in the data referenced by identifier-1, by the character string referenced by literal-4 or identifier-6.

5. If the BEFORE/AFTER phrase is absent, the replacement process begins at the leftmost character position of the data referenced by identifier-1 and proceeds to the rightmost character position of the data referenced by identifier-1.

   a. If the BEFORE phrase is used, replacement begins at the leftmost character position and terminates at, but not including, the first occurrence of literal-5 or the data referenced by identifier-7 that is found within the data referenced by identifier-1. If no occurrence of literal-5 or the data referenced by identifier-7 is encountered within the data referenced by identifier-1, then replacement terminates at the rightmost character position of the data referenced by identifier-1.

b. If the AFTER phrase is used, replacement begins after, but not including, the first occurrence of literal-5 or the data referenced by identifier-7 that occurs within the data referenced by identifier-1, and continues to the rightmost character position.

Rules Applicable to Format 3

Format-1 rules apply to the TALLYING phrase of format 3, and format-2 rules to the REPLACING phrase. However, all tallying takes place before any replacing.

Examples:

INSPECT word TALLYING count FOR LEADING 'L' BEFORE INITIAL 'A', count-1 FOR LEADING 'A' BEFORE INITIAL 'L'.

Where word = LARGE, count = 1, count-1 = 0.
Where word = ANALYST, count = 0, count-1 = 1.

INSPECT word TALLYING count FOR ALL 'L', REPLACING LEADING 'A' BY 'E' AFTER INITIAL 'L'.

Where word = CALLAR, count = 2, word = CALLAR.
Where word = SALAMI, count = 1, word = SALEMI.
Where word = LATTER, count = 1, word = LETTER.

INSPECT word REPLACING ALL 'A' BY 'G' BEFORE INITIAL 'X'.

Where word = ARXAX, word = GRXAX.
Where word = HANDAX, word = HGNDGX.

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL 'J' REPLACING ALL 'A' BY 'B'.

Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
Where word = JACK, count = 3, word = JBCK.
Where word = JUJMAB, count = 5, word = JUJMBB.

INSPECT word REPLACING ALL 'X' BY 'Y', 'B' BY 'Z', 'W' BY 'Q' AFTER INITIAL 'R'.

Where word = RXXBQWY, word = RYYZQQY.
Where word = YZACDWBR, word = YZACDWZR.
Where word = RAWRXEB, word = RAQRYEZ.

INSPECT word REPLACING CHARACTERS BY 'B' BEFORE INITIAL 'A'.

Word before: 1 2   X Z A B C D
Word after: B B B B B A B C D

MOVE Statement

The format of this statement is

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

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. When the CORRESPONDING option is used, selected items within identifier-1 are moved, together with any required editing, to selected areas within identifier-2. Items are selected by matching data-names of areas defined by identifier-1 with like data-names of areas defined by identifier-2.

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, with the exception of ZERO(S)(ES), 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 two types of MOVE statements are discussed in the following paragraphs.

Alphanumeric Moves

Source data is stored left-justified in the receiving area unless the receiving item is an elementary item that specifies JUSTIFIED RIGHT. If a group is moved, left justification is standard, and any specification to the contrary is overridden. 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:

Source Data	PICTURE of Receiving Item	Receiving Item
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:

Source Data	PICTURE of Receiving Item	Receiving Item
'ABCD'	X(4)	A B C D
ALL 'ABCD'	X(7)	A B C D A B C
'123'	X(2)	1 2
ALL '123'	X(7)	1 2 3 1 2 3 1
ALL '123'	X(7) JUSTIFIED RIGHT	3 1 2 3 1 2 3
ALL 123	X(7)	Illegal
123	X(5)	1 2 3 △ △
[ALL] QUOTES	X(5)	' ' ' ' '

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 at 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
1 2 ↑ 3	99V9	1 2 ↑ 3
1 2 ↑ 3	999V99	0 1 2 ↑ 3 0
1 2 3 ↑	9999	0 1 2 3 ↑
1 2 ↑ 3	9999	0 0 1 2 ↑
-1.23 (literal)	S9V99	1 ↑ 2 3̄
1 ↑ 2 3	9V9	1 ↑ 2
1 2 ↑ 3	9V9	2 ↑ 3

Editing

If the receiving item format specifies editing, the source data is 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:

Source Data	PICTURE of Receiving Item	Receiving Item
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 or the figurative-constant ZERO(S)(ES). 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:

Source Data	PICTURE of Receiving Item	Receiving Item
+1.23	S9V99	1 ↑ 2 $\overset{+}{3}$
+1.23	S9V9	1 ↑ $\overset{+}{2}$
123	9(5)	0 0 1 2 3
ZEROS	S99999	0 0 0 0 $\overset{+}{0}$
QUOTES	9999	Illegal
+37	S999V99	0 3 7 ↑ 0 $\overset{+}{0}$
ALL 37	S999V99	Illegal
ALL '37'	S999V99	Illegal
03737.3	$***9.9	$ 3 7 3 7 . 3

Permissible Moves

Permissible moves are listed in Table 8.

Table 8. Permissible Moves

Source Item		Receiving Field			
		GR	X	ND	C
Group	GR	A	A	A	A
Alphabetic, Alphanumeric, or Edited	X	A	A	9[1]	
Numeric Display	ND	A	A[1]	9	9
Computational	C	A	A[1]	9	9
Figurative-Constant		A	A[1]	A[2]	9[2]
Numeric Literal		A	A[1]	9	9
Non-numeric Literal		A	A	9[1]	

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

A[1] 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 (unless RIGHT JUSTIFIED is specified in DATA DIVISION).

A[2] Only the figurative-constant ZERO and ALL 'digit', where digit is a numeric character, are allowed.

9 Numeric MOVE.

9[1] Any non-numeric characters in the source field cause unpredictable data.

9[2] Only the figurative-constant ZERO is allowed.

Note: A blank box indicates the test is not permitted.

CORRESPONDING Option

The CORRESPONDING option in the MOVE statement moves one or more items within one group to the location of selected items within one or more other groups. The items moved depend on the names of the items within the source and receiving groups.

The rules stated for the simple MOVE statement apply to each pair of corresponding items in the MOVE CORRESPONDING statement; thus, the effect of a MOVE CORRESPONDING statement is equivalent to a series of simple MOVE statements. Each selected item is moved from the source area to the corresponding item in the receiving area; editing according to the format of the receiving area takes place concurrently with the move.

An item is selected for movement if

1. There is a like-named item in the receiving area.
2. The names of all higher level items in each area (the qualifiers for the pair of items), up to but not including identifier-1 and identifier-2, identifier-3, etc., are also identical.
3. At least one of the items of a pair of matching items is an elementary item.
4. Identifier-1, identifier-2, etc., are group items.
5. Neither identifier-1 nor identifier-2 are data items with level-numbers of 66, 77, or 88.
6. It does not include a data item subordinate to identifier-1 or identifier-2 that contains a REDEFINES or OCCURS clause in its description. However, items designated by identifier-1 and identifier-2 may be described with REDEFINES or OCCURS clauses or be subordinate to items described with REDEFINES or OCCURS clauses.
7. Either identifier-1 or identifier-2, when described with an OCCURS clause, is subscripted; each data item that corresponds is subscripted by the compiler.

STRING Statement

The form of this statement is

$$\underline{\text{STRING}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \begin{bmatrix}\text{, identifier-2}\\ \text{, literal-2}\end{bmatrix}\ldots\ \underline{\text{DELIMITED}}\ \text{BY} \begin{Bmatrix}\text{identifier-3}\\ \text{literal-3}\\ \underline{\text{SIZE}}\end{Bmatrix}$$

$$\left[\text{,} \begin{Bmatrix}\text{identifier-4}\\ \text{literal-4}\end{Bmatrix} \begin{bmatrix}\text{, identifier-5}\\ \text{, literal-5}\end{bmatrix}\ldots\ \underline{\text{DELIMITED}}\ \text{BY} \begin{Bmatrix}\text{identifier-6}\\ \text{literal-6}\\ \underline{\text{SIZE}}\end{Bmatrix}\right]\ldots$$

$$\underline{\text{INTO}}\ \text{identifier-7}\ [\text{WITH}\ \underline{\text{POINTER}}\ \text{identifier-8}]$$

$$\text{\_; ON}\ \underline{\text{OVERFLOW}}\ \text{imperative-statement}]$$

The STRING statement allows juxtaposing the partial or complete contents of two or more data items so that they form a single data item. Each literal may be any figurative constant, without the optional word ALL, and all literals must be described as nonnumeric literals. Identifier-3 and identifier-6 must reference a fixed-length data item, and the usage of all identifiers, except identifier-8, must be described implicitly or explicitly as DISPLAY. Identifier-7 must represent an elementary data item without editing symbols; identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus one of the area referenced by identifier-7.

All references to identifier-1, identifier-2, identifier-3, literal-1, literal-2, literal-3 apply equally to identifier-4, identifier-5, identifier-6, literal-4, literal-5 and literal-6, respectively, and all recursions thereof.

Identifier-1, literal-1, identifier-2, literal-2, represent the sending items. Identifier-7 represents the receiving item.

Literal-3, identifier-3, indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1, literal-1, identifier-2, literal-2, is moved. When a figurative constant is used as the delimiter, it stands for a single-character non-numeric literal.

When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit one-character data item whose usage is display.

When the STRING statement is executed, the transfer of data is governed by the following rules:

1. Characters from literal-1, literal-2, or from the contents of the data item, referenced by identifier-1, identifier-2, are transferred to the contents of identifier-7 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling will be provided (see "MOVE Statement").

2. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referenced by identifier-1, identifier-2, or the value of literal-1, literal-2, are transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character and continuing from left to right until the end of the data item is reached, or until the character(s) specified by literal-3 or by the contents of identifier-3 are encountered. The character(s) specified by literal-3 or by the data item referenced by identifier-3 are not transferred.

3. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or the contents of the data item referenced by identifier-1, identifier-2, are transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-7 until all data has been transferred or the end of the data item referenced by identifier-7 has been reached.

If the POINTER phrase is specified, identifier-8 is explicitly available to the programmer, and he is responsible for setting its initial value. The initial value must not be less than 1. If the POINTER phrase is not specified, an initial value of 1 is assumed.

When characters are transferred to the data item referenced by identifier-7, the moves behave as though the characters were moved one at a time from the source into the character position of the data item referenced by identifier-7, designated by the value associated with identifier-8, and increased by one prior to the move of each next character. This is the only way in which the value associated with identifier-8 is changed during execution of the STRING statement.

At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-7 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-7 will contain data that was present before this execution of the STRING statement.

If at any point at or after initialization of the STRING statement, but before execution of the STRING statement is completed, the value associated with identifier-8 is either less than one or exceeds the number of character positions in the data items referenced by identifier-7, no (further) data is transferred to the data item referenced by identifier-7 and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

If the ON OVERFLOW phrase is not specified, control passes to the next executable statement as written, or to the return mechanism of a PERFORM or a USE statement.

UNSTRING Statement

The form of this statement is

```
UNSTRING identifier-1

  [DELIMITED BY [ALL] {identifier-2} [, OR [ALL] {identifier-3}] ...]
                      {literal-1   }              {literal-2   }

   INTO identifier-4 [, DELIMITER IN identifier-5] [, COUNT IN identifier-6]

       [, identifier-7 [, DELIMITER IN identifier-8][, COUNT IN identifier-9] ...

   [WITH POINTER identifier-10] [TALLYING IN identifier-11]

   [; ON OVERFLOW imperative-statement]
```

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields. Each literal must be a non-numeric literal and may be any figurative constant without the optional word ALL.

Identifier-1, identifier-2, identifier-3, identifier-5, and identifier-8 must be described, implicitly or explicitly, as an alphanumeric data item. Identifier-4 and identifier-7 must be described as either alphabetic (except that the symbol 'B' may not be used in the PICTURE character-string), alphanumeric, or numeric (except that the symbol 'P' may not be used in the PICTURE character-string), and must be described as USAGE IS DISPLAY.

Identifier-6, identifier-9, identifier-10, and identifier-11 must be described as elementary numeric integer data items. No identifier may name a level 88 entry. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

All references to identifier-2, literal-1, identifier-4, identifier-5 and identifier-6, apply equally to identifier-3, literal-2, identifier-7, identifier-8, and identifier-9, repsectively, and all recursions thereof.

Identifier-1 represents the sending area. Identifier-4 represents the data receiving area, and identifier-5 represents the receiving area for delimiters, specified by literal-1 or the data item referenced by identifier-2. When a figurative constant is used as the delimiter, it stands for a single-character non-numeric literal.

Identifier-6 represents the count of the number of characters within the data item referenced by identifier-1, isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter character(s).

The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1; the data item referenced by identifier-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.

When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of literal-1 (figurative constant or not) or the contents of the data item referenced by identifier-2 are treated as if it were only one occurrence, and this occurrence is moved to the receiving data item according to the rules applicable to the DELIMITER IN phrase, below. When an examination encounters two contiguous delimiters, the current receiving area is either space-filled or zero-filled according to the description of the receiving area.

Literal-1 and the data item referenced by identifier-2 each represent one delimiter. When a delimiter contains two or more characters, all of the characters of the sending item must be present in contiguous positions and in the order given to be recognized as a delimiter. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computer's character set.

When two or more delimiters are specified in the DELIMITED BY phrase, an 'OR' condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter. Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

1. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the content of the data item referenced by identifier-10. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

2. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by the value of literal-1 or the data item referenced by identifier-2 is encountered. If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

   If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

3. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item and are moved into the current receiving area according to the rules for the MOVE statement (see above).

4. If the DELIMITER IN phrase is specified, the delimiting character(s) is treated as an elementary alphanumeric data item and is moved into the data item referenced by identifier-5 according to the rules for the MOVE statement (see above). If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space-filled.

5. If the COUNTER IN phrase is specified, a value equal to the number of characters thus examined (exluding the delimiter character(s), if any) is moved into the area referenced by identifier-6 according to the rules for an elementary move.

6. If the DELIMITED BY phrase is specified, the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

7. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The data transfer is repeated as described until either all the characters are exhausted in the data item referenced by identifier-1 or until there are no more receiving areas.

Initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user. The contents of the data item referenced by identifier-10 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the contents of the data item referenced by identifier-10 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referenced by identifier-11 contains a value equal to its initial value plus the number of receiving items acted upon.

Either of the following situations causes an overflow condition:

1. An UNSTRING is initiated, and the value in the data item referenced by identifier-10 is less than 1 or greater than the size of the data item referenced by identifier-1.

2. All receiving areas have been acted upon during execution of an UNSTRING statement and the data item referenced by identifier-1 contains characters that have not been examined.

When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative statement included in the ON OVERFLOW phrase is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next executable sentence.

Evaluation of subscripting and indexing for the identifiers is as follows:

1. Any subscripting or indexing associated with identifier-1, identifier-10, identifier-11 is evaluated only once, immediately before any data is transferred as the result of the execution of the UNSTRING statement.

2. Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, identifier-6 is evaluated immediately before the transfer of data into the respective data item.

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. ALTER changes the procedure-name in a GO TO elsewhere in the program so that sometime later, when that GO TO is encountered, it will release control to that location.

3. PERFORM causes statements in a remote procedure to be executed and control returned to the statement following the PERFORM.

4. STOP allows the program to terminate in an orderly manner.

5. 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 un'ess a GO TO branch is used in one or both paths.

6. EXIT merely declares that the paragraph in which it is contained is a transfer point that may be referenced by other sequence control statements.

7. ENTER allows the execution of non-COBOL subroutines. Paragraph and section names identify procedures so that they may be referenced by sequence control statements.

Normal Sequence Control

The starting location for the program is at the first statement of the PROCEDURE DIVISION, excluding the DECLARATIVES section. 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.

Loading of Priority Segments

Sections that have a priority-number equal to or exceeding the SEGMENT-LIMIT are automatically loaded into the overlay area when control is given to them either by sequence control statements or by normal control flow from the preceding paragraph. GO TO statements changed by the ALTER statement are completely unaffected by loading of priority segments. It is permissible for PERFORM to name a procedure beginning in one priority segment and ending in another. The second and successive priority segments are automatically loaded when needed, and the location of the controlling PERFORM statement is retained even if it is within yet another priority segment. Refer to Chapter 11, "Priority Segmentation", for additional discussion of this feature.

Reference to Unnamed Procedures

Any sequence control statements referencing procedures undefined in the COBOL program being compiled are diagnosed as undefined, but linkage is generated so that the procedure can be defined in a separately compiled COBOL program and the cross-reference handled by the Linkage Editor of the Monitor system. To do this, the procedure in the separately compiled COBOL program must be declared as an entry point by an ENTER COBOL statement. Refer to "ENTER Statement" later in this chapter for additional description of this feature.

GO TO Statement

The formats of this statement are

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. If procedure-name-1 is not specified, the GO TO statement must occupy a paragraph by itself and be assigned a paragraph-name. Procedure-name-1 is then supplied – using the assigned paragraph-name as a reference – by an ALTER statement prior to the first execution of the GO TO statement. If the execution of an

ALTER statement does not assign a procedure-name to the GO TO statement before statement execution, the object-time diagnostic is issued.

Examples:

1. GO TO TEST-ROUTINE.

2. IF A = 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. The USAGE of identifier-1 is either DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3, and 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 100 procedure-names may be used in one GO TO statement.

Example:

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

ALTER Statement

The format of this statement is

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

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

The ALTER statement modifies the branch point in an unconditional GO TO statement by supplying an alternate branch point, thus setting up a predetermined sequence of operations. A GO TO statement that is to be altered must be unconditional and must be the only statement contained within a paragraph, preceded by a paragraph-name. Procedure-names-1, -3, etc., are the names of paragraphs that contain only GO TO statements that are to be altered; when executed, these GO TO statements cause control to be transferred to procedure-names-2, -4, etc.

Example:

Assume GO TO and ALTER statements

ALTER SWITCH-5 TO PROCEED TO WRITE-IT-OUT.

SWITCH-5. GO TO PRINT-IT-OUT.

The effect would be to change the GO TO statement to

SWITCH-5. GO TO WRITE-IT-OUT.

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} TIMES
                                                 {integer-1   }
```

Format 3

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

Format 4

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

                                {index-name-2}
VARYING {index-name-1} FROM {identifier-2} BY {identifier-3} UNTIL condition-1
        {identifier-1}      {literal-2   }    {literal-3   }

                                {index-name-5}
[AFTER {index-name-4} FROM {identifier-5} BY {identifier-6} UNTIL condition-2
       {identifier-4}      {literal-5   }    {literal-6   }

                                {index-name-8}
[AFTER {index-name-7} FROM {identifier-8} BY {identifier-9} UNTIL condition-3]]
       {identifier-7}      {literal-8   }    {literal-9   }
```

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 the 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 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 3 illustrate the logic of the PERFORM statement when one, two, or three identifiers are varied. Let

1. Each d_i represent an identifier or index-name.
2. Each l_i represent a literal.
3. Each c_i represent a condition.
4. Each p_i represent a procedure-name.

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 = 11 AFTER CITY FROM 1 BY 1 UNTIL CITY IS GREATER THAN 12.
```

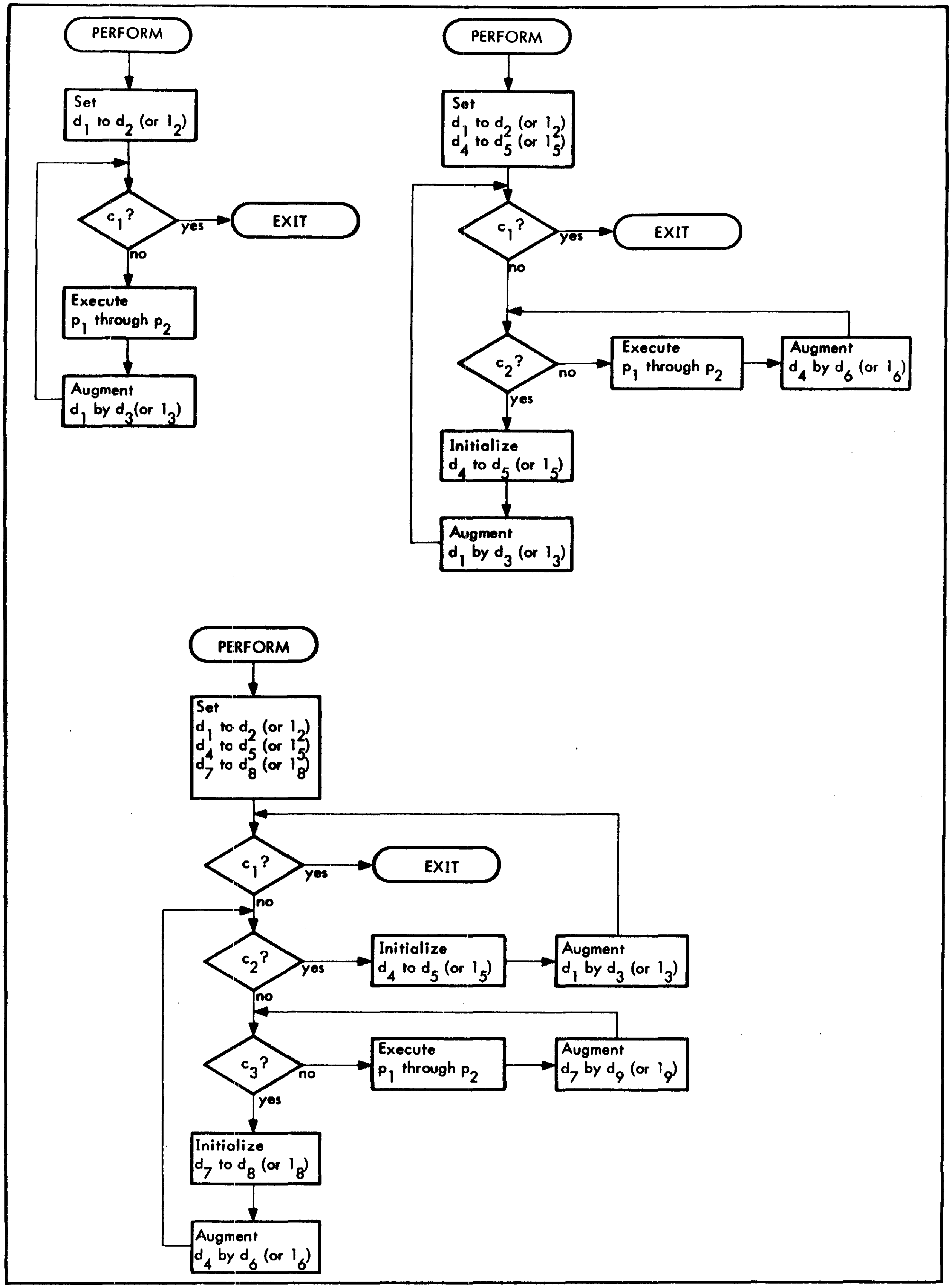

PERFORM
Set
d_1 to d_2 (or 1_2)
c_1?
yes
EXIT
no
Execute
p_1 through p_2
Augment
d_1 by d_3(or 1_3)
PERFORM
Set
d_1 to d_2 (or 1_2)
d_4 to d_5 (or 1_5)
c_1?
yes
EXIT
no
c_2?
no
Execute
p_1 through p_2
Augment
d_4 by d_6 (or 1_6)
yes
Initialize
d_4 to d_5 (or 1_5)
Augment
d_1 by d_3 (or 1_3)
PERFORM
Set
d_1 to d_2 (or 1_2)
d_4 to d_5 (or 1_5)
d_7 to d_8 (or 1_8)
c_1?
yes
EXIT
no
c_2?
yes
Initialize
d_4 to d_5 (or 1_5)
Augment
d_1 by d_3 (or 1_3)
no
c_3?
no
Execute
p_1 through p_2
Augment
d_7 by d_9 (or 1_9)
yes
Initialize
d_7 to d_8 (or 1_8)
Augment
d_4 by d_6 (or 1_6)

Figure 3. PERFORM Statement (VARYING Option)

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

$$\underline{\text{STOP}}\left\{\begin{array}{l}\text{literal}\\ \underline{\text{RUN}}\end{array}\right\}$$

The STOP statement temporarily or 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 until the operator takes steps to restore control after the STOP statement. Any literal or any figurative-constant except ALL may be used.

It is recommended that the programmer include a message for the operator to press the NEW LINE key to continue the run.

EXIT Statement

The format of this statement is

$$\text{paragraph-name.}\ \ \underline{\text{EXIT}}.$$

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 or to a NOTE paragraph, and generates no code.

IF Statement

The format of this statement is

$$\underline{\text{IF}}\ \text{condition}\left\{\begin{array}{l}\text{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\left[\underline{\text{ELSE}}\left\{\begin{array}{l}\text{statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\right]$$

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 represented 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

$$\left\{\begin{array}{l}
\left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\\ \text{formula-1}\end{array}\right\}\left\{\text{IS}\,[\underline{\text{NOT}}]\left\{\begin{array}{l}\underline{\text{GREATER}}\ \text{THAN}\\ \underline{>}\\ \underline{\text{LESS}}\ \text{THAN}\\ \underline{<}\\ \underline{\text{EQUAL}}\ \underline{\text{TO}}\\ \underline{=}\end{array}\right\}\right\}\left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\\ \text{formula-2}\end{array}\right\}\\
\left\{\begin{array}{l}\text{identifier-3}\\ \text{formula-3}\end{array}\right\}\text{IS}\ [\underline{\text{NOT}}]\left\{\begin{array}{l}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{array}\right\}\\
[\text{identifier-4}]\ \text{IS}\ [\underline{\text{NOT}}]\left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{array}\right\}\\
[\underline{\text{NOT}}]\left\{\begin{array}{l}\text{condition-name}\\ \text{switch-status-name}\end{array}\right\}
\end{array}\right\}$$

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 uppercase 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.
```

The ENTER COBOL statement (Format 3) makes the locations of the named procedures available to the loader as entry points. Procedure-names are paragraph-names or section-names to which control may pass.

