Burroughs

LANGUAGE MANUAL

B 1000 Systems COBOL74

(Relative to Mark 11.0 System Software Release)

Priced Item
Printed in U.S.A
August 1984

1168622

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to Corporate Documentation-West, Burroughs Corporation, 1300 John Reed Court, City of Industry, California 91745, U.S.A.

LIST OF EFFECTIVE PAGES

Page	Issue
Title	Original
ii	Original
iii	Original
iv	Blank
v thru xviii	Original
xix	Original
xx	Blank
xxi thru xxiii	Original
xxiv	Blank
1-1 thru 1-2	Original
2-1 thru 2-23	Original
2-24	Blank
3-1 thru 3-5	Original
3-6	Blank
4-1 thru 4-3	Original
4-4	Blank
5-1 thru 5-27	Original
5-28	Blank
6-1 thru 6-77	Original
6-78	Blank
7-1 thru 7-153	Original
7-154	Blank
8-1 thru 8-16	Original
9-1 thru 9-24	Original
10-1 thru 10-14	Original
11-1 thru 11-8	Original
A-1 thru A-9	Original
A-10	Blank
B-1 thru B-38	Original
C-1 thru C-13	Original
C-14	Blank
D-1 thru D-21	Original
D-22	Blank
E-1 thru E-71	Original
E-72	Blank
F-1 thru F-16	Original
G-1 thru G-19	Original
G-20	Blank
1 thru 13	Original
14	Blank

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

FOREWORD

BURROUGHS EXTENSIONS TO ANSI 74 COBOL

Programming applications are written in the COBOL74 language as specified in this B 1000 Systems COBOL74 Reference Manual. The source language herein described is the USA Standard COBOL, X3.23-1974, which implements the lowest defined level of the Report Writer Module, and also the highest defined level of these Modules: Nucleus, Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Segmentation, Library, Debug, Inter-Program Communication, and Communication. To this base Burroughs extensions have been added and are shaded throughout the manual for easier recognition.

ACKNOWLEDGEMENT

INTRODUCTION

This manual provides a complete description of COBOL74 (COmmon Business Oriented Language) as implemented for use on the Burroughs B 1000 System. This concept of COBOL74 is designed along the guidelines of the American National Standards Institute (ANSI) 1974. This edition contains changes throughout.

COBOL74 ADVANTAGES

The long list of COBOL74 advantages is derived chiefly from its intrinsic quality of permitting the programmer to state the problem solution in English prose, and thus provide automatic program and system documentation. When users adopt in-house standardization of elements within files plus well-chosen data-names before attempting to program a system, maximum documentational advantages of the language described herein are obtained.

To a computer user, Burroughs COBOL74 offers the following major advantages:

1. Expeditious means of program implementation.
2. Accelerated programmer training and simplified retraining requirements.
3. Reduced conversion costs when changing from a computer of one manufacturer to that of another.
4. Significant ease of program modification.
5. Standardized documentation.
6. Documentation which facilitates nontechnical management participation in data processing activities.
7. Efficient object program code.
8. Segmentation capability which sets the maximum allowable program size well in excess of any practical requirement.
9. Because of the incorporation of debugging language statements, a high degree of sophistication in program design is achieved.
10. A comprehensive source program diagnostic capability.

COBOL74 CONCEPTS

A program written in COBOL74, called a source program, is accepted as input by the COBOL74 compiler. The compiler verifies that each source statement is syntactically correct, and then converts them into COBOL74 S-code.

The executable program can then be executed on the B 1000 System using the COBOL74 interpreter. The interpreter causes the system hardware to perform the operations specified by the S-code and thus the source program.

The B 1000 COBOL74 compiler operates under the control of the Master Control Program (MCP). Similarly, the S-code generated by the compiler is executed under control of the MCP.

A COBOL program that was compiled with the ANSI 68 COBOL compiler must be recompiled with the COBOL74 compiler in order to run with the COBOL74 interpreter.