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 formats of this statement are

Format 1

$$\underline{\text{SEARCH}}\ \text{identifier-1}\ \left[\underline{\text{VARYING}}\left\{\begin{matrix}\text{index-name-1}\\ \text{identifier-2}\end{matrix}\right\}\right]\ [;\ \text{AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]$$

$$;\ \underline{\text{WHEN}}\ \text{condition-1}\left\{\begin{matrix}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\left[;\ \underline{\text{WHEN}}\ \text{condition-2}\left\{\begin{matrix}\text{imperative-statement-3}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\right]\ldots$$

Format 2

$$\underline{\text{SEARCH}}\ \underline{\text{ALL}}\ \text{identifier-1}\ [;\ \text{AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]\ ;\ \underline{\text{WHEN}}\ \text{condition-1}\left\{\begin{matrix}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}$$

The SEARCH statement searches a table for a table element that satisifies the specified condition and adjusts the associated index-name to indicate that table element. In both Formats 1 and 2, identifier-1 may not be subscripted or indexed, but its description must contain an OCCURS and an INDEXED BY clause; in addition, the description of identifier-1 in Format 2 must also contain the KEY IS option in its OCCURS 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.

In Format 1, condition-1, condition-2, etc., may be any condition described under "Conditional Statements" at the beginning of this chapter. In Format 2, condition-1 may consist of a relation condition incorporating the relation EQUAL TO, or a condition-name condition where the VALUE clause that describes the condition-name contains only a single literal. Alternatively, condition-1 may be a compound condition formed from simple conditions of the type just mentioned, with AND as the only connective. Any data-name that appears in the KEY clause of identifier-1 may appear as the subject or object of a test, or be the name of the conditional variable with which the tested condition-name is associated. All data-names in the KEY clause must also be tested within condition-1: no other tests may appear within condition-1.

If Format 1 is used, 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 Format 2 is used, a binary search operation is performed wherein the initial setting of the index-name for identifier-1 is ignored and its setting is varied during search in the manner dictated by the binary search technique, with the following restrictions: At no time is the index-name set to a value exceeding that which corresponds to the last element of the table, or to a value less than that which corresponds to the first element of the table. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END clause appears or to the next sentence when this clause does not appear; in either case, the final index setting is not predictable. If condition-1 can be satisfied, the index indicates an occurrence that allows condition-1 to be satisfied and control passes to imperative-statement-2.

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 the Format 1 SEARCH operation containing two WHEN phrases is shown in Figure 4.

SET Statement

The formats of this statement are

Format 1

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

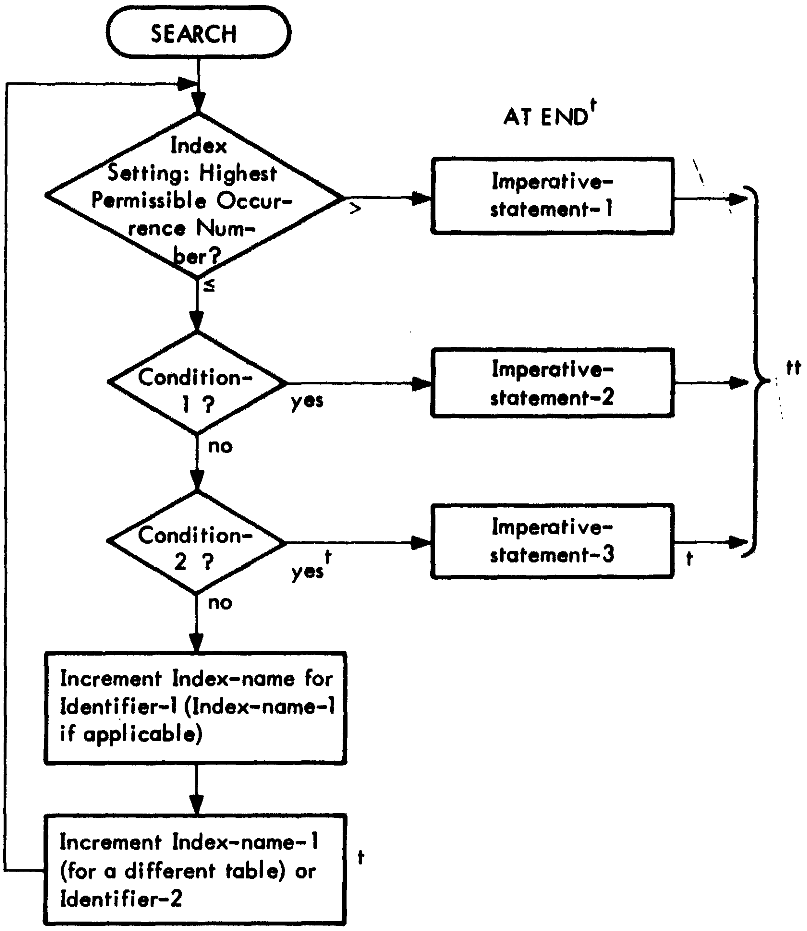
Format 2

$$\underline{\text{SET}}\ \text{index-name-4}\ [\text{, index-name-5}] \ldots \left\{\begin{array}{l}\underline{\text{UP BY}}\\ \underline{\text{DOWN BY}}\end{array}\right\} \left\{\begin{array}{l}\text{identifier-4}\\ \text{literal-2}\end{array}\right\}$$

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-4 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.

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


SEARCH
AT END†
Index Setting: Highest Permissible Occurrence Number?
>
≤
Imperative-statement-1
Condition-1 ?
yes
no
Imperative-statement-2
††
Condition-2 ?
yes†
no
Imperative-statement-3
†
Increment Index-name for Identifier-1 (Index-name-1 if applicable)
Increment Index-name-1 (for a different table) or Identifier-2
†

[†] These operations are only included when called for in the statement.

[††] Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

Figure 4. SEARCH Operation (Two WHEN Phrases)

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-3, identifier-3, or literal-1 corresponds. If identifier-3 is an index data item or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

2. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where 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-3; neither identifier-3 nor literal-1 can be used.

In Format 2 the value of index-name-4 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-4.

Compiler-Directing Statements

NOTE Statement

The format of this statement is

```
    NOTE comment.
```

The NOTE statement permits the writing of explanatory comments in the PROCEDURE DIVISION of a source program; such comments are printed on the program listing, but do not affect compilation. NOTE, when used, must begin a sentence. If NOTE is not the first word in a paragraph, the comment ends with a period followed by a space; if NOTE is the first word of a paragraph, any subsequent sentences within that paragraph are also considered notes. Proper format rules for paragraph structure and word composition must be observed.

The NOTE statement may be used only in the PROCEDURE DIVISION. However, comments may be written anywhere in the source program by placing an asterisk in column 7. The asterisk causes the compiler to treat the entire source line as comments.

USE Statement

The formats of this statement are

Format 1

```
                                          ⎧file-name-1 [, file-name-2]...⎫
                                          ⎪INPUT                         ⎪
    USE AFTER STANDARD ERROR PROCEDURE ON ⎨OUTPUT                        ⎬
                                          ⎪INPUT-OUTPUT                  ⎪
                                          ⎩I-O                           ⎭
```

Format 2

```
                                                                      ⎧file-name-1 [, file-name-2]...⎫
                                             ⎡REEL⎤                   ⎪INPUT                         ⎪
    USE ⎧BEFORE⎫ STANDARD ⎡BEGINNING⎤ ⎢FILE⎥ LABEL PROCEDURE ON ⎨OUTPUT                        ⎬
        ⎩AFTER ⎭          ⎣ENDING   ⎦ ⎣UNIT⎦                   ⎪INPUT-OUTPUT                  ⎪
                                                                      ⎩I-O                           ⎭
```

Format 3

```
    USE BEFORE REPORTING data-name.
```

A USE statement is valid only when written as the first statement in a section within the DECLARATIVES portion of the source program. USE is not an executable statement; rather, it defines the conditions under which the associated procedure, the DECLARATIVES section itself, is to be executed.

Format 1 specifies procedures to be followed if an input/output error occurs during file processing. The user-specified procedures are performed after the standard Monitor input/output error procedure is executed. A CLOSE statement may be written but no other input/output statements addressed to the offending file may appear within the error procedure.

Format 2 enables the user to create or examine his own file labels; it is effective only when a user label (LABEL RECORD IS record-name) is indicated for the file.

If the BEFORE option is specified, the DECLARATIVES section is performed before the label procedure is executed. Likewise, if the AFTER option is specified, the DECLARATIVES section is performed after the label procedure is executed.

BEGINNING labels refer to header labels, and ENDING labels refer to trailer labels. If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

If neither UNIT, REEL, nor FILE is included, the designated procedures are executed for both REEL or UNIT, whichever is appropriate, and FILE labels. The REEL option is not applicable to mass storage files. The UNIT option is not applicable to files in the random access mode.

Format 3 specifies PROCEDURE DIVISION statements to be executed prior to the reporting group named in the REPORT SECTION of the DATA DIVISION, i.e., it designates a procedure to be performed by the Report Writer before the report group indicated by the data-name is produced. The data-name may name a report group of any type except DETAIL. No Report Writer verbs may occur within procedures associated with this type of USE statement.

General Rules

1. When the conditions are met which cause the associated procedure to be executed that is defined by a USE statement, then a PERFORM is implied of the section containing the USE. After execution of a USE procedure, control is returned to the invoking routine.

2. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

COPY Statement

The format of this statement is

$$\left\{\begin{array}{l}\text{paragraph-name.}\\ \text{section-name }\ \underline{\text{SECTION}}\,[\text{priority-number}]\,.\end{array}\right\}\ \text{copy-statement.}$$

The COPY statement incorporates previously coded library source program statements from a user program into the PROCEDURE DIVISION of a source program. These statements must comprise a complete paragraph or section. The paragraph or section is copied from the library during compilation, and the result is the same as if the library statements were actually a part of the source program being compiled. In the PROCEDURE DIVISION, the verb INCLUDE may be used in place of the verb COPY. See Chapter 9, "COBOL Library", for a more detailed description.

7. REPORT WRITER

Introduction

The Report Writer allows the user to indicate the format of printed reports to be produced by the XDS ANS COBOL program. Each printed report is defined in the REPORT SECTION of the DATA DIVISION using the formats described in this chapter. Statements in the PROCEDURE DIVISION cause the report to be written on a file in the specified format. More than one report can be produced by a single source program.

If a report is to be defined, the REPORT SECTION must be included as the last section of the DATA DIVISION, and one or more File Description (FD) entries in the FILE SECTION must contain the names of the reports to be produced.

General Description

In describing a report both the data and the format of the data must be described. The format must be planned in terms of page width and length, and the organization of data on the page.

The concept of levels is used in organizing a report. Each report is divided into report groups, which may be further divided into group items and elementary items. A report group is a set of related data within a report presented on one or more lines of print. Any group or item within a report group that contains subordinate items is referred to as a group item; an item that contains no subordinate items is an elementary item. The highest level is the report itself, which is denoted by the level indicator RD followed by the report name. Next are report groups at 01 level; these contain lower level group or elementary items with level-numbers from 2 through 49. Each group at the 01 level is defined according to type: heading group, footing group, or detail group. The user may refer to the report-name and to DETAIL report groups from the main body of the PROCEDURE DIVISION. He may refer from the DECLARATIVES area to other report groups. He may also refer to elementary items if they are named and contain a SUM clause.

Every report description must contain a Report Description (RD) entry and a Report Group Description. The RD entry specifies the overall format: characteristics of the page; limits for the page and for heading, detail, and footings information within the page. This entry also specifies data items that control the printing of summary data or CONTROL FOOTINGs. Each report must be associated with an output file by being defined by a REPORT IS clause in an FD.

Each report group consists of one or more data items arranged in one or several lines. A report group must have a 01 level-number and contain a TYPE clause. The data-name is optional; however, it must be specified with any report group if it is referenced by statements in the PROCEDURE DIVISION. The order in which report groups are specified is not important (unless the Subcompile feature is used), since the position of a group in the report is determined by the TYPE clause.

Control Groups

It may be desirable to provide summary information within the body of a report. The concept of a control hierarchy makes it possible to produce required summary information automatically together with any heading and footing information for a control group. Control fields are specified in the RD in the same order as the control hierarchy. Any change in the contents of a control field produces a control break. Changes are recognized at each execution of a GENERATE statement and set in motion the automatic production of CONTROL HEADING and FOOTING report groups. The set of CONTROL HEADING, associated DETAIL report groups, and CONTROL FOOTING constitute a control group for a given CONTROL field. Within the hierarchy, lower level heading and footing report groups are included in a higher level control group.

Page/Overflow Conditions

PAGE HEADING and PAGE FOOTING clauses, if specified in a report, are mutually exclusive for any one page. When, following the rules associated with "last detail line", a control group is completed on one page and the next control group is to be started on the next page, a "page" condition exists.

The PAGE FOOTING group is printed following the LAST DETAIL or CONTROL FOOTING group if the PAGE condition exists. TYPE PAGE HEADING report groups, if specified, are produced at the beginning of each page regardless of the condition that prompted the new page. Likewise, the absence of a TYPE OVERFLOW FOOTING entry indicates that TYPE PAGE FOOTING report groups, if specified, are produced at the bottom of each page regardless of the conditions that ended the current page. If no control group exists or if the LAST DETAIL option of PAGE LIMIT is omitted, no overflow can occur and only the PAGE HEADING and/or FOOTING report groups are printed.

Special Counters

Fixed data items LINE-COUNTER and PAGE-COUNTER are generated automatically by the Report Writer for each report. When a program contains more than one report, any reference to either of these items must be qualified by the report-name.

LINE-COUNTER is used by the Report Writer to control spacing information on the page and to determine when a PAGE/OVERFLOW HEADING or FOOTING report group is to be presented. One LINE-COUNTER is supplied for each report. The LINE-COUNTER may be referenced by PROCEDURE DIVISION statements; however, if the LINE-COUNTER is changed by a PROCEDURE DIVISION statement, page format control may be unpredictable. Initially, the LINE-COUNTER is set to zero by the Report Writer. It is automatically tested and incremented during execution according to the PAGE LIMIT clause and the values specified by LINE NUMBER and NEXT GROUP. The LINE-COUNTER is reset to zero when PAGE LIMIT is exceeded during execution. Value of the LINE-COUNTER during execution represents the number of the last line of the previous report group or the number of the last line skipped by NEXT GROUP specification; this value of the LINE-COUNTER is tested from the PROCEDURE DIVISION.

PAGE-COUNTER is a special register automatically generated by the Report Writer for use as a data item to number the pages within a report. One PAGE-COUNTER item is supplied for each report, and may be referenced by PROCEDURE DIVISION statements.

The PAGE-COUNTER is automatically set to one when the Report Writer begins a report. The user may set the PAGE-COUNTER to an initial value other than one with a PROCEDURE DIVISION statement immediately following the INITIATE statement. The Report Writer automatically increments the PAGE-COUNTER by one at each page break (after any PAGE or OVERFLOW FOOTING or before any PAGE or OVERFLOW HEADING).

DATA DIVISION

Entry Formats

Formats used by the Report Writer are specified below in the order of appearance in an XDS ANS COBOL program The report must be named in the FILE SECTION and described in the REPORT SECTION of the DATA DIVISION. Statements that generate a report are specified in the PROCEDURE DIVISION.

FILE SECTION

An output file upon which the Report Writer is to be produced must be defined by a File Description entry in the DATA DIVISION. Refer to "The File Description – Complete Entry Skeleton" under "Physical and Logical Aspects of Data Description" in Chapter 5 for a complete discussion of the format of the File Description entry. The FD entry must name the reports in the REPORT IS clause:

$$\left\{\begin{array}{l}\underline{\text{REPORT}}\text{ IS}\\ \underline{\text{REPORTS}}\text{ ARE}\end{array}\right\}\text{ report-name-1 }\left[\text{, report-name-2}\right]\ldots$$

A report-name is specified in exactly the same way as a data-name. If more than one report-name is included in an FD entry, the file contains more than one report. If separate files are required for each report, a separate FD entry must be defined for each file.

REPORT SECTION

Each report named in an FD entry must be defined in a Report Description entry in the REPORT SECTION, which must be the last section in the DATA DIVISION. The header REPORT SECTION followed by a period must precede any report descriptions. This section specifies the layout of each page. The Report Description entry (level indicator RD) is required, and Report Group Description entries (level-number 01) are required to divide the report into groups.

Report Description Entry

The formats of this entry are

Format 1

```
RD report-name [WITH CODE mnemonic-name] copy-statement.
```

Format 2

```
RD report-name

    [WITH CODE mnemonic-name]

    [ ; {CONTROL IS  } {FINAL                                  } ]
    [   {CONTROLS ARE} {data-name-1 [, data-name-2] ...        } ]
    [                  {FINAL, data-name-1 [, data-name-2] ... } ]

    [ ; PAGE [LIMIT IS  ] integer-1 {LINE } [, HEADING integer-2]               ]
    [        [LIMITS ARE]           {LINES}                                     ]
    [                                                                           ]
    [        [, FIRST DETAIL integer-3] [, LAST DETAIL integer-4] [, FOOTING integer-5] ].
```

Format 1 is used when the RD entry is contained in the COBOL library (subordinate report groups are not copied by a COPY in the RD). For additional information see Chapter 9, "COBOL Library".

In both formats the RD level indicator is required; it starts in Area A and precedes the report-name, which must start in Area B. The unique report-name must be specified here and in a REPORT clause in at least one FD entry in the FILE SECTION. All clauses following report-name in this entry are optional.

CODE Clause. The format of this clause is

```
    [WITH CODE mnemonic-name]
```

The CODE clause is used when more than one report is generated and stored on the same file for subsequent printing. It specifies a unique character that identifies this report. The mnemonic-name must be equated to a non-numeric literal in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. The literal must be a single character. This is a unique identifier so that, following execution, a report selection program can inspect a file and print only the reports required.

CONTROL Clause. The format of this clause is

```
    [ ; {CONTROL IS  } {FINAL                                  } ]
    [   {CONTROLS ARE} {data-name-1 [, data-name-2] ...        } ]
    [                  {FINAL, data-name-1 [, data-name-2] ... } ]
```

The CONTROL clause specifies data-names associated with the control hierarchy within a report; therefore, such data-names must be listed in order from major to minor. FINAL is the highest control, data-name-1 is the major control, data-name-2 is next in order, and the last data-name is the minor control. Data-names must be defined in the FILE, WORKING-STORAGE, or COMMON-STORAGE SECTIONs of the DATA DIVISION.

This clause must be included when the TYPE clause specifies CONTROL FOOTING or CONTROL HEADING. Control footings and headings are printed automatically as a result of control breaks defined in this clause. A control break occurs whenever the value of a data-name specified in this clause changes. The CONTROL clause must also be included when the RESET clause is specified.

PAGE LIMIT Clause. The format of this clause is

$$\left[\; ;\ \underline{\text{PAGE}} \begin{bmatrix}\text{LIMIT IS}\\ \text{LIMITS ARE}\end{bmatrix} \text{integer-1} \begin{Bmatrix}\underline{\text{LINE}}\\ \underline{\text{LINES}}\end{Bmatrix} [,\ \underline{\text{HEADING}}\ \text{integer-2}] \quad [,\ \underline{\text{FIRST}}\ \underline{\text{DETAIL}}\ \text{integer-3}]\ [,\ \underline{\text{LAST}}\ \underline{\text{DETAIL}}\ \text{integer-4}]\ [,\ \underline{\text{FOOTING}}\ \text{integer-5}] \;\right]$$

The PAGE LIMIT clause is required if PAGE HEADING or PAGE FOOTING is specified in the TYPE clause or if LINE NUMBER or NEXT GROUP is specified for an item. The PAGE LIMIT clause may be omitted if no automatic positioning of report groups on the page is desired.

Only one PAGE LIMIT clause may be specified for each RD entry; it gives specific line control for positioning reports on a page. All the integers must be positive nonzero numbers. Integer-1 (LINES) specifies the depth of the report. The depth of the page may or may not be equal to the physical perforated continuous-form often associated in a report with the page length. Integer-2 through integer-5 each must not be greater than integer-1.

HEADING specifies the number of the first line on which the heading can appear; it must be greater than or equal to 1. If HEADING is not specified, the heading starts on line 1.

FIRST DETAIL specifies the number of the first line on which a DETAIL report group can start. If a heading extends beyond the line specified by FIRST DETAIL, the DETAIL group follows the last heading line. FIRST DETAIL must be equal to or greater than HEADING. If FIRST DETAIL is omitted, the detail line may begin on the first line following the heading or, if no heading is specified, it begins on the first line.

LAST DETAIL specifies the number of the last line on which the last line of a DETAIL report group can be printed. LAST DETAIL must not be less than FIRST DETAIL. If LAST DETAIL is not specified, the last line on which detail information can appear is either the last line on which a CONTROL FOOTING group (FOOTING) can appear or, if FOOTING is not specified, the last line of detail information may be the same as the last line of the page (PAGE LIMIT).

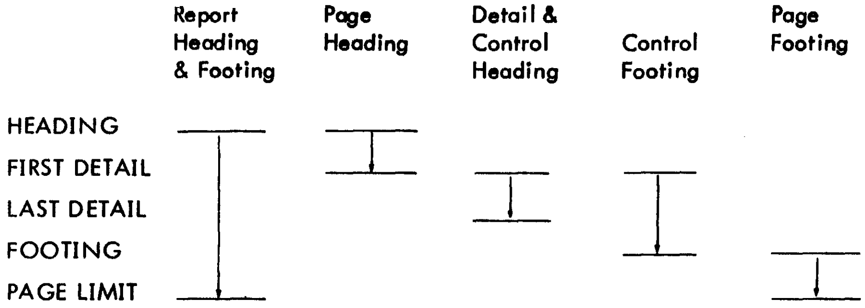
FOOTING specifies the number of the last line on which part of a CONTROL FOOTING group can appear. It must be equal to or greater than FIRST DETAIL. No control footing may begin before the line specified for the first line of detail information (FIRST DETAIL) or extend beyond the line specified by FOOTING. Page and overflow footings may start following the line specified by FOOTING, but they must not start on the same line or extend beyond the last line on the page (PAGE LIMIT).

If absolute line spacing is desired for all groups in the report, and if neither PAGE HEADING nor FOOTING is wanted, only PAGE LIMIT need be specified.

Example:

```
    REPORT SECTION.
    RD     RATIO-REPORT
           PAGE LIMIT IS 55 LINES.
```

The following chart represents the limits of page format when all options of the PAGE LIMIT clause are specified.



Report Group Description Entry

The formats of this entry are

Format 1

01 [data-name-1] copy-statement.

```
Format 2
     01 [data-name-1]

                          ⎧REPORT HEADING                                       ⎫
                          ⎪RH                                                   ⎪
                          ⎪PAGE HEADING                                         ⎪
                          ⎪PH                                                   ⎪
                          ⎪{CONTROL HEADING} {data-name-2}                      ⎪
                          ⎪{CH             } {FINAL      }                      ⎪
             ; TYPE IS   <DETAIL                                                 >
                          ⎪DE                                                   ⎪
                          ⎪{CONTROL FOOTING} {data-name-3}                      ⎪
                          ⎪{CF             } {FINAL      }                      ⎪
                          ⎪PAGE FOOTING                                         ⎪
                          ⎪PF                                                   ⎪
                          ⎪REPORT FOOTING                                       ⎪
                          ⎩RF                                                   ⎭


             ⎡                      ⎧integer-1         ⎫⎤
             ⎢ ; LINE NUMBER IS    <PLUS integer-2     >⎥
             ⎣                      ⎩NEXT PAGE         ⎭⎦

             [;[USAGE IS] DISPLAY]

             ⎡                      ⎧integer-3         ⎫⎤
             ⎢ ; NEXT GROUP IS     <PLUS integer-4     >⎥ .
             ⎣                      ⎩NEXT PAGE         ⎭⎦

Format 3
     nn [data-name-1]

             [; COLUMN NUMBER IS integer-1]

             ⎡                      ⎧integer-2         ⎫⎤
             ⎢ ; LINE NUMBER IS    <PLUS integer-3     >⎥
             ⎣                      ⎩NEXT PAGE         ⎭⎦

             [;[USAGE IS] DISPLAY]

             [; GROUP INDICATE]

             ⎧ ; SOURCE IS [SELECTED] data-name-2                                    ⎫
            <  ; SUM data-name-3 [, data-name-4] . . . [UPON data-name-5]             >
             ⎩ ; VALUE IS literal-1                                                  ⎭

             ⎡ ; RESET ON ⎧data-name-6⎫⎤
             ⎣            ⎩FINAL      ⎭⎦

             ⎡ ; ⎧PICTURE IS⎫ character-string⎤ [; BLANK WHEN ZERO]
             ⎣   ⎩PIC       ⎭                 ⎦

             ⎡ ; ⎧JUSTIFIED⎫ RIGHT ⎤
             ⎣   ⎩JUST     ⎭       ⎦ .
```

The Report Group Description entry defines the characteristics of each report group and of any group or elementary items within the report group.

Format 1 is used only when the entry is an element in the COBOL library. See Chapter 9, "COBOL Library".

Format 2 indicates a report group, which includes all items between this entry and the next level-01 entry. The data-name is optional but must be included if it is referenced from the PROCEDURE DIVISION. The TYPE clause is required for all report groups.

Format 3 indicates an elementary item or group item within a report group. If a report group consists of only one elementary entry, Format 3 may include the TYPE and NEXT GROUP clauses in order to specify the report group and elementary item in the same entry.

The following clauses, which have the same meaning for a Report Group Description entry as for a Record Description entry, are discussed under "Data Description Entries" in Chapter 5.

PICTURE

BLANK WHEN ZERO

JUSTIFIED

VALUE

<u>TYPE Clause.</u> The format of this clause is

$$; \ \underline{\text{TYPE}}\ \text{IS} \left\{ \begin{array}{ll} \underline{\text{REPORT}}\ \underline{\text{HEADING}} & \text{or } \underline{\text{RH}} \\ \underline{\text{PAGE}}\ \underline{\text{HEADING}} & \text{or } \underline{\text{PH}} \\ \underline{\text{CONTROL}}\ \underline{\text{HEADING}} & \text{or } \underline{\text{CH}} \left\{ \begin{array}{l} \text{data-name-2} \\ \underline{\text{FINAL}} \end{array} \right\} \\ \underline{\text{DETAIL}} & \text{or } \underline{\text{DE}} \\ \underline{\text{CONTROL}}\ \underline{\text{FOOTING}} & \text{or } \underline{\text{CF}} \left\{ \begin{array}{l} \text{data-name-3} \\ \underline{\text{FINAL}} \end{array} \right\} \\ \underline{\text{PAGE}}\ \underline{\text{FOOTING}} & \text{or } \underline{\text{PF}} \\ \underline{\text{REPORT}}\ \underline{\text{FOOTING}} & \text{or } \underline{\text{RF}} \end{array} \right\}$$

The TYPE clause specifies the type or the usage of a report group within the format of the report, and is required for every report group. A report consists of a title or heading for the whole report, headings and/or footings for each page, and at least one line or group (repeated as often as required) containing the detail information. Summary information can be printed at control breaks in the form of control headings or footings. Each type of report group must be specified in a TYPE clause.

The INITIATE statement in the PROCEDURE DIVISION does not produce the REPORT HEADING group, if one is specified. The GENERATE statement causes the TYPE DETAIL report group to be produced. GENERATE also produces specified PAGE HEADINGs and FOOTINGs, and the appropriate CONTROL HEADINGs or FOOTINGs when the specified control breaks occur. The TERMINATE statement produces the REPORT FOOTING, if specified, preceded by the CONTROL FOOTING groups from minor to major control level up to CONTROL FOOTING FINAL, if specified.

Types of report groups and their applications are

1. REPORT HEADING/REPORT FOOTING

   Only one REPORT HEADING may be specified for a report; it is printed once at the beginning of the report. Similarly, only one REPORT FOOTING may be specified; it is printed once at the end of the report. If SOURCE is specified in the description of a REPORT HEADING, it refers to the value of the items at the time INITIATE is executed. A SOURCE for a REPORT FOOTING group refers to the value of the items at the time TERMINATE is executed.

2. PAGE HEADING/PAGE FOOTING

   PAGE HEADING is a report group printed at the beginning of a page. PAGE FOOTING is a report group printed at the end of each page following a page condition. A report may have only one PAGE HEADING report group and one PAGE FOOTING report group.

3. CONTROL HEADING/CONTROL FOOTING

   A CONTROL FOOTING report group is printed following the control break specified by data-name-3 or any higher level control break; a CONTROL HEADING report group is printed to start off the next control group. Only one pair of report groups may be specified as a CONTROL HEADING and CONTROL FOOTING for each data-name in the CONTROLS clause.

   CONTROL HEADING FINAL specifies a report group to be printed between the REPORT HEADING and the first CONTROL HEADING group. A CONTROL FOOTING FINAL report group is printed between the end of a report and the REPORT FOOTING; a SOURCE clause in a CONTROL FOOTING FINAL report group refers to the item value at the time the TERMINATE statement is executed.

4. DETAIL

   A DETAIL report group is printed each time a GENERATE statement referring to that group is executed. A DETAIL type report group must have a unique data-name at the 01 level in order to be referenced by GENERATE, which automatically prints any applicable headings and footings.

If all the above report group types are specified, the Report Writer prints them in the following order.

1. REPORT HEADING (once only)

   PAGE HEADING

2. CONTROL HEADING

   DETAIL

   CONTROL FOOTING

3. PAGE FOOTING

4. REPORT FOOTING (once only)

CONTROL HEADING report groups are presented in the following order:

Final Control Heading

Major Control Heading

Minor Control Heading

CONTROL FOOTING report groups are presented in the following order:

Minor Control Footing

Major Control Footing

Final Control Footing

LINE NUMBER Clause. The format of this clause is

$$\left[;\ \underline{\text{LINE}}\ \text{NUMBER IS} \begin{Bmatrix} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{Bmatrix} \right]$$

The LINE NUMBER clause indicates the line number on which the report group is to be printed. It must be specified at or before the first elementary item in the group.

When LINE NUMBER specifies an absolute number, the LINE-COUNTER register is set to the value of integer-1 and all items within the report group are printed on this line until a new value for LINE-COUNTER is specified. If integer-1 is equal to or less than the previously specified value of LINE-COUNTER, the report group is printed on the next page after any PAGE FOOTINGs and HEADINGs.

When LINE NUMBER specifies a relative value, the LINE-COUNTER is incremented by integer-2 for this item and remains the same for subsequent items within the report group until a new LINE NUMBER clause resets the LINE-COUNTER. When LINE NUMBER is specified for an elementary item, all subsequent elementary items appear on the same line until a new LINE NUMBER clause is encountered. LINE NUMBER for an elementary item may not contradict that specified for a group item. Within a report group, entries must be assigned line numbers in ascending order; therefore, an absolute line number (integer-1) cannot be preceded by a relative line number (PLUS integer-2).

NEXT PAGE specifies the item to be printed first on the following page. (PAGE HEADINGs are assumed to be on NEXT PAGE.)

Example:

```
01  TYPE IS DE LINE NEXT PAGE.

    03  LINE 1 ... .

    03  LINE PLUS 1 ... .
```

NEXT GROUP Clause. The format of this clause is

$$\left[;\ \underline{\text{NEXT}}\ \underline{\text{GROUP}}\ \text{IS} \begin{Bmatrix} \text{integer-3} \\ \underline{\text{PLUS}}\ \text{integer-4} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{Bmatrix} \right]$$

Note: Integer-3 and integer-4 must be positive and nonzero.

The NEXT GROUP clause may appear only at the 01 level. It forces spacing between report groups. The LINE-COUNTER is set to the value of integer-3 after the last line of the preceding report group is printed. Note that the NEXT GROUP cannot be printed on this line, since it must specify a LINE NUMBER of at least PLUS 1; this value is a line preceding the line on which the next report group is to be printed. If integer-3 is less than or equal to the previous value of the LINE-COUNTER register, a page change takes place with PAGE FOOTING and PAGE HEADING.

Example:

```
01  DE-TAIL

    TYPE DE

    LINE PLUS 1 ...

    NEXT GROUP PLUS 3 .
```

DE-TAIL is printed on the line following the preceding group, and is followed by three blank lines before the next group is printed if the next report group specifies LINE PLUS 1.

COLUMN NUMBER Clause. The format of this clause is

```
[; COLUMN NUMBER IS integer-1]
```

The COLUMN NUMBER clause indicates the column number of the leftmost character of the space reserved on the printed page for the elementary item. This clause must be included in the description of every elementary item to be presented, unless the elementary item is to be suppressed when the report is produced at object time. Integer-1 must be a number 1 through 132. For a particular report group, COLUMN NUMBER entries must be presented in the order in which the items are to appear on the page, from left to right and top to bottom. When COLUMN NUMBER is specified, the elementary item description must also contain a PICTURE clause and one of the clauses SOURCE, SUM, or VALUE, or a PICTURE clause subordinate to a SOURCE SELECTED clause specified at the group level.

Example:

```
01  DE-TAIL
    TYPE DE LINE PLUS 1.
    03 COLUMN 1 PICTURE X(10)  SOURCE NAME.
    03 COLUMN 13 PICTURE ZZZZ9  SOURCE QUANTITY.
    03 COLUMN 25 PICTURE ZZZZ9  SOURCE LOCATION.
    03 COLUMN 33 PICTURE Z9.99  SOURCE RATE.
    03 COLUMN 44 PICTURE Z(5)9  SOURCE PRICE.
    03 COLUMN 57 PICTURE ZZZ9.99  SOURCE NEW-RATE.
```

GROUP INDICATE Clause. The format of this clause is

```
[; GROUP INDICATE]
```

The GROUP INDICATE clause indicates the elementary item is to be produced only once: at the first occurrence of the report group following a control break, or at the beginning of a new page. This clause must be specified only at the elementary item level within a DETAIL report group.

SOURCE, SUM, VALUE Clauses. The formats of these clauses are

```
{; SOURCE IS [SELECTED] data-name-2                              }
{; SUM data-name-3 [, data-name-4] ... [UPON data-name-5]        }
{; VALUE IS literal-1                                            }
```

The SOURCE, SUM, and VALUE clauses define the purpose and indicate the source of data for an item within the report group; one of these clauses must be included in the description of each elementary item.

Data-name indicates (1) an item appearing in another section of the DATA DIVISION; or (2) the name specified as a SUM counter in the same report; or (3) the LINE-COUNTER or PAGE-COUNTER. The literal specified in the VALUE clause may be non-numeric or a figurative constant, but the PICTURE of an item having a VALUE cannot specify editing.

SOURCE (without the SELECTED options), SUM, and VALUE clauses may appear only at the elementary level; SOURCE IS SELECTED may appear only at the group level. SUM specifies an item to be summed and may appear only upon an item in a CONTROL FOOTING report group. SOURCE, SUM, or VALUE must be specified for every elementary item unless SOURCE IS SELECTED is specified in the entry for the group containing the elementary item.

SOURCE and SUM are described below; VALUE is described in "VALUE Clause" under "DATA DIVISION Structure" in Chapter 5.

1. SOURCE Clause

   The SOURCE clause indicates a data item to be used as the source for the report item. PICTURE must also be specified in the entry for the report item; the value of the SOURCE data item is moved to the report item and edited according to the PICTURE in the item description.

Data-name-2 is an item in the FILE, WORKING-STORAGE, or COMMON-STORAGE SECTION whose value at object time is the effective value to be used in the report item. If LINE-COUNTER is specified, the current value of the line-counter is the source; this value is the number of the last line printed or skipped. If PAGE-COUNTER is named as the source, the current value of the page-counter is used.

When SOURCE IS SELECTED, data-name-2 must be a group item. This option is similar to a PROCEDURE DIVISION MOVE CORRESPONDING statement. The elementary level items within data-name-2 are matched against the data-names specified at the elementary level within the report group. Matching data items are selected as source items to be presented within the report group according to the PICTURE specifications given for the data items in the report group entry.

2. SUM Clause

The SUM clause indicates values to be accumulated until a control break occurs at object time. SUM may appear only in a CONTROL FOOTING report group at the elementary level. An entry containing a SUM clause defines a SUM counter. The data-names specified with SUM indicate items to be summed. These data-names must appear as operands of a SOURCE clause in a DETAIL report group or must appear as the name of a SUM clause at an equal or lower level of the control hierarchy. A PICTURE must be specified for each SUM counter; editing characters or BLANK WHEN ZERO may be specified. Any editing occurs as the contents of the SUM counters are moved to the print line.

The data-names specified in a SUM clause are added to the SUM counter at each execution of a GENERATE statement (unless the UPON option is used or the data-name represents another SUM counter). The UPON option is used for selective summation. Data-name-3, data-name-4, etc., must be SOURCE data items in data-name-5. Data-name-5 must be the name of a DETAIL report group. The values of data-name-3, data-name-4, etc., are added to the SUM counter only when data-name-5 is referenced by a GENERATE statement. When GENERATE is executed for a DETAIL report group. All summing takes place automatically as follows:

a. The CONTROL data-names specified in the CONTROL FOOTING report group are tested. If the data-names are unchanged, each SOURCE item in the DETAIL report group is added to each SUM counter in any CONTROL FOOTING report group that names the SOURCE item. Then the DETAIL line is produced.

b. If a data-name changes, a control break has occurred. Each SUM counter in the lowest level report group is added to each SUM counter in the same report group that names the SUM counter as an operand. Then the CONTROL FOOTING report group is produced, and the SUM counters are reset to zero.

c. If RESET is specified, adding the SUM counter into higher level SUM counters and resetting to zeros are postponed until the CONTROL FOOTING for the level indicated by the RESET is produced. However, the current contents of the SUM counter can be printed. This feature allows printing of cumulative totals.

The last two steps are repeated as necessary for each level CONTROL FOOTING until the level of the original control break is reached.

<u>RESET Clause.</u> The format of this clause is

$$\left[;\ \underline{\text{RESET}}\ \text{ON} \begin{Bmatrix} \text{data-name-6} \\ \underline{\text{FINAL}} \end{Bmatrix} \right]$$

The RESET clause may be used in an elementary item in a CONTROL FOOTING report group to override the automatic resetting of the SUM counter following the associated control break. It can be used only at an elementary item level in conjunction with the SUM clause. Data-name-6 must be one of the data-names described in the CONTROL clause in the RD entry for the report, and it must be a higher level data-name than the CONTROL data-name associated with the CONTROL FOOTING report group containing the SUM and RESET clauses.

When RESET is not specified, the SUM counters are automatically reset after presentation of the CONTROL FOOTING report group to which the SUM item is subordinate. RESET prevents automatic resetting of the SUM counters until the CONTROL FOOTING report group associated with data-name-6 is presented. This clause permits the progressive totaling of data while presenting subtotals at lower levels of the control hierarchy.

Example:

```
RD  RREPORTT
    CONTROLS ARE FINAL, DEPT, SECTN, GRUP, MAN.
    .
    .
    .
    01  GRUP-TOTALS TYPE CF GRUP
        LINE PLUS 2.
        02  COLUMN 35 PICTURE 9(12)
            SUM GRUP-HRS RESET ON SECTN .
```

GRUP-HRS are summed for this CONTROL FOOTING group and the subtotal continues to accumulate until the CONTROL FOOTING group presented at the SECTN level of the control hierarchy is produced.

PROCEDURE DIVISION

To produce a report defined in the REPORT SECTION, three PROCEDURE DIVISION statements are required: INITIATE, GENERATE, and TERMINATE. A USE BEFORE REPORTING statement may introduce a DECLARATIVES section through which the user may manipulate, alter, or inspect data immediately before it is printed.

INITIATE Statement

The format of this statement is

$$\underline{\text{INITIATE}}\left\{\begin{array}{l}\text{report-name-1 [, report-name-2] . . .}\\ \underline{\text{ALL}}\end{array}\right\}$$

The INITIATE statement initializes all counters and controls prior to producing a report, and begins the processing of a report. Report-names are the reports to be initiated. Each name must be defined by a Report Description (RD) entry in the REPORT SECTION. ALL specifies that all report-names defined by RD entries in the REPORT SECTION of this program are to be initiated.

A second INITIATE statement for a particular report-name may not be executed unless a TERMINATE statement is executed for that report-name subsequent to the first INITIATE statement. The INITIATE statement does not open the file with which the report is associated; an OPEN statement for the file must be executed prior to the INITIATE. The INITIATE statement does nothing if the report file is not open.

INITIATE performs Report Writer functions for individually described reports analogous to the input/output functions that OPEN performs for individually described files:

1. Sets all SUM counters to zero.
2. Sets up the initial values to be used for comparison in the control hierarchy.
3. Sets the PAGE-COUNTER; if some other initial value is desired, the user may reset this counter following execution of INITIATE.
4. Sets the LINE-COUNTER to zero.

GENERATE Statement

The format of this statement is

$\underline{\text{GENERATE}}$ data-name-1

The GENERATE statement links the PROCEDURE DIVISION to the Report Writer at object time: it presents a DETAIL report group (under PROCEDURE DIVISION control) to the Report Writer. If data-name-1 names a DETAIL report group, GENERATE handles all relevant automatic operations and produces a DETAIL report. This is called detail reporting.

If data-name-1 names a report-name, GENERATE handles all the relevant automatic operations and updates the FOOTING report groups in the report without producing any DETAIL report groups. This is called summary reporting.

If the report includes more than one DETAIL report group, all SUM counters except SUM UPON counters are incremented each time GENERATE is executed.

A GENERATE statement, implicit in both detail and summary reporting, produces the following automatic operations as needed:

1. Recognizes any specified control breaks to produce CONTROL HEADING and CONTROL FOOTING report groups.
2. Accumulates all specified data-names into the SUM counters. Resets the SUM counters at control breaks. Performs a "rolling forward" of each set of SUM counters between control break levels.
3. Executes any specified routines defined by USE before generation of report groups.

During execution of the first GENERATE statement, all CONTROL HEADING report groups specified for the report are produced in order: FINAL and major through minor, immediately followed by any DETAIL report group specified in the statement. If a control break is recognized when a GENERATE statement is executed, all CONTROL FOOTING report groups specified for the report are produced from the minor report group up to and including the report group specified for the data-name that caused the control break. The CONTROL HEADING report groups specified for the report, from the report group specified for the data-name that caused the control breakdown to the minor report group, are then produced in that order. The DETAIL report group specified in the GENERATE statement is produced last. When data is moved to a report group, it is edited according to the rules described in "MOVE Statement" under "Data Manipulation Statements" in Chapter 6.

TERMINATE Statement

The format of this statement is

$$\underline{\text{TERMINATE}}\left\{\begin{array}{l}\text{report-name-1 [, report-name-2]...}\\ \underline{\text{ALL}}\end{array}\right\}$$

The TERMINATE statement ends processing of a report. Report-names are the reports to be terminated. ALL specifies that all report names defined in the program are to be terminated.

TERMINATE produces all the CONTROL FOOTING report groups associated with the report as if a control break has just occurred at the highest level (FINAL), and completes the Report Writer functions for the named reports. If SOURCE clauses are included in the FINAL CONTROL FOOTING or REPORT FOOTING groups, values for the SOURCE data-names are the values of the data items upon execution of the TERMINATE statement.

A second TERMINATE for a particular report may not be executed unless an intervening INITIATE is executed for that report. TERMINATE does not close the file with which the report is associated: the CLOSE statement for the file must be specified by the user.

USE BEFORE REPORTING Statement

The format of this statement is

$\underline{\text{USE}}$ $\underline{\text{BEFORE}}$ $\underline{\text{REPORTING}}$ data-name .

The USE BEFORE REPORTING statement may follow a section header in the DECLARATIVES section. It introduces procedures to be performed immediately before the specified report groups are produced.

The data-name may be of any type report group (01 level) except DETAIL, and must not appear in more than one USE BEFORE REPORTING statement. The Report Writer verbs GENERATE, INITIATE, and TERMINATE may not be used in any procedure introduced by USE BEFORE REPORTING.

The USE BEFORE REPORTING implies a PERFORM of the section containing the USE statement for each report group specified by the data-names. This section is executed immediately before the report group is produced, i.e., after any summing but before data is moved into the line image and before the LINE-COUNTER is incremented.

8. SORT FEATURE

General

The Sort feature is comprised of the SORT verb in the PROCEDURE DIVISION and descriptions of the records to be sorted provided by the Sort-File Description in the DATA DIVISION. The Sort orders records based on sort keys given in the SORT statement, each key being a data item defined in a record description associated with the Sort-File to which the SORT statement is directed. Records may be arranged by the Sort in ascending or descending sequence or in a combination of ascending and descending sequences, since the individual sort keys may be specified as ascending or descending quite independently of each other; the final order reflects the combination indicated.

Sort Feature Components

In order to employ the Sort capability, certain information must be provided in all of the active divisions of the source programs.

The ENVIRONMENT DIVISION must contain SELECT sentences for the Sort-File itself and for the files that act as input and output for the Sort when USING or GIVING options are written (see "SORT Statement" below).

The DATA DIVISION must contain file descriptions of the Sort-File and of all files that contain input to and output from the Sort.

The PROCEDURE DIVISION contains the SORT statement itself and, when the programmer indicates that special input or output processing is to be performed, the PROCEDURE DIVISION must contain such procedures.

ENVIRONMENT DIVISION Statements

The format of this statement is

$$\underline{\text{SELECT}}\ \text{sort-file}\ \underline{\text{ASSIGN}}\ \text{TO}\ [\text{integer-1}]\text{implementor-name-1}[,\ \text{implementor-name-2}]\ldots$$

The FILE-CONTROL paragraph must include SELECT sentences for the Sort-File and for files named in the USING or GIVING options, when specified. Because the Sort-File is not a true file, the ASSIGN portion of the SELECT sentence need not be written unless restriction of Sort work file assignments is desired (see ASSIGN Clause, Chapter 4).

DATA DIVISION Statements

A Sort-File Description entry and related Record Description entries must be supplied for each Sort-File in the source program together with the normal File Description entries for the input and output files.

Sort-File Description – Complete Entry Skeleton

The formats of this entry are

Format 1

$$\underline{\text{SD}}\ \text{file-name copy-statement.}$$

Format 2

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

$$\left[;\ \underline{\text{RECORD}}\ \text{CONTAINS}[\text{integer-1}\ \underline{\text{TO}}]\text{integer-2 CHARACTERS}\right].$$

The Sort-File description furnishes information concerning the physical structure, identification, and record-names of the file to be sorted, unless the USING and/or GIVING options of the sort are employed. When the USING and/or GIVING options are specified, the record sizes and descriptions of the file being sorted are taken from the record descriptions of the files named in the USING/GIVING clauses.

Record Descriptions

Sort record descriptions do not differ in any way from other record descriptions.

Sort Keys

Sort keys must have a fixed length, must not be subscripted, and must be one of the types of data items listed below. Corresponding to each data item type are its maximum length and a collating sequence used for sorting.

Data Item	Collating Sequence	Maximum Length (Bytes)
Computational or Index (Binary)	Algebraic	4
Computational-1 (Floating-Point)	Algebraic	4
Computational-2 (Floating-Point)	Algebraic	8
Computational-3 (Packed Decimal)	Algebraic	16
Numeric Display (Zoned Decimal)	Algebraic	31
Alphabetic	EBCDIC	255
Alphanumeric	EBCDIC	255
Group	EBCDIC	255

Records to be sorted may contain 1 to 16 key fields with maximum length as indicated above.

When assigned to sort keys, data-names may be qualified but must be unique. Also, once a data-name is assigned to a sort key, it is assumed that the key appears in the same location in every record to be sorted. Data-name sort keys may not contain or be subordinate to entries that contain an OCCURS clause.

PROCEDURE DIVISION Statements

A SORT statement is required for each sorting operation. In addition, INPUT or OUTPUT PROCEDUREs must be provided to specify any processing to be executed before or after the Sort.

SORT Statement

The format of this statement is

$$\underline{\text{SORT}}\ \text{file-name-1 ON}\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\text{KEY data-name-1}\ldots\left[;\ \text{ON}\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\text{KEY data-name-2}\ldots\right]\ldots$$

$$\left\{\begin{array}{l}\underline{\text{INPUT}}\ \underline{\text{PROCEDURE}}\ \text{IS section-name-1}\ \left[\underline{\text{THRU}}\ \text{section-name-2}\right]\\ \underline{\text{USING}}\ \text{file-name-2}\end{array}\right\}$$

$$\left\{\begin{array}{l}\underline{\text{OUTPUT}}\ \underline{\text{PROCEDURE}}\ \text{IS section-name-3}\ \left[\underline{\text{THRU}}\ \text{section-name-4}\right]\\ \underline{\text{GIVING}}\ \text{file-name-3}\end{array}\right\}$$

The SORT statement yields information that controls the Sort feature. This information (1) obtains records for sorting from either an INPUT PROCEDURE or an existing file; (2) sorts the records on a set of specified keys; and (3) either makes individual records available in sorted order to an OUTPUT PROCEDURE or creates an ordered output file.

The SORT statement provides

1. The appropriate Sort-File.

2. Names of any INPUT or OUTPUT PROCEDUREs to be invoked.

3. Names of the sort keys and, for each key, indication of ascending or descending sequence.

4. Names of the input or output files.

File-name-1 is the name given in the Sort-File Description entry describing the records to be ordered. ASCENDING and DESCENDING indicate the order in which the records are to be sorted, based on one or more keys; at least one KEY clause must be specified, but both types may be written in the same statement. The data-names are the names of the sort keys. Keys must be specified in the logical order of comparison during the sorting operation.

USING implies that file-name-2 contains all of the records to be sorted and that control of this file passes directly to the Sort upon execution of the SORT statement. This file is opened, read, and closed automatically by the Sort program. Sequential access should be specified for this file.

GIVING names a file into which Sort deposits sorted records; Sort automatically opens, writes into, and closes this file. Consecutive organization and sequential access must be specified for file-name-3.

INPUT PROCEDURE Option

The INPUT PROCEDURE option names a programmer-supplied input procedure in the form of one or more sections that process the input records and pass them to the Sort program. The INPUT PROCEDURE contains procedural statements required to select, create, or modify records. Control should not be passed to the procedure except during execution of the SORT statement, as RELEASE statements are not meaningful unless a SORT is in process. An INPUT PROCEDURE may not contain a SORT statement, since SORTs cannot be nested. The INPUT PROCEDURE accomplishes the following functions:

1. It builds the records to be sorted singly in the record area assigned to the Sort-File. If the input originates from an existing file, an OPEN statement must be executed prior to execution of the SORT statement or within the INPUT PROCEDURE.
2. It submits a processed record to the Sort by execution of a RELEASE statement (see below), after which the record is no longer available.
3. After the last record has been released to the Sort, control is transferred to the Sort process by execution of the final statement in the INPUT PROCEDURE.

RELEASE Statement. The format of this statement is

RELEASE record-name [FROM identifier]

The RELEASE statement transfers one logical record (record-name) belonging to the Sort-File to the sorting operation. It is meaningless outside of an INPUT PROCEDURE; conversely, each INPUT PROCEDURE must contain at least one RELEASE statement.

If the FROM option is used, the contents of the identifier data area are moved to record-name, and then the contents of record-name are released to the Sort-File. Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option.

OUTPUT PROCEDURE Option

The OUTPUT PROCEDURE option names a programmer-supplied output procedure composed of one or more distinct sections for processing sorted records. The OUTPUT PROCEDURE contains the procedural statements required to select, modify, or copy the records that are being returned singly, in order, from the Sort. Control should not be passed to an OUTPUT PROCEDURE except in the course of the sorting operation to which it applies. An OUTPUT PROCEDURE may not contain a SORT statement. The OUTPUT PROCEDURE accomplishes the following functions:

1. It acquires sorted records singly from the Sort program through execution of a RETURN statement (see below).
2. It processes the record just returned by referring to the Sort-File record area. If records are to be transferred to an output file, an OPEN statement addressed to that file must be executed prior to execution of the SORT statement itself or within the OUTPUT PROCEDURE.
3. When all sorted records are obtained, it executes the AT END statement of the RETURN verb. In order to terminate the OUTPUT PROCEDURE and thereby resume the program after the SORT statement, control must be directed through the end point of the OUTPUT PROCEDURE.

RETURN Statement. The format of this statement is

RETURN sort-file RECORD [INTO identifier] ; AT END imperative-statement

The RETURN obtains sorted records from the final phase of a Sort operation. The INTO option causes the record just obtained also to be moved into the area indicated by the identifier. Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option. (Refer to "MOVE Statement" under "Data Manipulation Statements" in Chapter 6.)

9. 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 Monitor sense rather than in COBOL terms) residing in the user complex. A library element is incorporated into a source program by the compiler in response to a COPY statement.

The COPY statement causes a search within the user library 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

$$\underline{\text{COPY}}\ \text{library-name}\ \left[\underline{\text{REPLACING}}\ \text{word-1}\ \underline{\text{BY}} \left\{\begin{matrix}\text{word-2}\\ \text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \left[,\ \text{word-3}\ \underline{\text{BY}} \left\{\begin{matrix}\text{word-4}\\ \text{identifier-2}\\ \text{literal-2}\end{matrix}\right\}\right] \ldots\right]$$

A 'word' in this format may represent one of the following and must conform to the definition of words: data-name, procedure-name, condition-name, mnemonic-name, or file-name.

If the REPLACING phrase is used, each occurrence of word-1, word-3, etc., in the text being copied from the library is replaced by the word, identifier, or literal associated with it in the REPLACING phrase. Use of the REPLACING option does not alter the material as it appears on the library.

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.

   SPECIAL-NAMES. Copy-statement.

   FILE-CONTROL. Copy-statement.

   I-O-CONTROL. Copy-statement.

2. In the FILE SECTION

   FD file-name copy-statement.

   SD sort-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 REPORT SECTION

   RD report-name copy-statement.

   01 data-name copy-statement.

5. In the PROCEDURE DIVISION

$$\left\{\begin{array}{l}\text{paragraph-name.}\\ \text{section-name }\underline{\text{SECTION}}\,[\text{priority-number}].\end{array}\right\}\ \text{copy-statement.}$$

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, and any conflicting information occurring in the copied sequence is discarded.

In the Procedure Division (case 5 above), the verb INCLUDE may be used in place of the verb COPY. 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 COPY statement automatically replacing the procedure-name of the routine. The name of the section (in the source program) containing the routine becomes the only qualifier for the procedure-name(s) within that routine. Procedure-name references in the routine must be unique with respect to the section containing 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 and priority of the COPY section automatically replacing the section-name and priority of the section being copied from the library. Also, all references to the section-name of the routine from within the routine are automatically replaced by references to the section-name associated with the COPY statement. Procedure-name references in the routine need be unique only with respect to the routine containing them.

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 REPLACING COUNT BY G-COUNT.

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

```
01  SUMMARY-A.
    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 G-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.
```

10. INTER-PROGRAM COMMUNICATION

The Inter-Program Communication module provides a facility by which a program can communicate with one or more programs. This communication is provided by the ability to transfer control from one program to another within a run unit and let both programs have access to the same data items.

LINKAGE SECTION

The LINKAGE SECTION in a program is meaningful only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called programs. No space is allocated in the program for data items referenced by data-name in the LINKAGE SECTION of that program. PROCEDURE DIVISION references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. In the case of index-names, no such correspondence is established. Index-names in the called and calling programs always refer to separate indexes.

Data items defined in the LINKAGE SECTION of the called program may be referenced within the PROCEDURE DIVISION of the called program only if they are specified as operands of the USING phrase of the PROCEDURE DIVISION header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the LINKAGE SECTION is the same as that previously described for the WORKING-STORAGE SECTION, beginning with a section header, followed by Data Description entries for noncontiguous data items, Record Description entries, or both.

Each LINKAGE SECTION record-name and noncontiguous item name must be unique within the called program since it cannot be qualified. Data items defined in the LINKAGE SECTION of the called program must not be associated with data items defined in the REPORT SECTION of the calling program.

Of those items defined in the LINKAGE SECTION, only identifier-1, identifier-2, data items subordinate to these identifiers, and condition-names or index-names associated with such identifiers or subordinate data items may be referenced in the PROCEDURE DIVISION.

The name used in a PROGRAM-ID is output as a DEF for programs that use the LINKAGE SECTION. The name is prefixed by an "L:". (Example: for "PROGRAM-ID. FCLOSE. ", COBOL generates a DEF L:FCLOSE.)

Noncontiguous Linkage Storage

Items in the LINKAGE SECTION that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementrary items. Each of these data items is defined in a separate Data Description entry which begins with the special level-number 77.

The following data clauses are required in each Data Description entry:

- Level-number 77
- Data-name
- The PICTURE clause

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

Linkage Records

Data elements in the LINKAGE SECTION that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause used in an input or output record description can be used in a LINKAGE SECTION.

Initial Values

The VALUE clause must not be specified in the LINKAGE SECTION except in condition-name entries (level 88).

PROCEDURE DIVISION Header

The PROCEDURE DIVISION is identified by and must begin with the header

$$\underline{\text{PROCEDURE DIVISION}}\ [\underline{\text{USING}}\ \text{identifier-1}\,[\text{,identifier-2}]\ .\ .\ .].$$

The USING phrase is present if, and only if, the object program is to function under the control of a CALL statement and the CALL statement in the calling program contains a USING phrase.

Each of the operands in the USING phrase of the PROCEDURE DIVISION header must be defined as a data item in the LINKAGE SECTION of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, LINKAGE SECTION data items are processed according to their data descriptions given in the called program.

When the USING phrase is present, the object program operates as if identifier-1 of the PROCEDURE DIVISION header in the called program and identifier-2 in the USING phrase of the CALL statement in the calling program refer to a single set of data that is equally available to both the called and calling programs. Their descriptions must define an equal number of character positions; however, identifier-1 and identifier-2 need not be the same name. In like manner, there is an equivalent relationship between identifier-2, . . ., in the USING phrase of the called program and identifier-3, . . ., in the USING phrase of the CALL statement in the calling program. An identifier must not appear more than once in the USING phrase in the PROCEDURE DIVISION header of the called program; however, a given identifier may appear more than once in the same USING phrase of a CALL statement.

CALL Statement

The form of this statement is

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

The CALL statement causes control to be transferred from one object program to another within the run unit.

Identifier-1 must be defined as an alphanumeric data item whose value can be a program-name. Literal-1 must be a non-numeric literal.

The USING phrase is included in the CALL statement only if there is a USING phrase in the PROCEDURE DIVISION header of the called program. The number of operands in each USING phrase must be identical, and each of the operands must have been defined as a data item in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION, and have a level-number of 01 or 77.

The program whose name is specified by the value of literal-1 or identifier-1 is the called program; the program in which the CALL statement appears is the calling program. Execution of a CALL statement causes control to pass to the called program, which is in its initial state the first time it is called within a run unit. On all other entries, the state of the called program remains unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

The identifiers specified by the USING phrase of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the identifiers in the USING phrase to the CALL statement and the USING phrase in the PROCEDURE DIVISION header is critical. Corresponding identifiers refer to a single set of data available to the called and calling programs. The correspondence is positional, not by name. In the case of index names, no such correspondence is established. Index names in the called programs always refer to separate indexes.

The CALL statement may appear anywhere within a segmented program. Therefore, when a CALL statement appears in a section with a segment number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

EXIT PROGRAM Statement

The form of this statement is

EXIT PROGRAM

The statement marks the logical end of a called program. It must appear in a sentence by itself and must be the only sentence in the paragraph.

Execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program that is not called behaves as if the statement were an EXIT statement (see "EXIT Statement"). Examples of a calling and a called program are shown in Figure 5 below.

Example of a Calling Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LK-IF-MOVE.
AUTHOR. XEROX CORPORATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. XEROX-560.
OBJECT-COMPUTER. XEROX-560.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01  R1.
    02 RG.
       03 RG1                PIC X(8).
       03 RG2                PIC X(8).
       03 RG3                PIC X(7).
    02 RN.
       03 RN1  COMP.
       03 RN2  COMP-1.
       03 RN3  COMP-2.
       03 RN4  COMP-3        PIC S9(3).
       03 RN5                PIC 9(4).
       03 RN6                PIC XBXXBBXX.
       03 RN7                PIC $$99.99.
PROCEDURE DIVISION.
START.
     CALL 'LD-IF-MOVE' USING R1.
     STOP RUN.
```

Example of a Called Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LD-IF-MOVE.
AUTHOR. XEROX CORPORATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. XEROX-SIGMA-7.
OBJECT-COMPUTER. XEROX-SIGMA-7.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
LINKAGE SECTION.
01  L1.
    02 LG.
       03 LG1                      PIC X(8).
       03 LG2                      PIC X(8).
       03 LG3                      PIC X(7).
    02 LN.
       03 LN1  COMP.
       03 LN2  COMP-1.
       03 LN3  COMP-2.
       03 LN4  COMP-3              PIC S9(3).
       03 LN5                      PIC 9(4).
       03 LN6                      PIC XBXXBBXX.
       03 LN7                      PIC $$99.99.
PROCEDURE DIVISION USING L1.
START.
     MOVE 789.56 TO LN7.
     DISPLAY LN7 UPON PRINTER.
LNK-EXIT.
     EXIT PROGRAM.
```

Figure 5. Calling and Called Programs

Subcompile Feature

The Subcompile feature is not part of the ANS COBOL language, but is a language extension implemented in Xerox ANS COBOL only. The Subcompile feature enables a single, logical problem solution expressed in Xerox ANS COBOL to be subdivided into two or more separate source programs that can be compiled independently and subsequently combined into a single executable program. The advantages of smaller, more manageable compilations are obvious: for example, maintenance is simplified and the distribution of problem-solving effort among a number of programmers is facilitated. The Subcompile feature yields these benefits with virtually no impairment of operating efficiency and only modest increase in total compilation time.

Any given COBOL source program may be subdivided into two or more parts, each of which can be compiled independently. One of these subdivisions must be designated as the main program at both compilation and execution times. The remaining subdivisions are called subprograms. Each subdivision of the total program, whether the main program or a subprogram, has the format of a complete COBOL source program. Each subdivision must contain IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE DIVISIONs.

Rules for Usage

Successful usage of the Subcompile feature requires observance of two alternative sets of rules. The first set is somewhat restrictive, but requires a minimal knowledge of the contents of the main program and its s ıbprograms and thus is less susceptible to programmer error.

1. The ENVIRONMENT DIVISIONs must all be complete with regard to the total program, and should be identical.

2. The FILE SECTIONs, COMMON-STORAGE SECTIONs and REPORT SECTIONs must all be complete with regard to the total program, and should be identical.

The second set of rules requires a careful and detailed analysis of the individual source programs but permits omission of repetitious entries, thus reducing the size of the programs and improving compilation time.

1. ENVIRONMENT DIVISION

   a. Main Program

      The complete ENVIRONMENT DIVISION for the total program must be written.

   b. Subprograms

      Each subprogram must contain SELECT sentences only for those files referenced in its PROCEDURE DIVISION (and described in its DATA DIVISION).

2. DATA DIVISION

   a. FILE SECTION

      (1) Main Program

         The file and record descriptions for all files in the total program must be included.

      (2) Subprograms

         The file and record descriptions for all files referenced in the PROCEDURE DIVISION (and mentioned in an ENVIRONMENT DIVISION SELECT sentence) must be included.

   b. COMMON-STORAGE SECTION

      The COMMON-STORAGE SECTIONs must be identical within all elements composing the total program.

   c. REPORT SECTION

      (1) Main Program

         The Report Descriptions of all reports used in the total program must appear.

      (2) Subprograms

         Each subprogram must contain only the descriptions of reports actually referenced therein. (The file description of the file containing the associated REPORT IS clause must also be present.)

3. PROCEDURE DIVISION

   The main program must contain all DECLARATIVES sections desired in the total program.

Memory space is allocated and Data Control Blocks generated for the files described in the FILE SECTION of the main program. All subprograms making reference to reports or report data, when incorporated into the total program at run-time, refer to the areas reserved by the main program. Similarly, memory space is assigned in accordance with the COMMON-STORAGE SECTION description in the main program, and this area is shared by the main program and all associated subprograms when combined at run-time. The main program and each subprogram may have its own WORKING-STORAGE SECTION; data described therein is not shared, but is private to the program in which it is defined. However, WORKING-STORAGE items in a main program may be referred to in a subprogram by the use of a LINKAGE SECTION in the subprogram and the PROCEDURE DIVISION USING statement.

The Subcompile feature enables program control to flow naturally between independent compilations employing the normal COBOL verbs GO TO and PERFORM. Only one additional statement is introduced into the Xerox ANS COBOL language to provide this natural flow. Any procedure point to which control may be passed by a separately compiled program must be declared as an external definition. The ENTER COBOL statement names those entry points (section- and paragraph-names) within the program that are to be visible to sequence control statements in other compilations.

ENTER Statement

The formats of this statement are

Format 1

ENTER language-name routine-name.

Format 2

ENTER [language-name] routine-name [, parameter-name] . . .

Format 3

ENTER COBOL procedure-name-1 [, procedure-name-2]. . .

The ENTER statement (Formats 1 and 2) allows entry into a closed, machine-language routine. The acceptable language-name is METASYM. Routine-name is an entry point in the routine to be entered. Parameter-name may be a data-name, a procedure-name, or a file-name. These parameter-names are supplied to the routine in the COBOL-generated calling sequence by listing the core memory location of these items. Data-names may not be subscripted.

The ENTER COBOL statement (Format 3) makes the locations of the named procedures available to the loader as entry points. Procedure-names are paragraph-names or section-names to which control may pass.

Only the ENTER statement can be used to call Extended Data Management System (EDMS) service routines. The CALL statement cannot be used to perform this function (see the EDMS Reference Manual, 90 30 12).

While both the LINKAGE SECTION and the COMMON-STORAGE SECTION can appear in the same program, they can only be referred to by their proper associated statements (PROCEDURE DIVISION USING statements in the case of the LINKAGE SECTION, or the ENTER statement in the case of COMMON-STORAGE). An ENTER statement cannot refer to items in a LINKAGE SECTION and, conversely, a CALL statement or PROCEDURE DIVISION USING statement cannot refer to data items in COMMON-STORAGE.

11. PRIORITY SEGMENTATION

Segmentation is a facility that provides a means of communicating object program overlay requirements to the compiler.

Although it is not mandatory, the PROCEDURE DIVISION of a source program is usually written as a succession of sections, each of which embraces a series of closely related operations involved in accomplishing a particular task. When all sections of the program are defined it is possible to classify them in accordance with their importance, frequency of use, and interrelationships. Each section is classified as belonging to either the fixed portion, which is always in memory during execution, or to one of the independent overlayable segments of the object program. An overlayable segment is a program segment that, although logically treated as permanent in memory, is capable of being overlaid by another such segment to optimize memory utilization. Only one priority segment may be in memory at a time; the amount of object space reserved for the nonfixed portion of the object program is the size of the largest priority segment.

Sections are classified by priority-numbers included in the section header

section-name <u>SECTION</u> [priority-number] .

and must be an integer ranging in value from 0 through 99. If the priority-number is omitted a value of zero is assumed.

All sections of the same priority-number are grouped together to form a program segment with that priority. Normally, segments with priority-number 0 through 49 belong to the fixed portion and segments with priority-number 50 through 99 are overlayable. This arbitrary separation can be adjusted by employing the SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION to enable more overlayable segments to be created. When the SEGMENT-LIMIT clause is used, those segments having priority-numbers ranging from the specified value through 99 are produced as overlayable segments.

Segmentation does not alter the logical structure of the program in any way. The logical sequence is the same as the physical sequence except for specific transfers of control; this pertains in spite of any effective reordering of the program occasioned by the grouping of procedure sections (possibly scattered throughout the source program) by priority-number to form a segment. Segmentation in no way affects the need for qualification of procedure names to ensure uniqueness.

The following points should be remembered when segmentation of a COBOL program is considered:

1. Determination of the need for segmentation is a programmer responsibility. Indiscriminate use of segmentation may degrade object program efficiency severely, and the compiler has no means of controlling the use – or pointing out possible misuse – of segmentation.

2. Sections required for reference at all times or referred to frequently should be allocated to the fixed portion.

3. Sections used less frequently (e.g., initialization or closeout routines) or executed in a chronological series are suitable candidates for segmentation.

4. Sections that communicate frequently with each other should be assigned to the same segment.

12. DEBUGGING FACILITIES

The Debug module provides a means by which the user can describe his debugging algorithm including the conditions under which data items or procedures are to be monitored during the execution of the object program.

The features of the COBOL language that support the Debug module are

- A compile time switch – WITH DEBUGGING MODE.
- An object time switch.
- A USE FOR DEBUGGING statement.
- A special register – DEBUG-ITEM.
- Debugging lines.

DEBUG-ITEM

The reserved word DEBUG-ITEM is the name for a special register generated automatically by the compiler code that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

Compile Time Switch

The WITH DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph. It serves as a compile time switch over the debugging statements written in the program.

When the WITH DEBUGGING MODE clause is specified in a program, all debugging sections and all debugging lines are compiled as specified in this section. When the WITH DEBUGGING MODE clause is not specified, all debugging lines and all debugging sections are compiled as if they were comment lines.

Object Time Switch

An object time switch dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch is on, all the effects of the debugging language written in the source program are permitted. If the switch is off, all the effects described under USE FOR DEBUGGING statement are inhibited. Recompilation of the source program is not required to provide or take away this facility.

The object time switch has no effect on debugging lines nor on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time.

WITH DEBUGGING MODE Clause

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

<u>SOURCE-COMPUTER.</u> computer-name [WITH <u>DEBUGGING MODE</u>].

If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the CONFIGURATION SECTION of a program, all USE FOR DEBUGGING statements and debugging lines are compiled.

If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the CONFIGURATION SECTION of a program, any USE FOR DEBUGGING statements and all associated debugging sections, and any debugging lines are compiled as if they were comment lines.

USE FOR DEBUGGING Statement

The form of this statement is

section-name SECTION.

```
                             ⎧[ALL REFERENCES OF] identifier-1⎫ ⎡[ALL REFERENCES OF] identifier-2⎤
USE FOR DEBUGGING ON ⎨file-name-1                      ⎬ ⎢file-name-2                      ⎥
                             ⎪PROCEDURE procedure-name-1       ⎪ ⎢PROCEDURE procedure-name-2       ⎥
                             ⎩ALL PROCEDURES                   ⎭ ⎣ALL PROCEDURES                   ⎦
```

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

All references to identifier-1, procedure-name-1, and file-name-1 apply equally to identifier-2, procedure-name-2, and file-name-2, respectively.

Debugging section(s), if specified, must appear immediately after the DECLARATIVES header and except in the USE FOR DEBUGGING statement itself, there must be no reference to any nondeclarative procedure within the debugging section.

Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections. Statements appearing within a given debugging section, except for the USE FOR DEBUGGING statement itself, may reference procedure-names defined within a different USE procedure only with a PERFORM statement. Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

Any identifier, file-name, or procedure-name can appear in only one USE FOR DEBUGGING statement, and the ALL PROCEDURES phrase can appear only once in a program. When the ALL PROCEDURES phrase is specified, procedure-name-1 must not be specified in any USE FOR DEBUGGING statement.

Identifier-1 must not be a data item defined in the REPORT SECTION except sum counters. If the data description of identifier-1 contains an OCCURS clause or is subordinate to a data item that contains an OCCURS clause, identifier-1 must be specified without the subscripting or indexing normally required.

References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

Execution of a debugging section depends on the USE FOR DEBUGGING specification.

When file-name-1 is specified, execution takes place after one of the following:

1. The execution of any OPEN or CLOSE statement that references file-name-1.

2. After the execution of any READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement.

When a procedure-name-1 is specified, execution takes place

1. Immediately before a GO TO, PERFORM, or SORT statement that transfers control to procedure-name-1.

2. Immediately before procedure-name-1 is executed because of transfer of control implicit from the previous paragraph.

3. Immediately after the execution of an ALTER statement that alters procedure-name-1.

4. Immediately before a USE procedure (procedure-name-1) is executed.

5. Immediately before the named procedure is executed for the first time if it is the first procedure in the non-declarative portion of the program.

The ALL PROCEDURES phrase causes these effects to occur for every procedure-name in the program, except those appearing within a debugging section.

When the ALL REFERENCES OF identifier-1 phrase is specified, that debugging section is executed immediately after every statement that explicitly references identifier-1. If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed both

1. Immediately after the execution of any COBOL statement that explicitly references and replaces the contents of the data item referenced by identifier-1.

2. Immediately before the execution of any WRITE statement that explicitly references identifier-1.

Again, if identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

The debugging section associated with a specific operand is not executed more than once as a result of the execution of a single statement regardless of the number of times that operand is explicitly specified.

Within an imperative statement, each individial occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

A reference to file-name-1, identifier-1, or procedure-name-1 as a qualifier does not constitute reference to that item for the debugging described above.

The special register DEBUG-ITEM is associated with each execution of a debugging section. The register fields are updated automatically – in accordance with the rules of the MOVE statement – immediately before control is passed to the debugging section to provide information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01 DEBUG-ITEM.
    02 DEBUG-LINE          PICTURE IS X(6).
    02 FILLER              PICTURE IS X VALUE SPACE.
    02 DEBUG-NAME          PICTURE IS X(30).
    02 FILLER              PICTURE IS X VALUE SPACE.
    02 DEBUG-SUB-1         PICTURE IS 9999.
    02 FILLER              PICTURE IS X VALUE SPACE.
    02 DEBUG-SUB-2         PICTURE IS 9999.
    02 FILLER              PICTURE IS X VALUE SPACE.
    02 DEBUG-SUB-3         PICTURE IS 9999.
    02 FILLER              PICTURE IS X VALUE SPACE.
    02 DEBUG-CONTENTS      PICTURE IS X(n).
```

Contents of the register fields vary in accordance with USE FOR DEBUGGING specifications, as set forth in Table 9, below. In general terms, DEBUG-LINE contains information identifying a particular source statement. DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed. All qualifiers of the name are separated by the word 'IN' or 'OF'. Subscripts or indexes, if any, are not entered into DEBUG-NAME.

DEBUG-SUB-1, DEBUG-SUB-2, and DEBUG-SUB-3 contain spaces if the item that caused the debugging section to be executed is not subscripted or indexed. Otherwise, the occurrence number of each level is entered as necessary.

DEBUG-CONTENTS will be large enough to contain the required data.

Table 9. Contents of DEBUG-ITEM Register Fields

Contents / Field	Condition				
	1	2	3	4	5
DEBUG-LINE	Identifies source statement that referenced procedure-name-1.	Identifies previous statement.	Identifies statement that caused execution of USE procedure.	Identifies source statement that referenced file-name-1.	Identifies source statement that referenced identifier-1.
DEBUG-NAME	Contains name of procedure-name-1.	Name of procedure-name-1.	Name of procedure-name-1.	Name of file-name-1.	Name of identifier-1.
DEBUG-CONTENTS	Spaces, except after an ALTER statement, when it contains name of procedure-name-2 of the ALTER.	'FALL THROUGH'.	'USE PROCEDURE'.	Entire record read, or spaces.	Contents of identifier-1 after the execution of statement that referenced it.

Condition causing debugging section to be executed:

1 – Reference to procedure-name-1.
2 – Implicit control transfer from previous sequential paragraph to procedure-name-1.
3 – Procedure-name-1 is a USE procedure that is to be executed.
4 – References to file-name-1.
5 – Reference to identifier-1.

Debugging Lines

A debugging line is any line with a 'D' in the continuation indicator area of the line. The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines. A debugging line will be considered to have all the characteristics of a comment line if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

Successive debugging lines are allowed. Continuation of debugging lines is permitted subject to two constraints:

1. Each continuation line must contain a 'D' in the continuation indicator area.

2. Character-strings may not be broken across two lines.

Debugging lines are permitted in the program only after the OBJECT-COMPUTER paragraph and may not be used with the COPY REPLACING function.

Extended Debugging Features

The Xerox ANS COBOL language includes two debugging statements – TRACE and EXHIBIT – that are an extension of the ANS COBOL language and can be inserted anywhere in a COBOL source program. Use of these statements greatly reduces the time required to debug the logic of successfully compiled programs.

Omitting the DEBUG option in a COBOL processor control command containing other specified options suppresses these debugging statements at compilation time. This means that a debugged COBOL program need not be changed but merely recompiled to obtain an operational object program.

TRACE Statement

The format of this statement is

$$\left\{\begin{matrix}\underline{\text{READY}}\\ \underline{\text{RESET}}\end{matrix}\right\}\ \underline{\text{TRACE}}$$

The TRACE statement is used in two forms: READY TRACE starts the TRACE process in which a message is output on the system listing output device each time the execution of a new paragraph or section of the program begins; RESET TRACE terminates the notification of processing progress started by the preceding READY TRACE statement.

EXHIBIT Statement

The format of this statement is

$$\underline{\text{EXHIBIT}} \left\{ \begin{array}{l} \underline{\text{NAMED}} \\ \underline{\text{CHANGED}}\ \underline{\text{NAMED}} \\ \underline{\text{CHANGED}} \end{array} \right\} \left\{ \begin{array}{l} \text{data-name} \\ \text{non-numeric-literal} \end{array} \right\} \ldots$$

Note: The sum of the sizes of the operands of an EXHIBIT statement cannot exceed the maximum logical record length for the system listing output device.

The EXHIBIT NAMED statement causes a formatted display of the data-names (or non-numeric-literals) listed in the statement; the system listing output device is used. Output format for each data-name listed in the NAMED or CHANGED NAMED form of an EXHIBIT statement is

1. Blank

2. Original data-name (including qualifiers, if written)

3. Blank

4. Equal sign

5. Blank

6. Value of data-name

When displayed, literals listed in the statement are preceded by a blank.

The CHANGED form of the EXHIBIT statement provides for display of an item when it changes value (compared to the value at the previous time the EXHIBIT CHANGED statement was executed). The first time such a statement is executed, a value is considered to have changed; it is displayed and saved for purposes of comparison.

Only one data-name can be listed in an EXHIBIT CHANGEDstatement. Note that if two distinct EXHIBIT CHANGED data-name statements appear in a program, changes in data-name are associated with the two separate statements. Depending on the path of program flow, the values of data-name saved for comparison may differ for the two statements. If the list of operands in an EXHIBIT CHANGED statement includes literals, they are printed as remarks and are preceded by a blank.

The CHANGED NAMED form of the EXHIBIT statement causes a printout of each changed value for items listed in the statement. Only those values representing changes in their identifying names are printed. A fixed columnar format for the data to be displayed cannot be created with EXHIBIT CHANGED NAMED.

APPENDIX A. XEROX ANS COBOL RESERVED WORDS

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BLANK
BLOCK
BY

CALL
CANCEL
CF
CH
CHANGED[†]
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMMON-STORAGE
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-SUB1
DEBUG-SUB2
DEBUG-SUB3
DEBUG-NAME
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISPLAY
DIVIDE
DIVISION
DOWN

ELSE
END
ENDING
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE[†]
EXHIBIT[†]
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INCLUDE[†]
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY
KEYS

LABEL
LAST
LEADING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PROCEDURE
PROCEDURES
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID

QUOTE
QUOTES

RANDOM
RD
READ
READY
RECORD
RECORDS
REDEFINES
REEL
REFERENCES
RELEASE
REMAINDER
REMARKS
RENAMES
RENAMING[†]
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SELECTED
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
STRING
SUBTRACT
SUM
SYNC
SYNCHRONIZED

TALLY
TALLYING
TAPE
TERMINATE
THAN
THROUGH
THRU
TIMES
TO
TRACE
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

[†] Although still operational, these language forms have been made obsolete by changes to the COBOL standard.

APPENDIX B. SAMPLE XEROX ANS COBOL PROBLEM

In Figure B-1 a master tape file (each record consisting of a 5-digit account number, 21-character name, 6-digit quantity-on-hand, 6-digit unit price, and 6-digit date-record-established) is to be updated by a card file containing three types of cards. The program must ascertain the type of card and then branch to one of three routines to update the master file. The cards are the same format as the tape file except for the addition of the card code in the character following the date field. Codes that identify card type are

Code 1 Adds the quantity on the card to the quantity field of the master and reestablishes date of record.

Code 2 Subtracts the quantity on the card from the quantity field of the master and reestablishes date of record.

Code 3 (header card) Changes all data fields in the master record or establishes a new master record. Date of record must again be established.

Any other code is treated as an error condition. Multiple transactions of types 1 and 2 are allowed for any accounts already established in the master tape file. All input master records, plus new masters added, are written on an output master tape.

Various checks detect error conditions in the input card file that would cause a card to be punched. The following error flags indicate the type of error.

Card Flag 5 A card type other than 1, 2, or 3.

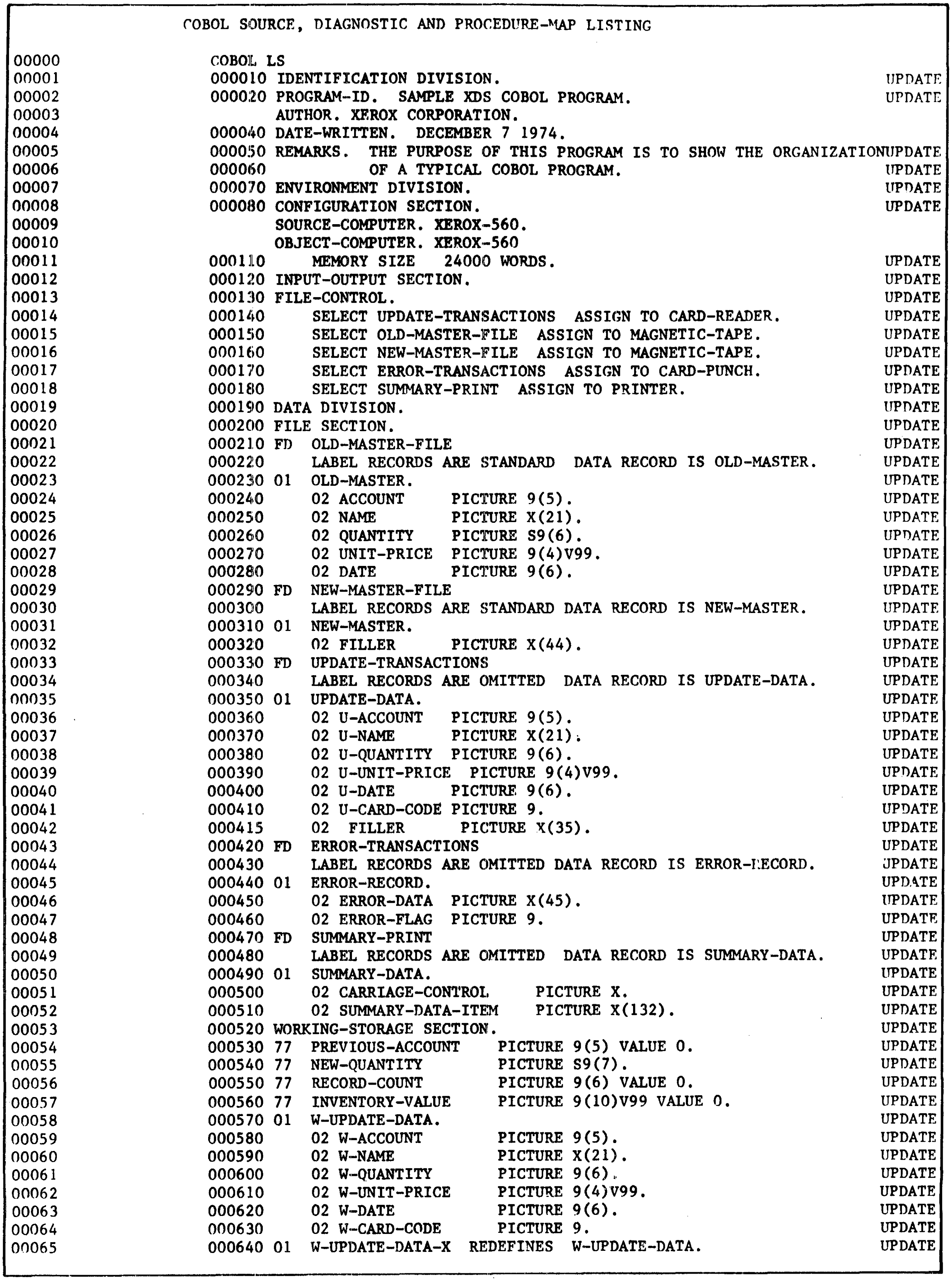
Card Flag 6 The input card file not in sequence by account number.

Card Flag 7 No matching master record for card types 1 or 2.

Card Flag 8 The quantity in the input card to be subtracted from the master record balance causes a negative balance.

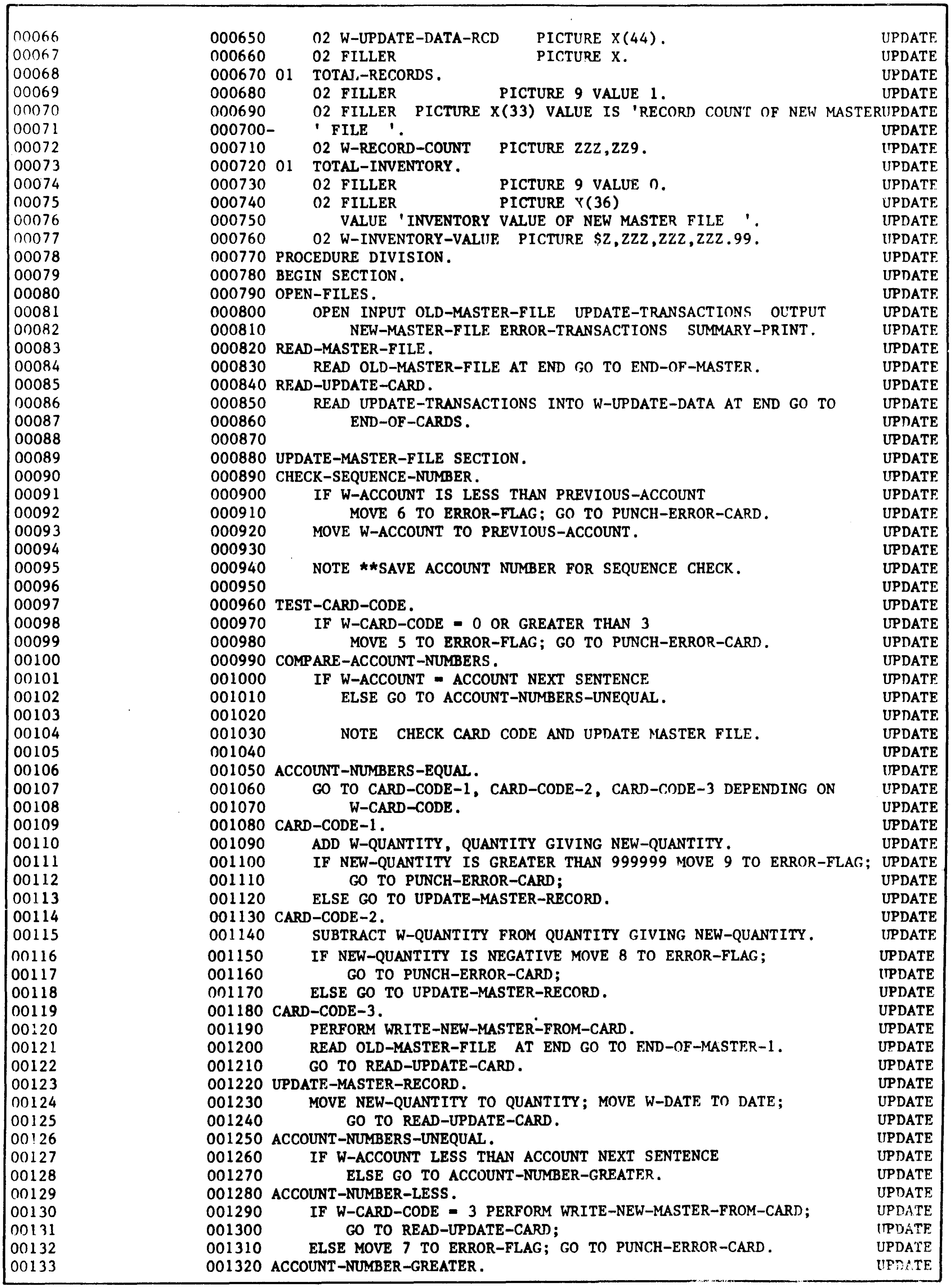
Card Flag 9 The quantity in the input card plus the master record balance overflows into the next field if not detected.

The format of these error cards is identical to that of the update cards with the addition of the card flag in the next available position following the card code. A record tally for the new master file is maintained together with the value of the inventory in the new master file. These items are printed on the standard system listing output device.



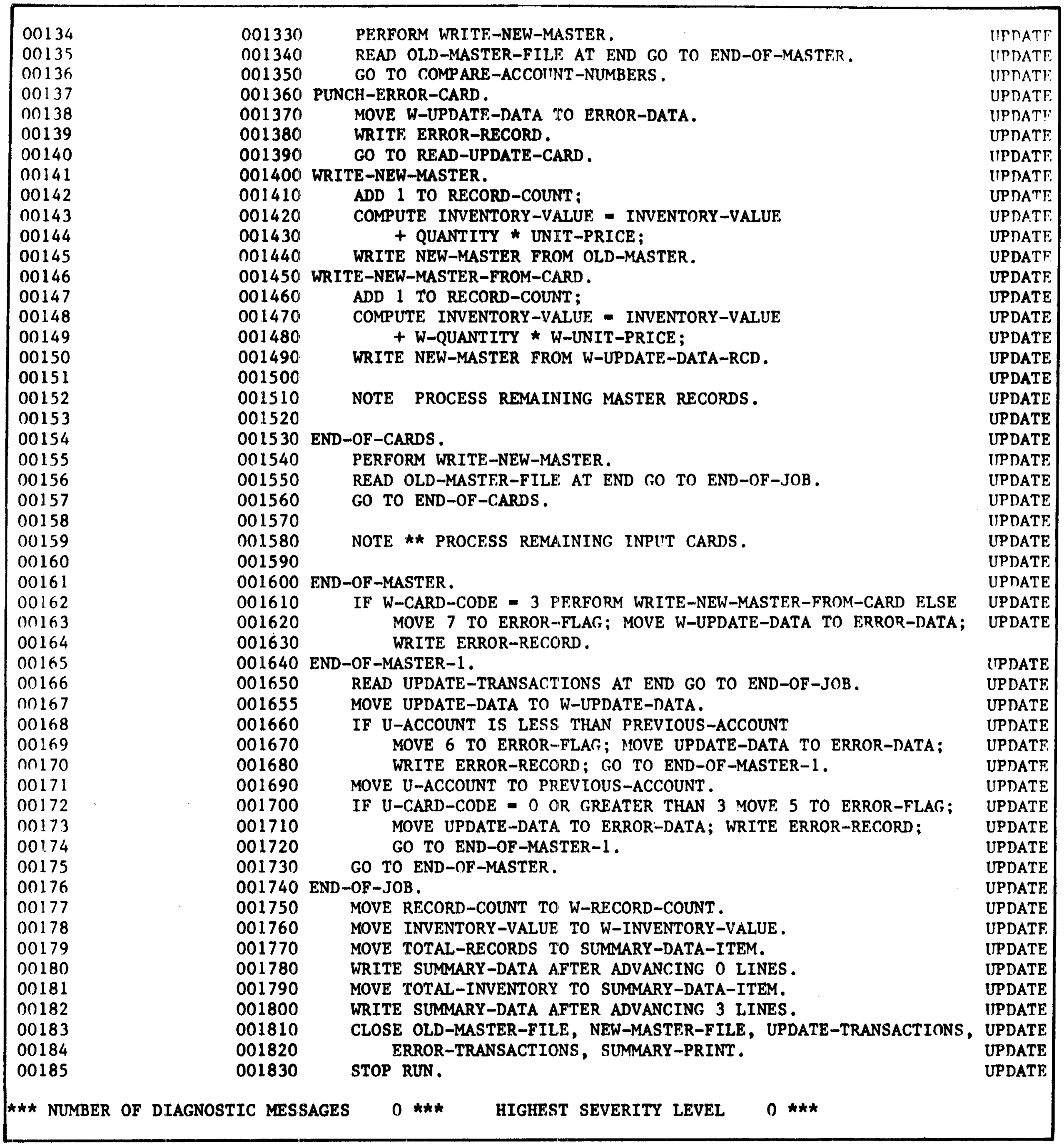
```
                  COBOL SOURCE, DIAGNOSTIC AND PROCEDURE-MAP LISTING

00000                COBOL LS
00001                000010 IDENTIFICATION DIVISION.                                         UPDATE
00002                000020 PROGRAM-ID.  SAMPLE XDS COBOL PROGRAM.                           UPDATE
00003                       AUTHOR. XEROX CORPORATION.
00004                000040 DATE-WRITTEN.  DECEMBER 7 1974.
00005                000050 REMARKS.  THE PURPOSE OF THIS PROGRAM IS TO SHOW THE ORGANIZATIONUPDATE
00006                000060           OF A TYPICAL COBOL PROGRAM.                            UPDATE
00007                000070 ENVIRONMENT DIVISION.                                            UPDATE
00008                000080 CONFIGURATION SECTION.                                           UPDATE
00009                       SOURCE-COMPUTER. XEROX-560.
00010                       OBJECT-COMPUTER. XEROX-560
00011                000110     MEMORY SIZE   24000 WORDS.                                   UPDATE
00012                000120 INPUT-OUTPUT SECTION.                                            UPDATE
00013                000130 FILE-CONTROL.                                                    UPDATE
00014                000140     SELECT UPDATE-TRANSACTIONS  ASSIGN TO CARD-READER.           UPDATE
00015                000150     SELECT OLD-MASTER-FILE  ASSIGN TO MAGNETIC-TAPE.             UPDATE
00016                000160     SELECT NEW-MASTER-FILE  ASSIGN TO MAGNETIC-TAPE.             UPDATE
00017                000170     SELECT ERROR-TRANSACTIONS  ASSIGN TO CARD-PUNCH.             UPDATE
00018                000180     SELECT SUMMARY-PRINT  ASSIGN TO PRINTER.                     UPDATE
00019                000190 DATA DIVISION.                                                   UPDATE
00020                000200 FILE SECTION.                                                    UPDATE
00021                000210 FD  OLD-MASTER-FILE                                              UPDATE
00022                000220     LABEL RECORDS ARE STANDARD  DATA RECORD IS OLD-MASTER.       UPDATE
00023                000230 01  OLD-MASTER.                                                  UPDATE
00024                000240     02 ACCOUNT     PICTURE 9(5).                                 UPDATE
00025                000250     02 NAME        PICTURE X(21).                                UPDATE
00026                000260     02 QUANTITY    PICTURE S9(6).                                UPDATE
00027                000270     02 UNIT-PRICE  PICTURE 9(4)V99.                              UPDATE
00028                000280     02 DATE        PICTURE 9(6).                                 UPDATE
00029                000290 FD  NEW-MASTER-FILE                                              UPDATE
00030                000300     LABEL RECORDS ARE STANDARD DATA RECORD IS NEW-MASTER.        UPDATE
00031                000310 01  NEW-MASTER.                                                  UPDATE
00032                000320     02 FILLER      PICTURE X(44).                                UPDATE
00033                000330 FD  UPDATE-TRANSACTIONS                                          UPDATE
00034                000340     LABEL RECORDS ARE OMITTED  DATA RECORD IS UPDATE-DATA.       UPDATE
00035                000350 01  UPDATE-DATA.                                                 UPDATE
00036                000360     02 U-ACCOUNT   PICTURE 9(5).                                 UPDATE
00037                000370     02 U-NAME      PICTURE X(21).                                UPDATE
00038                000380     02 U-QUANTITY  PICTURE 9(6).                                 UPDATE
00039                000390     02 U-UNIT-PRICE  PICTURE 9(4)V99.                            UPDATE
00040                000400     02 U-DATE      PICTURE 9(6).                                 UPDATE
00041                000410     02 U-CARD-CODE PICTURE 9.                                    UPDATE
00042                000415     02  FILLER      PICTURE X(35).                               UPDATE
00043                000420 FD  ERROR-TRANSACTIONS                                           UPDATE
00044                000430     LABEL RECORDS ARE OMITTED DATA RECORD IS ERROR-RECORD.       UPDATE
00045                000440 01  ERROR-RECORD.                                                UPDATE
00046                000450     02 ERROR-DATA  PICTURE X(45).                                UPDATE
00047                000460     02 ERROR-FLAG  PICTURE 9.                                    UPDATE
00048                000470 FD  SUMMARY-PRINT                                                UPDATE
00049                000480     LABEL RECORDS ARE OMITTED  DATA RECORD IS SUMMARY-DATA.      UPDATE
00050                000490 01  SUMMARY-DATA.                                                UPDATE
00051                000500     02 CARRIAGE-CONTROL     PICTURE X.                           UPDATE
00052                000510     02 SUMMARY-DATA-ITEM    PICTURE X(132).                      UPDATE
00053                000520 WORKING-STORAGE SECTION.                                         UPDATE
00054                000530 77  PREVIOUS-ACCOUNT    PICTURE 9(5) VALUE 0.                    UPDATE
00055                000540 77  NEW-QUANTITY        PICTURE S9(7).                           UPDATE
00056                000550 77  RECORD-COUNT        PICTURE 9(6) VALUE 0.                    UPDATE
00057                000560 77  INVENTORY-VALUE     PICTURE 9(10)V99 VALUE 0.                UPDATE
00058                000570 01  W-UPDATE-DATA.                                               UPDATE
00059                000580     02 W-ACCOUNT        PICTURE 9(5).                            UPDATE
00060                000590     02 W-NAME           PICTURE X(21).                           UPDATE
00061                000600     02 W-QUANTITY       PICTURE 9(6).                            UPDATE
00062                000610     02 W-UNIT-PRICE     PICTURE 9(4)V99.                         UPDATE
00063                000620     02 W-DATE           PICTURE 9(6).                            UPDATE
00064                000630     02 W-CARD-CODE      PICTURE 9.                               UPDATE
00065                000640 01  W-UPDATE-DATA-X  REDEFINES  W-UPDATE-DATA.                   UPDATE
```

Figure B-1. Sample Xerox ANS COBOL Program



```
000650     02 W-UPDATE-DATA-RCD    PICTURE X(44).                       UPDATE
000660     02 FILLER               PICTURE X.                           UPDATE
000670 01  TOTAL-RECORDS.                                               UPDATE
000680     02 FILLER           PICTURE 9 VALUE 1.                       UPDATE
000690     02 FILLER  PICTURE X(33) VALUE IS 'RECORD COUNT OF NEW MASTERUPDATE
000700-    ' FILE  '.                                                   UPDATE
000710     02 W-RECORD-COUNT   PICTURE ZZZ,ZZ9.                         UPDATE
000720 01  TOTAL-INVENTORY.                                             UPDATE
000730     02 FILLER           PICTURE 9 VALUE 0.                       UPDATE
000740     02 FILLER           PICTURE X(36)                            UPDATE
000750        VALUE 'INVENTORY VALUE OF NEW MASTER FILE  '.             UPDATE
000760     02 W-INVENTORY-VALUE  PICTURE $Z,ZZZ,ZZZ,ZZZ.99.             UPDATE
000770 PROCEDURE DIVISION.                                              UPDATE
000780 BEGIN SECTION.                                                   UPDATE
000790 OPEN-FILES.                                                      UPDATE
000800     OPEN INPUT OLD-MASTER-FILE  UPDATE-TRANSACTIONS  OUTPUT      UPDATE
000810         NEW-MASTER-FILE ERROR-TRANSACTIONS  SUMMARY-PRINT.       UPDATE
000820 READ-MASTER-FILE.                                                UPDATE
000830     READ OLD-MASTER-FILE AT END GO TO END-OF-MASTER.             UPDATE
000840 READ-UPDATE-CARD.                                                UPDATE
000850     READ UPDATE-TRANSACTIONS INTO W-UPDATE-DATA AT END GO TO     UPDATE
000860         END-OF-CARDS.                                            UPDATE
000870                                                                  UPDATE
000880 UPDATE-MASTER-FILE SECTION.                                      UPDATE
000890 CHECK-SEQUENCE-NUMBER.                                           UPDATE
000900     IF W-ACCOUNT IS LESS THAN PREVIOUS-ACCOUNT                   UPDATE
000910         MOVE 6 TO ERROR-FLAG; GO TO PUNCH-ERROR-CARD.            UPDATE
000920     MOVE W-ACCOUNT TO PREVIOUS-ACCOUNT.                          UPDATE
000930                                                                  UPDATE
000940     NOTE **SAVE ACCOUNT NUMBER FOR SEQUENCE CHECK.               UPDATE
000950                                                                  UPDATE
000960 TEST-CARD-CODE.                                                  UPDATE
000970     IF W-CARD-CODE = 0 OR GREATER THAN 3                         UPDATE
000980         MOVE 5 TO ERROR-FLAG; GO TO PUNCH-ERROR-CARD.            UPDATE
000990 COMPARE-ACCOUNT-NUMBERS.                                         UPDATE
001000     IF W-ACCOUNT = ACCOUNT NEXT SENTENCE                         UPDATE
001010        ELSE GO TO ACCOUNT-NUMBERS-UNEQUAL.                       UPDATE
001020                                                                  UPDATE
001030        NOTE  CHECK CARD CODE AND UPDATE MASTER FILE.             UPDATE
001040                                                                  UPDATE
001050 ACCOUNT-NUMBERS-EQUAL.                                           UPDATE
001060     GO TO CARD-CODE-1, CARD-CODE-2, CARD-CODE-3 DEPENDING ON     UPDATE
001070         W-CARD-CODE.                                             UPDATE
001080 CARD-CODE-1.                                                     UPDATE
001090     ADD W-QUANTITY, QUANTITY GIVING NEW-QUANTITY.                UPDATE
001100     IF NEW-QUANTITY IS GREATER THAN 999999 MOVE 9 TO ERROR-FLAG; UPDATE
001110         GO TO PUNCH-ERROR-CARD;                                  UPDATE
001120     ELSE GO TO UPDATE-MASTER-RECORD.                             UPDATE
001130 CARD-CODE-2.                                                     UPDATE
001140     SUBTRACT W-QUANTITY FROM QUANTITY GIVING NEW-QUANTITY.       UPDATE
001150     IF NEW-QUANTITY IS NEGATIVE MOVE 8 TO ERROR-FLAG;            UPDATE
001160         GO TO PUNCH-ERROR-CARD;                                  UPDATE
001170     ELSE GO TO UPDATE-MASTER-RECORD.                             UPDATE
001180 CARD-CODE-3.                                                     UPDATE
001190     PERFORM WRITE-NEW-MASTER-FROM-CARD.                          UPDATE
001200     READ OLD-MASTER-FILE  AT END GO TO END-OF-MASTER-1.          UPDATE
001210     GO TO READ-UPDATE-CARD.                                      UPDATE
001220 UPDATE-MASTER-RECORD.                                            UPDATE
001230     MOVE NEW-QUANTITY TO QUANTITY; MOVE W-DATE TO DATE;          UPDATE
001240         GO TO READ-UPDATE-CARD.                                  UPDATE
001250 ACCOUNT-NUMBERS-UNEQUAL.                                         UPDATE
001260     IF W-ACCOUNT LESS THAN ACCOUNT NEXT SENTENCE                 UPDATE
001270         ELSE GO TO ACCOUNT-NUMBER-GREATER.                       UPDATE
001280 ACCOUNT-NUMBER-LESS.                                             UPDATE
001290     IF W-CARD-CODE = 3 PERFORM WRITE-NEW-MASTER-FROM-CARD;       UPDATE
001300         GO TO READ-UPDATE-CARD;                                  UPDATE
001310     ELSE MOVE 7 TO ERROR-FLAG; GO TO PUNCH-ERROR-CARD.           UPDATE
001320 ACCOUNT-NUMBER-GREATER.                                          UPDATE
```

Figure B-1. Sample Xerox ANS COBOL Program (cont.)



```
00134          001330     PERFORM WRITE-NEW-MASTER.                                      UPDATE
00135          001340     READ OLD-MASTER-FILE AT END GO TO END-OF-MASTER.               UPDATE
00136          001350     GO TO COMPARE-ACCOUNT-NUMBERS.                                 UPDATE
00137          001360 PUNCH-ERROR-CARD.                                                  UPDATE
00138          001370     MOVE W-UPDATE-DATA TO ERROR-DATA.                              UPDATE
00139          001380     WRITE ERROR-RECORD.                                            UPDATE
00140          001390     GO TO READ-UPDATE-CARD.                                        UPDATE
00141          001400 WRITE-NEW-MASTER.                                                  UPDATE
00142          001410     ADD 1 TO RECORD-COUNT;                                         UPDATE
00143          001420     COMPUTE INVENTORY-VALUE = INVENTORY-VALUE                      UPDATE
00144          001430         + QUANTITY * UNIT-PRICE;                                   UPDATE
00145          001440     WRITE NEW-MASTER FROM OLD-MASTER.                              UPDATE
00146          001450 WRITE-NEW-MASTER-FROM-CARD.                                        UPDATE
00147          001460     ADD 1 TO RECORD-COUNT;                                         UPDATE
00148          001470     COMPUTE INVENTORY-VALUE = INVENTORY-VALUE                      UPDATE
00149          001480         + W-QUANTITY * W-UNIT-PRICE;                               UPDATE
00150          001490     WRITE NEW-MASTER FROM W-UPDATE-DATA-RCD.                       UPDATE
00151          001500                                                                    UPDATE
00152          001510     NOTE  PROCESS REMAINING MASTER RECORDS.                        UPDATE
00153          001520                                                                    UPDATE
00154          001530 END-OF-CARDS.                                                      UPDATE
00155          001540     PERFORM WRITE-NEW-MASTER.                                      UPDATE
00156          001550     READ OLD-MASTER-FILE AT END GO TO END-OF-JOB.                  UPDATE
00157          001560     GO TO END-OF-CARDS.                                            UPDATE
00158          001570                                                                    UPDATE
00159          001580     NOTE ** PROCESS REMAINING INPUT CARDS.                         UPDATE
00160          001590                                                                    UPDATE
00161          001600 END-OF-MASTER.                                                     UPDATE
00162          001610     IF W-CARD-CODE = 3 PERFORM WRITE-NEW-MASTER-FROM-CARD ELSE     UPDATE
00163          001620         MOVE 7 TO ERROR-FLAG; MOVE W-UPDATE-DATA TO ERROR-DATA;    UPDATE
00164          001630         WRITE ERROR-RECORD.
00165          001640 END-OF-MASTER-1.                                                   UPDATE
00166          001650     READ UPDATE-TRANSACTIONS AT END GO TO END-OF-JOB.              UPDATE
00167          001655     MOVE UPDATE-DATA TO W-UPDATE-DATA.                             UPDATE
00168          001660     IF U-ACCOUNT IS LESS THAN PREVIOUS-ACCOUNT                     UPDATE
00169          001670         MOVE 6 TO ERROR-FLAG; MOVE UPDATE-DATA TO ERROR-DATA;      UPDATE
00170          001680         WRITE ERROR-RECORD; GO TO END-OF-MASTER-1.                 UPDATE
00171          001690     MOVE U-ACCOUNT TO PREVIOUS-ACCOUNT.                            UPDATE
00172          001700     IF U-CARD-CODE = 0 OR GREATER THAN 3 MOVE 5 TO ERROR-FLAG;     UPDATE
00173          001710         MOVE UPDATE-DATA TO ERROR-DATA; WRITE ERROR-RECORD;        UPDATE
00174          001720         GO TO END-OF-MASTER-1.                                     UPDATE
00175          001730     GO TO END-OF-MASTER.                                           UPDATE
00176          001740 END-OF-JOB.                                                        UPDATE
00177          001750     MOVE RECORD-COUNT TO W-RECORD-COUNT.                           UPDATE
00178          001760     MOVE INVENTORY-VALUE TO W-INVENTORY-VALUE.                     UPDATE
00179          001770     MOVE TOTAL-RECORDS TO SUMMARY-DATA-ITEM.                       UPDATE
00180          001780     WRITE SUMMARY-DATA AFTER ADVANCING 0 LINES.                    UPDATE
00181          001790     MOVE TOTAL-INVENTORY TO SUMMARY-DATA-ITEM.                     UPDATE
00182          001800     WRITE SUMMARY-DATA AFTER ADVANCING 3 LINES.                    UPDATE
00183          001810     CLOSE OLD-MASTER-FILE, NEW-MASTER-FILE, UPDATE-TRANSACTIONS,   UPDATE
00184          001820         ERROR-TRANSACTIONS, SUMMARY-PRINT.                         UPDATE
00185          001830     STOP RUN.                                                      UPDATE

*** NUMBER OF DIAGNOSTIC MESSAGES     0 ***     HIGHEST SEVERITY LEVEL     0 ***
```

Figure B-1. Sample Xerox ANS COBOL Program (cont.)

APPENDIX C. SLACK BYTES

Although the smallest individually addressable unit in the Xerox computing systems is the byte or the character, the basic unit of information is the word and many computer operations address computer words and even doublewords. COBOL is a character-oriented programming language, but efficiency demands that certain data types expressable in COBOL be positioned on the appropriate boundary to render the data readily accessible to the addressing capabilities of the computer.

Accordingly, the compiler assures that logical records originate upon doubleword boundaries at object time. Within records, the compiler assumes that each data item is positioned at the boundary, relative to the base of the record, that is appropriate to its type.

If the programmer specifies a USAGE of COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or INDEX within a data hierarchy, it may become necessary to insert bytes between adjacent data items in order to ensure correct boundary alignment of all data items within the hierarchy. These bytes contain no information: their sole purpose is to effect proper boundary alignment of data items, when necessary. Bytes of this nature may be termed "slack bytes". Consider the example

```
01    A.

    02   B     PICTURE X.

    02   C     USAGE COMPUTATIONAL.
```

If A is considered the origin of the record, data-name B occupies the first byte within the record. The next available storage space for data-name C, therefore, begins at the second byte within the record. However, Xerox ANS COBOL dictates that all data entries with USAGE COMPUTATIONAL occupy one word (4 bytes) of storage. Since the manipulation performed on a COMPUTATIONAL data entry word is word-oriented, any COMPUTATIONAL data entry must be aligned on a word boundary. To achieve this desired effect the compiler introduces slack bytes between data-names B and C, thereby causing C to be aligned on a word boundary. The example above then transforms effectively to

```
01    A.

    02   B     PICTURE X.

    02   SLACK-BYTES PICTURE XXX.

    02   C     USAGE COMPUTATIONAL.
```

Note that an actual data entry is never generated for slack bytes; they are introduced merely by adjusting the displacement counter relative to the origin of the record. The form used in the example is given only to help clarify location of the slack bytes.

Actually, when data-name B is encountered its relative byte displacement within the record is assigned a value of 0. Since B is one byte in length, the displacement counter is incremented by 1. When C is scanned and it is determined that a word boundary alignment is required, the displacement counter is set forward to the nearest word boundary: in this case, 4. Bytes 1, 2, and 3 then become slack bytes.

Another case of particular interest appears when an OCCURS clause exists on a group data-name and a data-name within the group is of USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or INDEX. For example,

```
01    A.

    02   B     OCCURS 5 TIMES.

        03   C     USAGE COMPUTATIONAL.

        03   D     PICTURE X.
```

For the first occurrence of group B, data-name C is aligned on a word boundary and occupies the first 4 bytes of the record. Data-name D occupies the 5th byte. For the second occurrence of group B, however, it appears that data-name C begins on the 6th byte of the record; this is not allowable, since the 6th byte of the record is not a word boundary. Therefore, slack bytes must be introduced at the end of group B to ensure that each occurrence of C within B falls on a word boundary. The example then transforms to

```
01    A.
   02    B      OCCURS 5 TIMES.
      03    C      USAGE COMPUTATIONAL.
      03    D      PICTURE X.
      03    SLACK-BYTES PICTURE XXX.
```

A summary of USAGEs and their appropriate boundary alignments follows.

USAGE	Type of Alignment	Byte Multiple
COMPUTATIONAL	Word	4
COMPUTATIONAL-1	Word	4
COMPUTATIONAL-2	Doubleword	8
INDEX	Word	4

Values in the third column represent the byte modulus utilized in computation of the data entry boundary under consideration. For example, the displacement counter currently has a value of 5. It is designed to align the next data entry on a word boundary. In order to determine the next available word boundary, the current displacement is divided by the byte modulus. If the remainder of the division is 0, no adjustment of the displacement is required, as the displacement is already at the correct boundary; otherwise, the remainder from the division is subtracted from the byte modulus and the resultant number of slack bytes must be generated. For this example, displacement = 5 and the byte modulus = 4. The division 5/4 yields a remainder of 1. Subtracting 1 from the byte modulus 4 gives the final answer of 3. To restate:

Upon dividing the displacement by the byte modulus (bm),

if remainder = 0, no action required;

if remainder ≠ 0, bm - remainder = number of slack bytes to be generated.

APPENDIX D. EVALUATION OF ARITHMETIC-EXPRESSIONS

Arithmetic-expressions in the Xerox ANS COBOL may be performed in integer binary (fixed-point), decimal, or floating-point (either single or double precision). The mode is chosen by the USAGE of operands and, to some extent, the final result items (i. e., when the operand USAGE differs, the result may determine the mode). With certain exceptions, as much significance is retained as the available arithmetic instructions in the chosen modes themselves provide; sometimes additional operations are performed to obtain more precision.

Number Representation

All arithmetic operations, comparisons, and data movements use instructions that require the pair of operands involved to be of the same USAGE; thus the compiler must insert code to convert data from one USAGE to another. The USAGEs are

1. INDEX/COMPUTATIONAL: binary integer

   These index or data items are carried as signed 31-bit integer quantities; maximum absolute value is $2^{31}-1$ or approximately 2.15×10^9.

2. COMPUTATIONAL-1: single precision floating-point

   These data items consist of a sign bit and a 7-bit biased hexadecimal exponent, and cover a normalized range from 16^{-65} to $(1 - 16^{-6}) \times 16^{63}$ or approximately 5.4×10^{-79} to 6.5×10^{63}.

3. COMPUTATIONAL-2: double precision floating-point

   These data items consist of a sign bit, a 7-bit characteristic, and 56 bits of significance and cover a normalized range from 16^{-65} to $(1 - 16^{-14}) \times 16^{63}$ or approximately 5.4×10^{-79} to 7.2×10^{75}.

4. COMPUTATIONAL-3: packed decimal

   These data items consist of a maximum of 30 digits (the numbers 0 through 9, in 4-bit code) and a decimal sign character X'A' - X'F' (X'B' and X'D' are negative; all others positive) occupying the right half of the rightmost byte. Unsigned fields are always positive. Additional truncation may be needed before the conversion if the desired internal format required it.

5. DISPLAY: zoned (unpacked) decimal

   These data items consist of the numbers 0 through 9 in 8-bit EBCDIC. The left half of the rightmost byte contains the 4-bit decimal sign. Although there are no limitations placed on these items as to "size", the maximum for conversion to other USAGEs is 30 digits.

Numeric Conversion

Loss of accuracy can result when it is necessary to convert from one USAGE to another. The following discussion indicates the accuracy of the conversion and the manner in which truncation is effected as data is converted.

1. INDEX/COMPUTATIONAL

   a. To COMPUTATIONAL-1: 24 bits are retained; the least significant bits in groups of 4 bits (maximum of 8 bits) are lost if truncation is necessary.

   b. To COMPUTATIONAL-2: All 31 bits are retained.

   c. To COMPUTATIONAL-3: All 31 bits are converted; truncation is possible when result is decimal-point aligned if decimal field size to left of decimal point is ≤10 digits.

   d. To DISPLAY: (Same as 1.c. except unpacking is necessary.)

2. COMPUTATIONAL-1

   a. To INDEX/COMPUTATIONAL: Extension to double precision floating-point is pe·formed first. The integer value, consisting of a maximum of 24 bits, is retained. The most significant bits are lost if the value 2^{31}. Zero fill takes place on the right if truncation is necessary to obtain an integer.

   b. To COMPUTATIONAL-2: Extension to double precision floating-point; least significant 32 bits are zero filled. No loss of significance.

   c. To COMPUTATIONAL-3: (Same as 2.b., then conversion to packed decimal. Truncation loss may result due to point alignment after conversion.)

   d. To DISPLAY: (Same as 2.c. except unpacking is required.)

3. COMPUTATIONAL-2

   a. To INDEX/COMPUTATIONAL: (Same as 2.a. except extension to double precision is unnecessary. A maximum of 31 bits of significance is retained.)

   b. To COMPUTATIONAL-1: Least significant half (word 2) of double precision floating-point number is discarded.

   c. To COMPUTATIONAL-3: (Same as 2.c. except extension to double precision is unnecessary.)

   d. To DISPLAY: (Same as 3.c. except unpacking is required.)

4. COMPUTATIONAL-3

   a. To INDEX/COMPUTATIONAL: Fractional digits are discarded; maximum value retained is $2^{31}-1$ or approximately 2.15×10^9.

   b. To COMPUTATIONAL-1: Maximum of 30 decimal digits are converted to a double precision floating-point value. The characteristic is adjusted and the value normalized. Only the most significant half (word 1) of the converted value is retained.

   c. To COMPUTATIONAL-2: (Same as 4.b. except the entire double precision floating-point number formed is used.)

   d. To DISPLAY: Unpack; no loss of accuracy.

5. DISPLAY

   a. To INDEX/COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2: Same as 4.a., 4.b., and 4.c., respectively, except preliminary packing is required.

   b. To COMPUTATIONAL-3: Pack; no loss of accuracy.

Intermediate Results

When the arithmetic-expression contains only a pair of operands, intermediate results are not generated. The operands may require conversion to the internal representation suitable to the chosen mode of operation; point alignment procedures may be performed either before or after execution of the arithmetic operation, or both. If multiple receiving items are specified, the final result is converted and aligned as required for each result item.

An intermediate result is generated when the arithmetic-expression contains a series of operands or arithmetic operations as in

1. An ADD or SUBTRACT statement with multiple operands.

2. A COMPUTE statement comprised of a series of arithmetic operations.

3. Arithmetic-expressions contained within conditions in IF, PERFORM, or SEARCH statements.

The mode of computation remains the same when the operands and result are all of the same USAGE, requiring possible additional instructions only for point alignment and added precision, when required. However, with multiple operands, a change in the mode of computation is required if there is different USAGE. Such a change in mode requires conversion of the intermediate result to the proper USAGE.

Decimal Scaling of Intermediate Results

The following abbreviations are used in discussion of the compiler algorithms for determining the number of integers and decimal place composing the intermediate result in the three modes of operation.

Op. 1	First operand in arithmetic statement.
Op. 2	Second operand in arithmetic statement. Specifically, for DIVIDE (/), the divisor and for exponentiation, the exponent.
IR	Intermediate result.
I1, I2, I	Number of integer places in the Op. 1, Op. 2, and IR, respectively.
D1, D2, D	Number of decimal (fractional) places in Op. 1, Op. 2, and IR, respectively.
D max	Maximum number of decimal (fractional) places defined for any operand or receiving field.

COMPUTATIONAL (Binary Interger Mode)

All operations are carried out in single precision arithmetic when both Op. 1 and Op. 2 are binary (i.e., USAGE COMPUTATIONAL). Thus the maximum value of IR is $2^{31} - 1$, maximum value of I is 10, and D is always zero.

COMPUTATIONAL-1, COMPUTATIONAL-2 (Floating-Point Mode-Either Single or Double Precision)

If both Op. 1 and Op. 2 are COMPUTATIONAL-1 or single precision, the operation is carried out in "short" floating-point arithmetic where up to 24 bits of precision are retained in IR. Otherwise, "long" floating-point arithmetic is used and 56 bits of precision are retained.

The value of I and D are not meaningful except during conversion to or from floating-point.

COMPUTATIONAL-3 (Decimal Mode)

In this mode I + D consists of a maximum of 30 digits. If the value of I + D exceeds 30, the intermediate result must be computed differently to avoid the need for using more than 30 digits of precision.

The number of integer places (I) contained in IR is obtained by determining the number of integers constituting the result produced if the arithmetic statement were performed using the worst possible values in each operand.

1. If the operand is a data-name, the maximum value of the data-name as defined by the PICTURE (i.e., maximum value of PICTURE 9V99 is 9.99) is used.
2. If the operand is an intermediate result, the maximum value that can be placed in the number of positions chosen for the previous arithmetic result is used.
3. If the operation is division, the minimum nonzero value of the digit in the PICTURE for the data-name (i.e., minimum value for the PICTURE 9V99 is 0.01) is used as the divisor. If Op.2 is an intermediate result, the minimum nonzero value of the PICTURE that represents the intermediate result is used.

The number of decimal (fractional) places contained in IR is calculated for the different arithmetic operations as follows:

Operation	Number of Decimal Places
ADD (+) or SUBTRACT (-)	Max (D1, D2)
MULTIPLY (*)	D1 + D2
DIVIDE (/)	Max (D1 - D2, D max)
Exponentiation (**)	D max if Op. 2 is a nonintegral or a data-name, or D1 * Op. 2 if Op. 2 is an integral literal

Table D-1 illustrates the value of I and D carried in IR for the number of integer (i) and decimal (d) places obtained by the methods described above under these exceptional circumstances.

Table D-1. I and D Values

Value i + d	Value of d	Value of i + D max	Value of I in IR	Value of D in IR
≤ 30	any value	any value	i	d
> 30	≤ D max	any value	30 - d (truncate i)	d
	> D max	≤ 30	i	30 - i (truncate d)
		> 30	30 - D max (truncate i)	D max

In case part of the intermediate result must be truncated at some time during computation, the compiler issues a compile-time diagnostic.

ROUNDED and SIZE ERROR Options

The USAGEs of the final result and of the receiving item determine the manner in which rounding is accomplished and the SIZE ERROR condition is detected (see Table D-2). In all cases, overflow conditions detected earlier in the arithmetic-expression while intermediate results were being obtained or converted are also considered a SIZE ERROR. The overflow conditions detected during the execution of arithmetic instructions and the process or point alignment for either intermediate or final results are also recognized as SIZE ERROR conditions. If the SIZE ERROR statement is not used and the computation produces overflow, the result item receives an unpredictable value.

Table D-2. ROUNDED and SIZE ERROR Options

Type	Final Result	Receiving Item	Rounding	Size Error
1	COMPUTATIONAL	COMPUTATIONAL	None	If ≥ 2^{31}
2	COMPUTATIONAL	COMPUTATIONAL-1	None	If ≥ 2^{31}
3	COMPUTATIONAL	COMPUTATIONAL-2	None	If ≥ 2^{31}
4	COMPUTATIONAL	COMPUTATIONAL-3	After conversion to decimal	If ≥ 2^{31} or if there is left decimal overflow after converting
5	COMPUTATIONAL-1	COMPUTATIONAL	None	Floating-point overflow or if ≥ 2^{31}
6	COMPUTATIONAL-1	COMPUTATIONAL-1	None	Floating-point overflow
7	COMPUTATIONAL-1	COMPUTATIONAL-2	None	Floating-point overflow
8	COMPUTATIONAL-1	COMPUTATIONAL-3	After conversion to decimal	Floating-point overflow or if there is left decimal overflow after converting, rounding, and alignment
9	COMPUTATIONAL-2	COMPUTATIONAL	None	Same as 5
10	COMPUTATIONAL-2	COMPUTATIONAL-1	None	Same as 6
11	COMPUTATIONAL-2	COMPUTATIONAL-2	None	Floating-point overflow
12	COMPUTATIONAL-2	COMPUTATIONAL-3	After conversion to decimal	Same as 8
13	COMPUTATIONAL-3	COMPUTATIONAL	None	If ≥ 2^{31}
14	COMPUTATIONAL-3	COMPUTATIONAL-1	None	Same as 6
15	COMPUTATIONAL-3	COMPUTATIONAL-2	None	Same as 6
16	COMPUTATIONAL-3	COMPUTATIONAL-3	If point location of final result is to left of that of receiving item	If there is left decimal overflow after rounding and alignment

APPENDIX E. SORT FEATURE SAMPLE PROGRAM

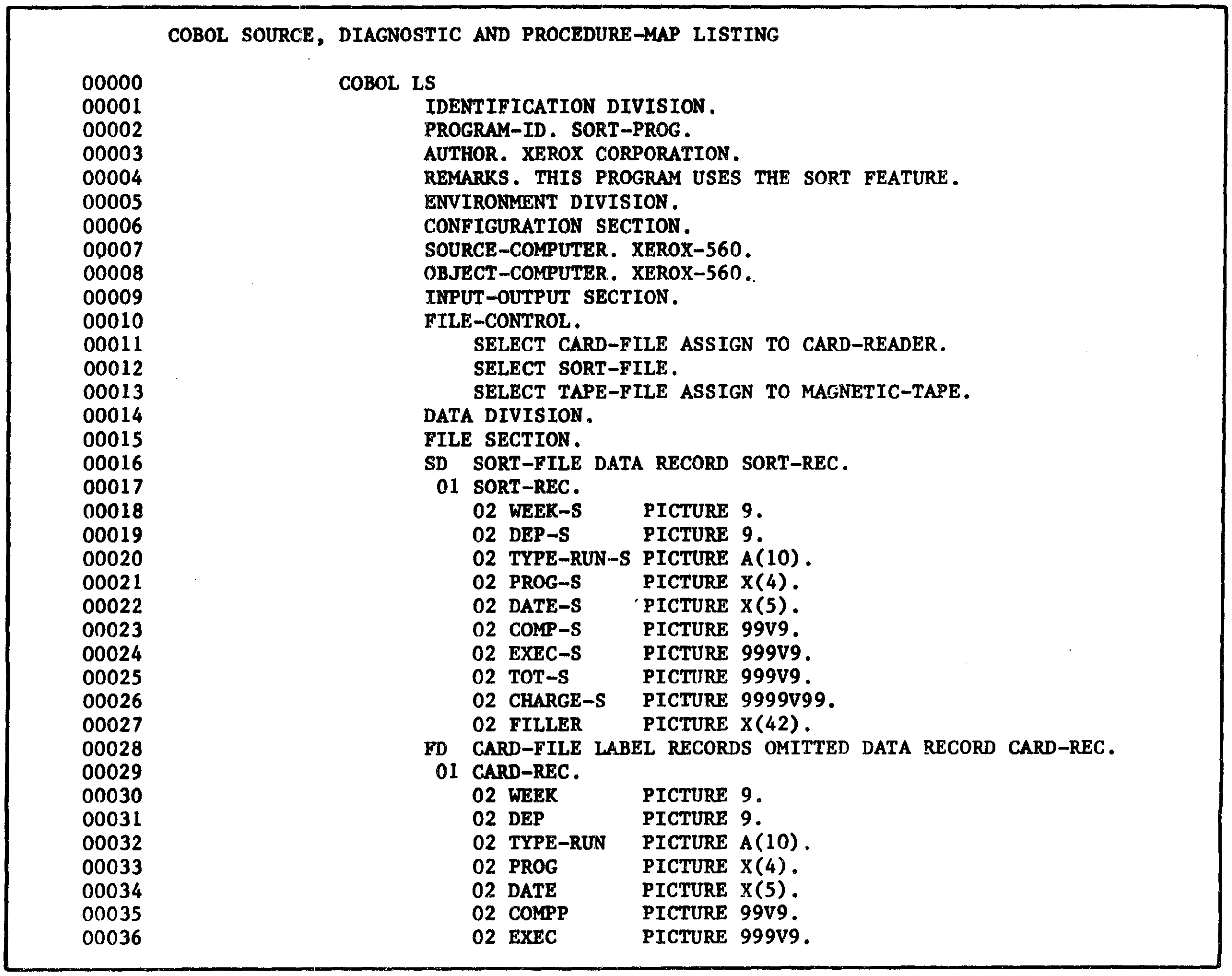
The program in Figure E-1 illustrates the Sort feature, which sorts and adds information to records dealing with computer usage. The format of the cards to be sorted is Week, Column 1; Department, Column 2; Type-Run, Columns 3 through 12; Program, Columns 13 through 16; Date, Columns 17 through 21; Compilation Time, Columns 22 through 24; and Execution Time, Columns 25 through 28.

The order of sort as specified by the ASCENDING KEY and DESCENDING KEY clauses is

1. By Department, lowest first.
2. By Week, lowest first.
3. By Type-Run, highest first (T before P).
4. By Program Identification, lowest first.
5. By Date, lowest first.

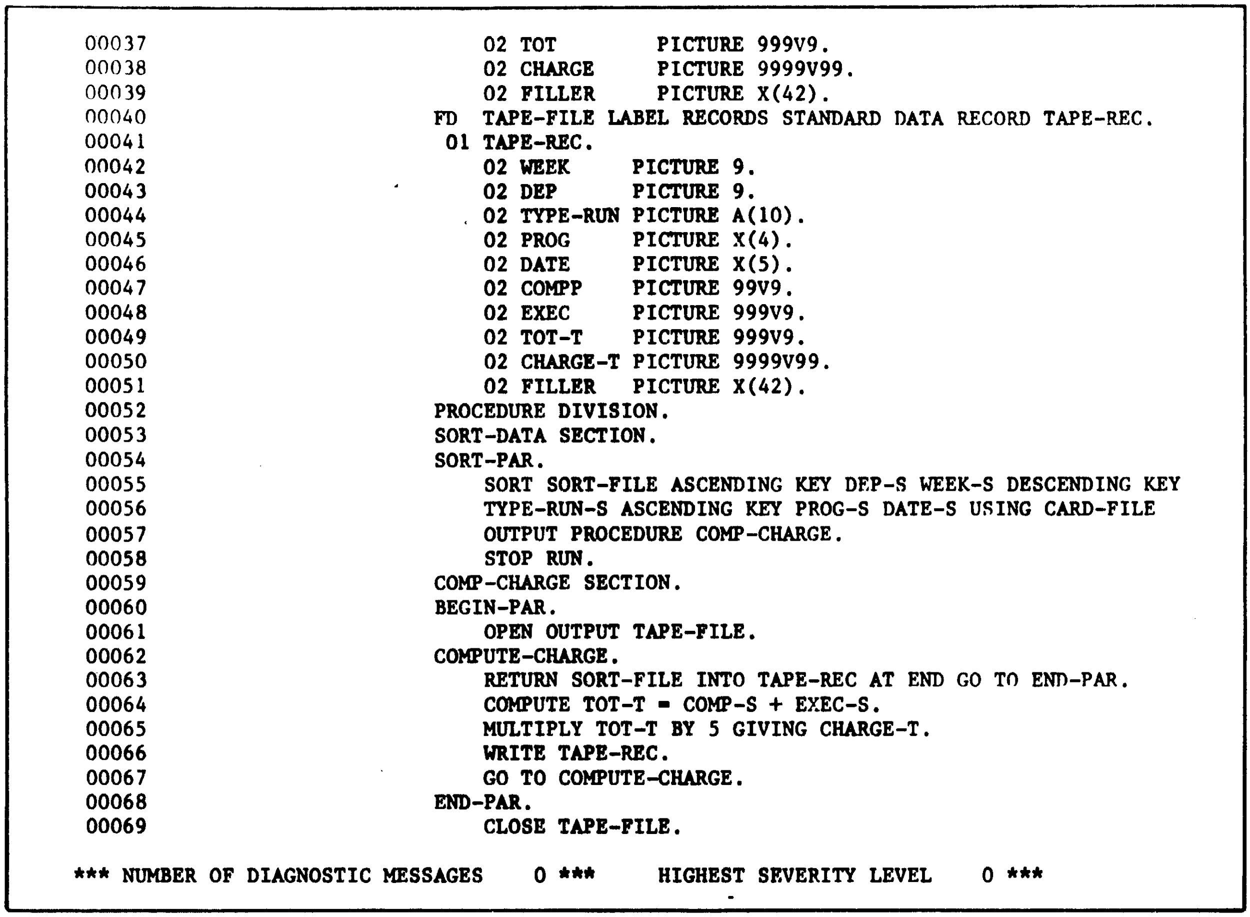
USING CARDFILE automatically causes CARDFILE to be opened, read, passed to the SORTFILE, and closed when the SORT statement is executed.

The sorted records are available from the SORTFILE by use of the RETURN statement during the OUTPUT PROCEDURE. The OUTPUT PROCEDURE (COMP-CHARGE SECTION) computes the total charge and outputs the data on tape.



```
COBOL SOURCE, DIAGNOSTIC AND PROCEDURE-MAP LISTING

              COBOL LS
                     IDENTIFICATION DIVISION.
                     PROGRAM-ID. SORT-PROG.
                     AUTHOR. XEROX CORPORATION.
                     REMARKS. THIS PROGRAM USES THE SORT FEATURE.
                     ENVIRONMENT DIVISION.
                     CONFIGURATION SECTION.
                     SOURCE-COMPUTER. XEROX-560.
                     OBJECT-COMPUTER. XEROX-560..
                     INPUT-OUTPUT SECTION.
                     FILE-CONTROL.
                         SELECT CARD-FILE ASSIGN TO CARD-READER.
                         SELECT SORT-FILE.
                         SELECT TAPE-FILE ASSIGN TO MAGNETIC-TAPE.
                     DATA DIVISION.
                     FILE SECTION.
                     SD  SORT-FILE DATA RECORD SORT-REC.
                      01 SORT-REC.
                         02 WEEK-S     PICTURE 9.
                         02 DEP-S      PICTURE 9.
                         02 TYPE-RUN-S PICTURE A(10).
                         02 PROG-S     PICTURE X(4).
                         02 DATE-S     PICTURE X(5).
                         02 COMP-S     PICTURE 99V9.
                         02 EXEC-S     PICTURE 999V9.
                         02 TOT-S      PICTURE 999V9.
                         02 CHARGE-S   PICTURE 9999V99.
                         02 FILLER     PICTURE X(42).
                     FD  CARD-FILE LABEL RECORDS OMITTED DATA RECORD CARD-REC.
                      01 CARD-REC.
                         02 WEEK       PICTURE 9.
                         02 DEP        PICTURE 9.
                         02 TYPE-RUN   PICTURE A(10).
                         02 PROG       PICTURE X(4).
                         02 DATE       PICTURE X(5).
                         02 COMPP      PICTURE 99V9.
                         02 EXEC       PICTURE 999V9.
```

Figure E-1. SORT Program



```
    02 TOT        PICTURE 999V9.
    02 CHARGE     PICTURE 9999V99.
    02 FILLER     PICTURE X(42).
FD  TAPE-FILE LABEL RECORDS STANDARD DATA RECORD TAPE-REC.
 01 TAPE-REC.
    02 WEEK     PICTURE 9.
    02 DEP      PICTURE 9.
    02 TYPE-RUN PICTURE A(10).
    02 PROG     PICTURE X(4).
    02 DATE     PICTURE X(5).
    02 COMPP    PICTURE 99V9.
    02 EXEC     PICTURE 999V9.
    02 TOT-T    PICTURE 999V9.
    02 CHARGE-T PICTURE 9999V99.
    02 FILLER   PICTURE X(42).
PROCEDURE DIVISION.
SORT-DATA SECTION.
SORT-PAR.
    SORT SORT-FILE ASCENDING KEY DEP-S WEEK-S DESCENDING KEY
    TYPE-RUN-S ASCENDING KEY PROG-S DATE-S USING CARD-FILE
    OUTPUT PROCEDURE COMP-CHARGE.
    STOP RUN.
COMP-CHARGE SECTION.
BEGIN-PAR.
    OPEN OUTPUT TAPE-FILE.
COMPUTE-CHARGE.
    RETURN SORT-FILE INTO TAPE-REC AT END GO TO END-PAR.
    COMPUTE TOT-T = COMP-S + EXEC-S.
    MULTIPLY TOT-T BY 5 GIVING CHARGE-T.
    WRITE TAPE-REC.
    GO TO COMPUTE-CHARGE.
END-PAR.
    CLOSE TAPE-FILE.

*** NUMBER OF DIAGNOSTIC MESSAGES     0 ***      HIGHEST SEVERITY LEVEL     0 ***
```

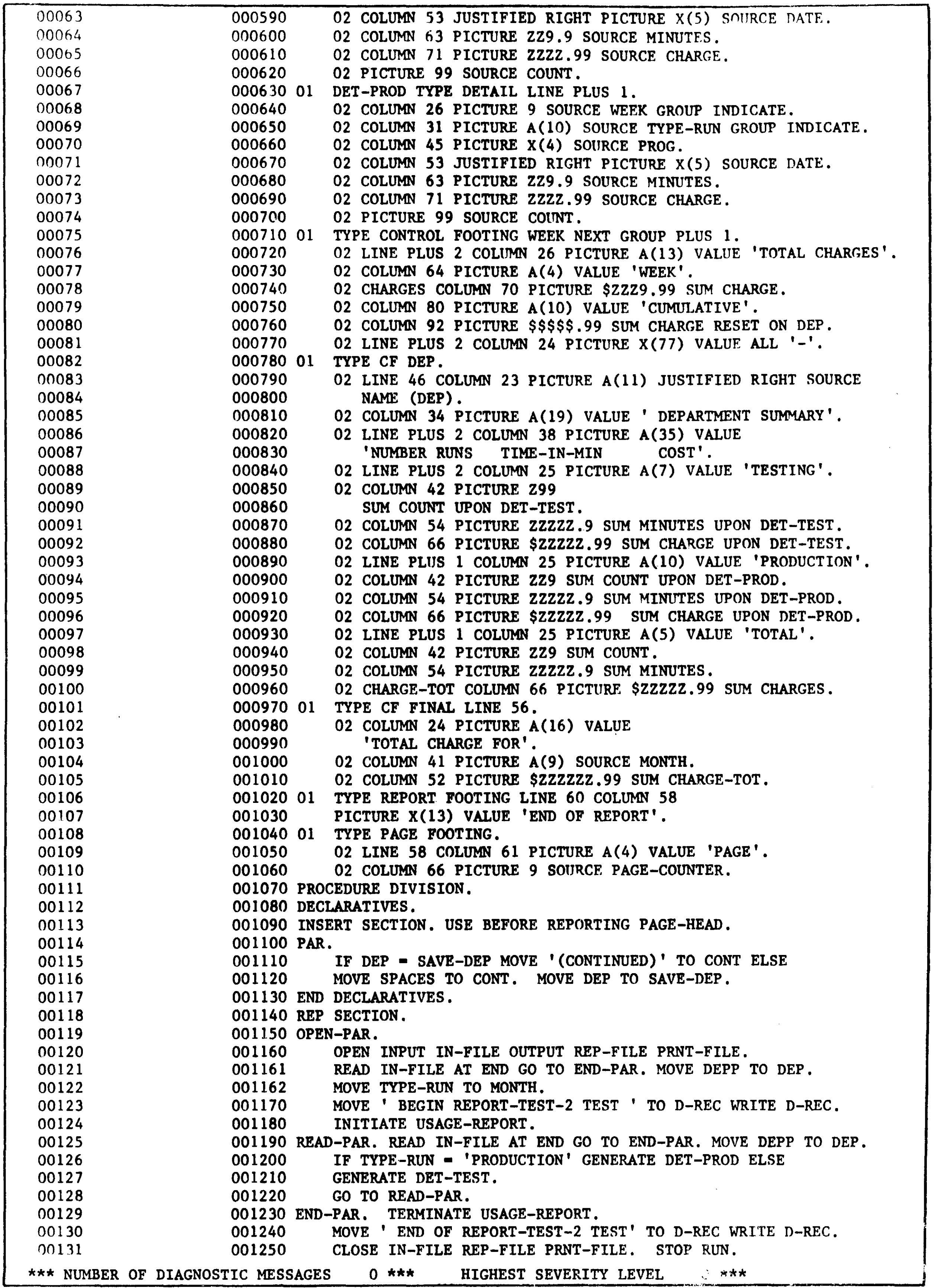
Figure E-1. SORT Program (cont.)

APPENDIX F. REPORT WRITER SAMPLE PROGRAM

Figure F-1 is an example of a program using the Report Writer feature. Figure F-2 shows a Report Writer generated report.

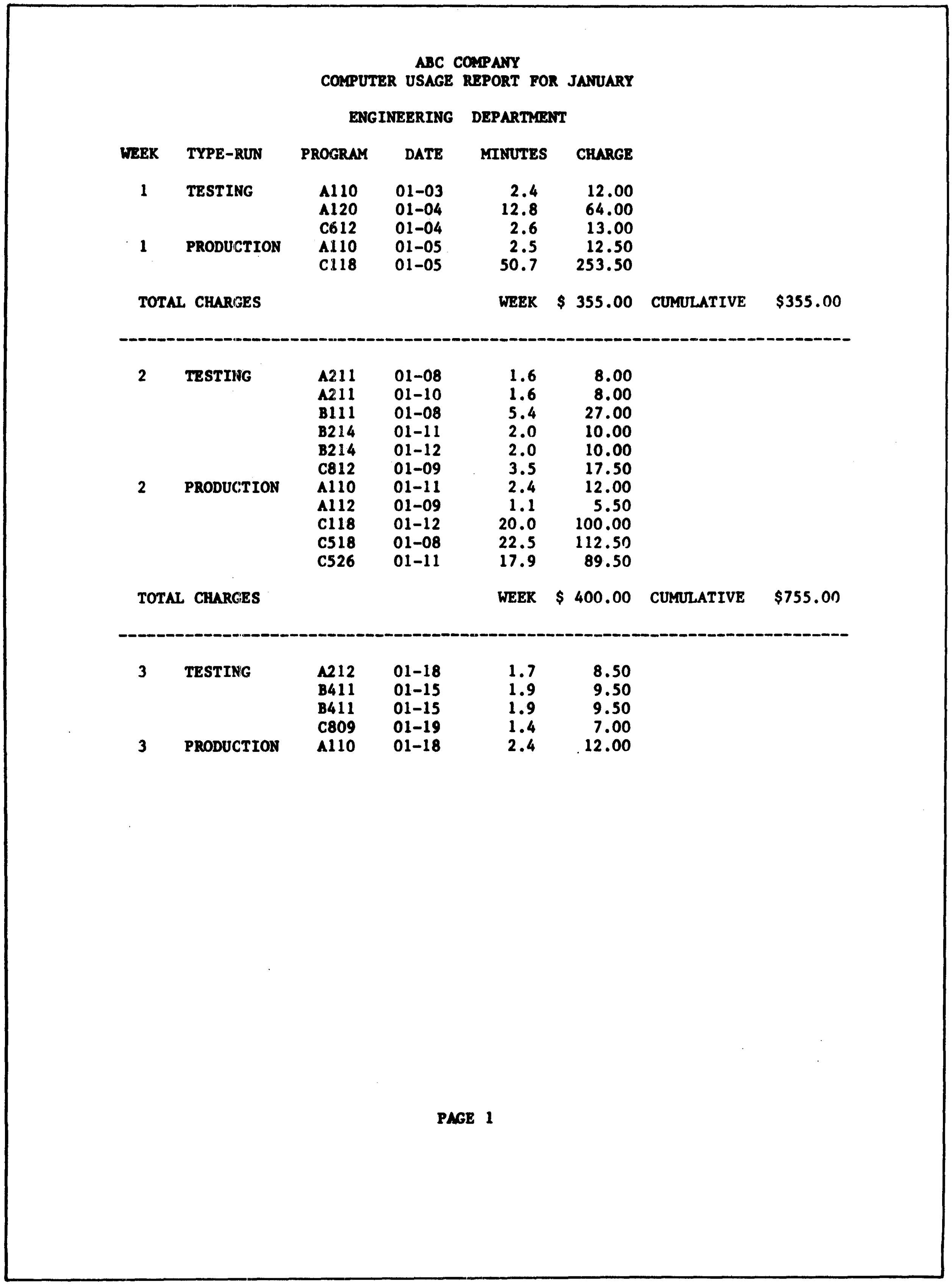
```
COBOL SOURCE, DIAGNOSTIC AND PROCEDURE-MAP LISTING
        COBOL LS,GO
        000010 IDENTIFICATION DIVISION.
        000020 PROGRAM-ID. REPORT-TEST-2.
               AUTHOR. XEROX CORPORATION.
        000030 REMARKS. THIS PROGRAM USES THE REPORT WRITER FEATURE.
        000040 ENVIRONMENT DIVISION.
        000050 CONFIGURATION SECTION.
               SOURCE-COMPUTER. XEROX-560.
               OBJECT-COMPUTER. XEROX-560.
        000080 INPUT-OUTPUT SECTION.
        000090 FILE-CONTROL.
        000100     SELECT IN-FILE ASSIGN TO CARD-READER.
        000110     SELECT REP-FILE ASSIGN TO PRINTER.
        000115     SELECT PRNT-FILE ASSIGN TO PRINTER.
        000120 DATA DIVISION.
        000130 FILE SECTION.
        000140 FD  IN-FILE LABEL RECORDS ARE STANDARD DATA RECORD IS IN-REC.
        000150  01 IN-REC.
        000160     02 WEEK      PICTURE 9.
        000170     02 DEPP      PICTURE 9.
        000180     02 TYPE-RUN  PICTURE A(10).
        000190     02 PROG      PICTURE X(4).
        000200     02 DATE      PICTURE X(5).
        000210     02 FILLER    PICTURE X(7).
        000220     02 MINUTES   PICTURE 999V9.
        000230     02 CHARGE    PICTURE 9999V9.
        000231     02 FILLER  PICTURE X(43).
        000240 FD  REP-FILE LABEL RECORDS ARE OMITTED REPORT IS USAGE-REPORT.
        000250 FD  PRNT-FILE LABEL RECORD OMITTED DATA RECORD D-REC.
        000260 01  D-REC PICTURE X(120).
        000270 WORKING-STORAGE SECTION.
        000280 77  MONTH PICTURE X(9).
        000290 77  COUNT PICTURE 9 VALUE 1.
        000300 77  CONT PICTURE X(11).
        000310 77  SAVE-DEP PICTURE 9 VALUE 0.
        000311 77  DEP PICTURE 9.
        000320 01  DEP-NAMES.
        000330     02 FILLER PICTURE A(11) VALUE 'ENGINEERING'.
        000340     02 FILLER PICTURE A(11) VALUE 'SALES'.
        000350     02 FILLER PICTURE A(11) VALUE 'ACCOUNTING'.
        000360 01  D-NAMES REDEFINES DEP-NAMES.
        000370     02 NAME PICTURE A(11) OCCURS 3 TIMES.
        000380 REPORT SECTION.
        000390 RD  USAGE-REPORT CONTROLS ARE FINAL, DEP, WEEK, TYPE-RUN
        000400     PAGE 62 LINES HEADING 1 FIRST DETAIL 1 LAST DETAIL 39
        000410     FOOTING 57.
        000420 01  TYPE REPORT HEADING.
        000430     02 LINE 1 COLUMN 55 PICTURE A(11) VALUE
        000440        'ABC COMPANY'.
        000450     02 LINE 2 COLUMN 45 PICTURE A(25) VALUE
        000460        'COMPUTER USAGE REPORT FOR'.
        000470     02 COLUMN 71 PICTURE X(9) SOURCE MONTH.
        000480 01  PAGE-HEAD TYPE PH NEXT GROUP PLUS 1.
        000490     02 LINE 5 COLUMN 48 PICTURE A(11) JUSTIFIED RIGHT
        000500        SOURCE NAME (DEP).
        000510     02 COLUMN 61 PICTURE A(11) VALUE 'DEPARTMENT'.
        000520     02 COLUMN 72 PICTURE A(11) SOURCE CONT.
        000530     02 LINE PLUS 2 COLUMN 24 PICTURE X(54) VALUE
        000540        'WEEK   TYPE-RUN     PROGRAM     DATE     MINUTES    CHARGE'.
        000550 01  DET-TEST TYPE DETAIL LINE PLUS 1.
        000560     02 COLUMN 26 PICTURE 9 SOURCE WEEK GROUP INDICATE.
        000570     02 COLUMN 31 PICTURE A(10) SOURCE TYPE-RUN GROUP INDICATE.
        000580     02 COLUMN 45 PICTURE X(4) SOURCE PROG.
```

Figure F-1. Report Writer Program



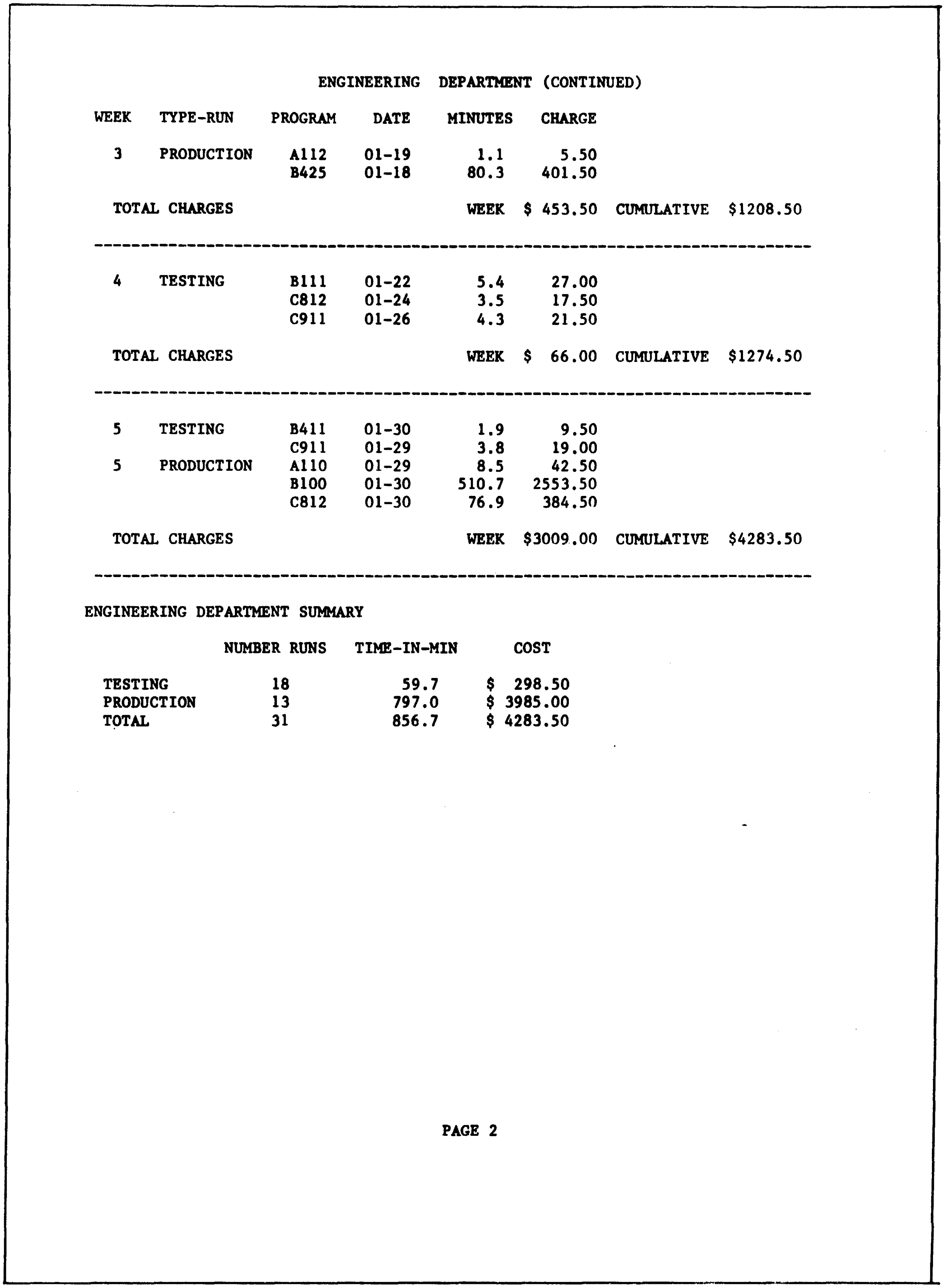
```
000590      02 COLUMN 53 JUSTIFIED RIGHT PICTURE X(5) SOURCE DATE.
000600      02 COLUMN 63 PICTURE ZZ9.9 SOURCE MINUTES.
000610      02 COLUMN 71 PICTURE ZZZZ.99 SOURCE CHARGE.
000620      02 PICTURE 99 SOURCE COUNT.
000630 01   DET-PROD TYPE DETAIL LINE PLUS 1.
000640      02 COLUMN 26 PICTURE 9 SOURCE WEEK GROUP INDICATE.
000650      02 COLUMN 31 PICTURE A(10) SOURCE TYPE-RUN GROUP INDICATE.
000660      02 COLUMN 45 PICTURE X(4) SOURCE PROG.
000670      02 COLUMN 53 JUSTIFIED RIGHT PICTURE X(5) SOURCE DATE.
000680      02 COLUMN 63 PICTURE ZZ9.9 SOURCE MINUTES.
000690      02 COLUMN 71 PICTURE ZZZZ.99 SOURCE CHARGE.
000700      02 PICTURE 99 SOURCE COUNT.
000710 01   TYPE CONTROL FOOTING WEEK NEXT GROUP PLUS 1.
000720      02 LINE PLUS 2 COLUMN 26 PICTURE A(13) VALUE 'TOTAL CHARGES'.
000730      02 COLUMN 64 PICTURE A(4) VALUE 'WEEK'.
000740      02 CHARGES COLUMN 70 PICTURE $ZZZ9.99 SUM CHARGE.
000750      02 COLUMN 80 PICTURE A(10) VALUE 'CUMULATIVE'.
000760      02 COLUMN 92 PICTURE $$$$$.99 SUM CHARGE RESET ON DEP.
000770      02 LINE PLUS 2 COLUMN 24 PICTURE X(77) VALUE ALL '-'.
000780 01   TYPE CF DEP.
000790      02 LINE 46 COLUMN 23 PICTURE A(11) JUSTIFIED RIGHT SOURCE
000800         NAME (DEP).
000810      02 COLUMN 34 PICTURE A(19) VALUE ' DEPARTMENT SUMMARY'.
000820      02 LINE PLUS 2 COLUMN 38 PICTURE A(35) VALUE
000830         'NUMBER RUNS    TIME-IN-MIN      COST'.
000840      02 LINE PLUS 2 COLUMN 25 PICTURE A(7) VALUE 'TESTING'.
000850      02 COLUMN 42 PICTURE Z99
000860         SUM COUNT UPON DET-TEST.
000870      02 COLUMN 54 PICTURE ZZZZZ.9 SUM MINUTES UPON DET-TEST.
000880      02 COLUMN 66 PICTURE $ZZZZZ.99 SUM CHARGE UPON DET-TEST.
000890      02 LINE PLUS 1 COLUMN 25 PICTURE A(10) VALUE 'PRODUCTION'.
000900      02 COLUMN 42 PICTURE ZZ9 SUM COUNT UPON DET-PROD.
000910      02 COLUMN 54 PICTURE ZZZZZ.9 SUM MINUTES UPON DET-PROD.
000920      02 COLUMN 66 PICTURE $ZZZZZ.99  SUM CHARGE UPON DET-PROD.
000930      02 LINE PLUS 1 COLUMN 25 PICTURE A(5) VALUE 'TOTAL'.
000940      02 COLUMN 42 PICTURE ZZ9 SUM COUNT.
000950      02 COLUMN 54 PICTURE ZZZZZ.9 SUM MINUTES.
000960      02 CHARGE-TOT COLUMN 66 PICTURE $ZZZZZ.99 SUM CHARGES.
000970 01   TYPE CF FINAL LINE 56.
000980      02 COLUMN 24 PICTURE A(16) VALUE
000990         'TOTAL CHARGE FOR'.
001000      02 COLUMN 41 PICTURE A(9) SOURCE MONTH.
001010      02 COLUMN 52 PICTURE $ZZZZZZ.99 SUM CHARGE-TOT.
001020 01   TYPE REPORT FOOTING LINE 60 COLUMN 58
001030      PICTURE X(13) VALUE 'END OF REPORT'.
001040 01   TYPE PAGE FOOTING.
001050      02 LINE 58 COLUMN 61 PICTURE A(4) VALUE 'PAGE'.
001060      02 COLUMN 66 PICTURE 9 SOURCE PAGE-COUNTER.
001070 PROCEDURE DIVISION.
001080 DECLARATIVES.
001090 INSERT SECTION. USE BEFORE REPORTING PAGE-HEAD.
001100 PAR.
001110      IF DEP = SAVE-DEP MOVE '(CONTINUED)' TO CONT ELSE
001120      MOVE SPACES TO CONT.  MOVE DEP TO SAVE-DEP.
001130 END DECLARATIVES.
001140 REP SECTION.
001150 OPEN-PAR.
001160      OPEN INPUT IN-FILE OUTPUT REP-FILE PRNT-FILE.
001161      READ IN-FILE AT END GO TO END-PAR. MOVE DEPP TO DEP.
001162      MOVE TYPE-RUN TO MONTH.
001170      MOVE ' BEGIN REPORT-TEST-2 TEST ' TO D-REC WRITE D-REC.
001180      INITIATE USAGE-REPORT.
001190 READ-PAR. READ IN-FILE AT END GO TO END-PAR. MOVE DEPP TO DEP.
001200      IF TYPE-RUN = 'PRODUCTION' GENERATE DET-PROD ELSE
001210      GENERATE DET-TEST.
001220      GO TO READ-PAR.
001230 END-PAR.  TERMINATE USAGE-REPORT.
001240      MOVE ' END OF REPORT-TEST-2 TEST' TO D-REC WRITE D-REC.
001250      CLOSE IN-FILE REP-FILE PRNT-FILE.  STOP RUN.
*** NUMBER OF DIAGNOSTIC MESSAGES     0 ***      HIGHEST SEVERITY LEVEL     0 ***
```

Figure F-1. Report Writer Program (cont.)



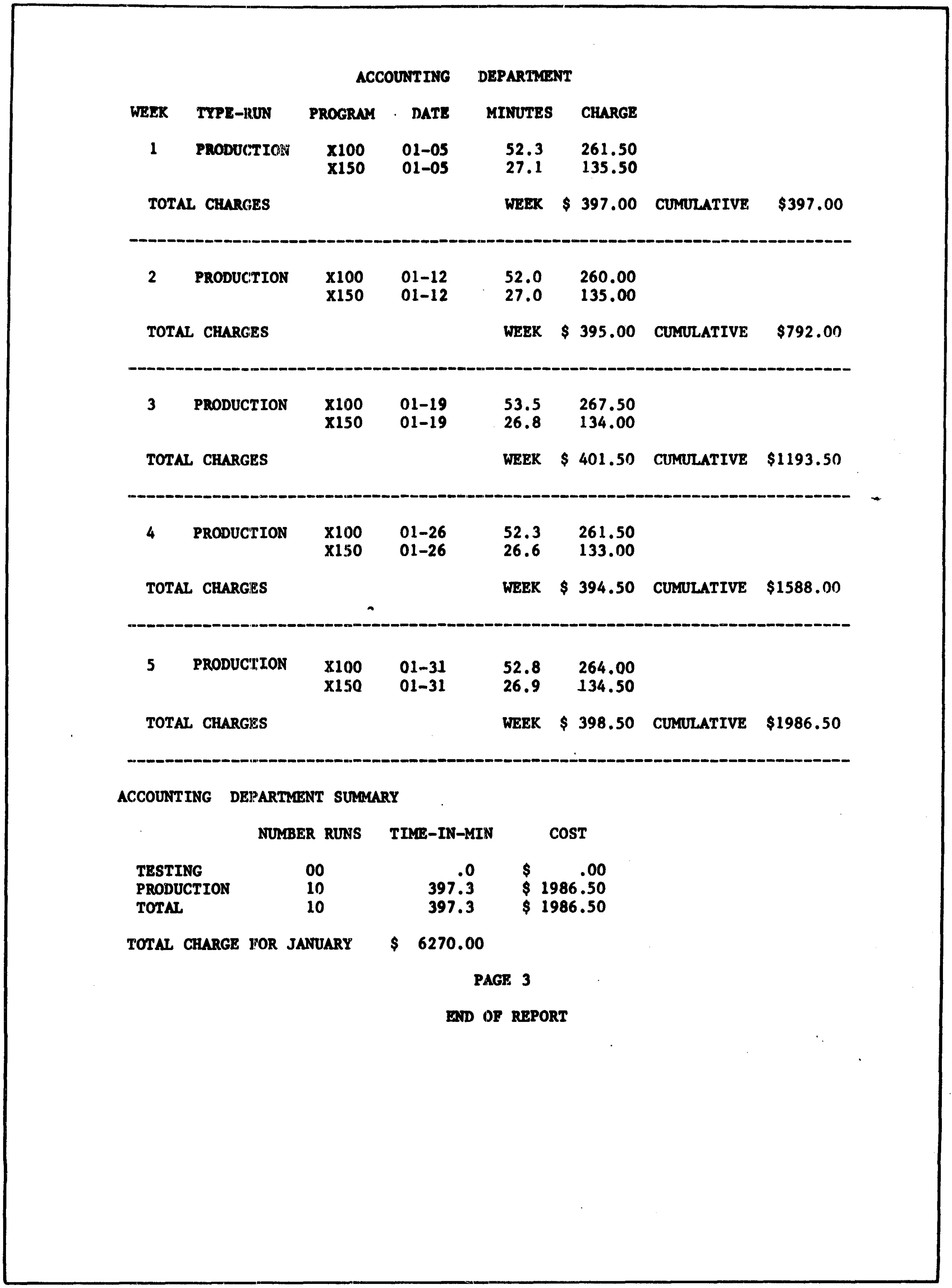
```
                             ABC COMPANY
                  COMPUTER USAGE REPORT FOR JANUARY

                       ENGINEERING  DEPARTMENT

WEEK   TYPE-RUN     PROGRAM    DATE    MINUTES    CHARGE

  1    TESTING        A110    01-03        2.4     12.00
                      A120    01-04       12.8     64.00
                      C612    01-04        2.6     13.00
  1    PRODUCTION     A110    01-05        2.5     12.50
                      C118    01-05       50.7    253.50

 TOTAL CHARGES                            WEEK  $ 355.00  CUMULATIVE    $355.00

------------------------------------------------------------------------------

  2    TESTING        A211    01-08        1.6      8.00
                      A211    01-10        1.6      8.00
                      B111    01-08        5.4     27.00
                      B214    01-11        2.0     10.00
                      B214    01-12        2.0     10.00
                      C812    01-09        3.5     17.50
  2    PRODUCTION     A110    01-11        2.4     12.00
                      A112    01-09        1.1      5.50
                      C118    01-12       20.0    100.00
                      C518    01-08       22.5    112.50
                      C526    01-11       17.9     89.50

 TOTAL CHARGES                            WEEK  $ 400.00  CUMULATIVE    $755.00

------------------------------------------------------------------------------

  3    TESTING        A212    01-18        1.7      8.50
                      B411    01-15        1.9      9.50
                      B411    01-15        1.9      9.50
                      C809    01-19        1.4      7.00
  3    PRODUCTION     A110    01-18        2.4     12.00











                                PAGE 1
```

Figure F-2. Report Writer Generated Report



```
                         ENGINEERING  DEPARTMENT (CONTINUED)

    WEEK   TYPE-RUN     PROGRAM    DATE    MINUTES   CHARGE

      3    PRODUCTION     A112    01-19       1.1       5.50
                          B425    01-18      80.3     401.50

      TOTAL CHARGES                        WEEK  $ 453.50  CUMULATIVE  $1208.50

    ----------------------------------------------------------------------------

      4    TESTING        B111    01-22       5.4      27.00
                          C812    01-24       3.5      17.50
                          C911    01-26       4.3      21.50

      TOTAL CHARGES                        WEEK  $  66.00  CUMULATIVE  $1274.50

    ----------------------------------------------------------------------------

      5    TESTING        B411    01-30       1.9       9.50
                          C911    01-29       3.8      19.00
      5    PRODUCTION     A110    01-29       8.5      42.50
                          B100    01-30     510.7    2553.50
                          C812    01-30      76.9     384.50

      TOTAL CHARGES                        WEEK  $3009.00  CUMULATIVE  $4283.50

    ----------------------------------------------------------------------------

   ENGINEERING DEPARTMENT SUMMARY

                  NUMBER RUNS   TIME-IN-MIN      COST

     TESTING           18             59.7     $  298.50
     PRODUCTION        13            797.0     $ 3985.00
     TOTAL             31            856.7     $ 4283.50




                                      PAGE 2
```

Figure F-2. Report Writer Generated Report (cont.)



```
                            ACCOUNTING   DEPARTMENT

WEEK   TYPE-RUN     PROGRAM    DATE     MINUTES    CHARGE

  1    PRODUCTION     X100     01-05      52.3     261.50
                      X150     01-05      27.1     135.50

  TOTAL CHARGES                          WEEK   $ 397.00  CUMULATIVE    $397.00

--------------------------------------------------------------------------------

  2    PRODUCTION     X100     01-12      52.0     260.00
                      X150     01-12      27.0     135.00

  TOTAL CHARGES                          WEEK   $ 395.00  CUMULATIVE    $792.00

--------------------------------------------------------------------------------

  3    PRODUCTION     X100     01-19      53.5     267.50
                      X150     01-19      26.8     134.00

  TOTAL CHARGES                          WEEK   $ 401.50  CUMULATIVE   $1193.50

--------------------------------------------------------------------------------

  4    PRODUCTION     X100     01-26      52.3     261.50
                      X150     01-26      26.6     133.00

  TOTAL CHARGES                          WEEK   $ 394.50  CUMULATIVE   $1588.00

--------------------------------------------------------------------------------

  5    PRODUCTION     X100     01-31      52.8     264.00
                      X150     01-31      26.9     134.50

  TOTAL CHARGES                          WEEK   $ 398.50  CUMULATIVE   $1986.50

--------------------------------------------------------------------------------
ACCOUNTING  DEPARTMENT  SUMMARY

                NUMBER RUNS   TIME-IN-MIN      COST

  TESTING           00                .0     $     .00
  PRODUCTION        10             397.3     $ 1986.50
  TOTAL             10             397.3     $ 1986.50

 TOTAL CHARGE FOR JANUARY     $  6270.00

                                     PAGE 3

                                  END OF REPORT
```

Figure F-2. Report Writer Generated Report (cont.)

INDEX

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the **most significant page is listed first**. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

□ Learning □ Installing □ Sales

□ Reference □ Maintaining □ Operating

Is the material presented effectively?

□ Fully Covered □ Well Illustrated □ Well organized □ Clear

What is your overall rating of this publication?

□ Very Good □ Fair □ Very Poor

□ Good □ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your name & Return Address

Thank You For Your Interest (fold & fasten as shown on back, no postage needed if mailed in U S A)

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

CUT ALONG LINE

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 59153 LOS ANGELES,CA 90045

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
5250 W. CENTURY BOULEVARD
LOS ANGELES, CA 90045

ATTN: PROGRAMMING PUBLICATIONS

FOLD ALONG LINE

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

24779, 5C1079, Printed in U.S.A.