ORGANIZATION

This manual consists of 11 sections and 7 appendices:

Section	Contents
1	PROGRAM ORGANIZATION Introduces the four divisions of a COBOL source program and describes the major functions of each.
2	LANGUAGE CONCEPTS The rules for creating a COBOL74 source program are defined in this section.
3	CODING FORM The standard format of the COBOL74 coding form and the rules for spacing are described in this section.
4	IDENTIFICATION DIVISION The structure of the IDENTIFICATION DIVISION and the rules for coding are given.
5	ENVIRONMENT DIVISION The structure of the ENVIRONMENT DIVISION and the rules for coding are given.
6	DATA DIVISION The four sections of the DATA DIVISION are described.
7	PROCEDURE DIVISION The rules for coding and structuring the PROCEDURE DIVISION are given.
8	FILE ATTRIBUTES The file attribute names and the rules for changing attributes are given.
9	DATA BASE MANAGEMENT This section contains the verbs and constructs of COBOL74 that are available for interfacing with DMSII.
10	DEBUG Contains an explanation of the debug facilities available.
11	COBOL74 COMPILER CONTROL Compiler options which are available in the COBOL74 compiler are explained.

RELATED DOCUMENTS

The following documents are referenced in this document:

B 1000 Systems System Software Operation Guide, Volume 1, form number 1151982.

B 1000 Systems System Software Operation Guide, Volume 2, form number 1152097.

B 1000 Systems Data Management System II (DMSII) Reference Manual, form number 1152089.

B 1000 Systems Burroughs Network Architecture (BNA) Installation and Operation Manual, form number 1151974.

B 1000 Systems Network Definition Language (NDL) Reference Manual, form number 1152014.

B 1000 Systems SMCS Installation, Operation and Functional Description Manual, form number 1152279.

SECTION 1
PROGRAM ORGANIZATION

COBOL74 SOURCE PROGRAM DIVISIONS

Every COBOL74 source program must contain these four divisions in the following order:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

The IDENTIFICATION DIVISION identifies the program. In addition to required information, the programmer may include such optional pieces of information as the date compiled and programmer's name for documentation purposes. This division is completely machine-independent and does not produce object code.

The ENVIRONMENT DIVISION specifies the equipment being used. It contains computer descriptions and some information about the files the program will use.

The DATA DIVISION contains not only file and record descriptions describing the data files that the object program manipulates or creates, but also the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. Therefore, this division is to a large extent, computer-independent. While compatibility among computers cannot be absolutely assured, careful planning in the data layout will permit the same data descriptions, with minor modification, to apply to more than one computer.

The PROCEDURE DIVISION specifies user-supplied steps for computer execution. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This division of a COBOL74 program is often referred to as the "program." In reality, it is only part of the total program, and alone is insufficient to describe the entire program. This is true because repeated references must be made (either explicitly or implicitly) to information appearing in the other divisions. This division, more than any other, allows the user to express thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe procedures are basic, as is the use of conditional statements to provide alternative paths of action.

REQUIRED HEADERS

The standard for COBOL74 requires that a program consist of certain divisions, sections, and fixed paragraph names known as headers.

The following elements are the minimum required for a COBOL74 program:

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  MINIMUM.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  SOURCE-COMPUTER.  B-1000.
  OBJECT-COMPUTER.  B-1000.
DATA DIVISION.
PROCEDURE DIVISION.
PARAGRAPH-NAME.
    STOP RUN.
```

SECTION 2
LANGUAGE CONCEPTS

GENERAL

As stated in section 1, COBOL74 is a language based on English and is composed of words, statements, sentences, and paragraphs. The following paragraphs define the rules to be followed in the creation of this language. The use of the different constructs formed from the created words is covered in subsequent sections of this document.

LANGUAGE DESCRIPTION NOTATION

A nearly universal form of notation exists for COBOL reference manuals. This manual uses that notation as described in the paragraphs that follow.

The apostrophe (') is used to delimit characters with specific meanings. Other than its use in this manual as a delimiter, it has no specific use in the COBOL language.

Key Words

All underlined upper-case words are key words and are required when utilizing related functions. Omissions of key words will cause error conditions at compilation time. An example of key words follows:

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

The key words are IF, NOT, NUMERIC, and ALPHABETIC.

Optional Words

All upper-case words not underlined are optional words included for readability only and may be included or excluded in the source program. In the preceding example, the optional word is IS.

Generic Terms

All lower-case words represent generic terms which are used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied in that format position by the programmer. Where generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion. Identifier-1 and identifier-2 are generic terms in the following example:

```
MOVE identifier-1 TO identifier-2
```

Braces

The following symbols are braces: { }. When words or phrases are enclosed in braces, a choice of one of the entries must be made. In the previous example in the subsection titled Key Words, either NUMERIC or ALPHABETIC must be included in the statement.

Brackets

The following symbols are brackets: [ ]. Words and phrases enclosed in brackets represent optional portions of a statement. A programmer wishing to include the optional feature may do so by including the entry shown between brackets. Otherwise, the optional portion may be omitted. ([NOT] in the example titled Key Words, is optional.)

Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL74 program. In this document, the form 01, 02, ... , 09 is used to indicate level-numbers 1 through 9.

Ellipsis

The presence of the ellipsis (three consecutive periods (...)) within any format indicates the position at which repetition may occur at the programmer's option. The portion of the format that may be repeated is defined in the following paragraph.

The ellipsis applies to the words between the determined pair of of delimiters. Given the ellipsis in a clause or statement format, scanning right to left, determine the right bracket or right brace immediately to the left of the ...; continue scanning right to left and determine the logically matching left bracket or left brace.

Format Punctuation

The separators comma and semicolon are used to improve the readability of the program. Suggested uses are shown in General Format subsections throughout this manual, however, use of these separators is optional. In the source program, the comma, semicolon, and space separators are interchangeable. If desired, a semicolon or comma may be used between statements in the PROCEDURE DIVISION.

Paragraphs within the IDENTIFICATION and PROCEDURE DIVISIONS and entries within the ENVIRONMENT and DATA DIVISIONS must be terminated by the separator period. When a single period is shown in a format, it must appear in the same position whenever the source program calls for the use of that particular statement.

Special Characters in Formats

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

CHARACTER SET

The COBOL74 character set for the B 1000 System consists of the following 52 characters:

0	through 9	.	period or decimal point
A	through Z	;	semicolon
	blank or space	"	quotation mark
+	plus sign	(	left parenthesis
–	minus sign or hyphen	)	right parenthesis
*	asterisk	>	greater than symbol
/	slash	<	less than symbol
=	equal sign	@	"at" sign
$	currency sign	,	comma

Characters Used for Words

The character set for words consists of the following 37 characters:

0 through 9

A through Z

– (hyphen)

Punctuation Characters

The following characters may be used for program punctuation:

@	"at" sign		space or blank
"	quotation mark	.	period
(	left parenthesis	,	comma (see following note)
)	right parenthesis	;	semicolon

NOTE

For enchanced readability of the source program, commas may be used between statements, at the programmer's discretion. Use of commas implies that any succeeding statement is to be included as an element of the prior statement.

Editing Characters

The COBOL74 compiler accepts the following characters in editing:

$	currency sign	+	plus
*	asterisk (check protect)	–	minus
,	comma	CR	credit
/	slash	DB	debit
B	space or blank insert	Z	zero suppress
0	zero insert	.	period .

Characters Used in Arithmetic Expressions

The COBOL74 compiler accepts the following characters in arithmetic expressions:

+	addition	**	exponentiation
–	subtraction	(	left parenthesis
*	multiplication	)	right parenthesis
/	division		

Characters Used in Relation Conditions

The COBOL74 compiler accepts the following characters in relation conditions:

=	equal sign
<	less than symbol
>	greater than symbol

LANGUAGE STRUCTURE

The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

Separators

A separator is a string of one or more punctuation characters. The rules for formation of separators are:

1. The punctuation character space is a separator. Anywhere a space is used as a separator, more than one space may be used.

2. The punctuation characters comma, semicolon, and period are separators.

3. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by one of the separators space, comma, semicolon, or left parenthesis; a closing quotation mark must be immediately followed by one of the separators space, comma, semicolon, period, or right parenthesis.

   Quotation marks may appear only in balanced pairs delimiting nonnumeric literals except when the literal is continued.

4. The punctuation characters right and left parentheses are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, indices, arithmetic expressions, or conditions.

5. Pseudo-text delimiters are separators. An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon, or period.

   Pseudo-text delimiters (–) may appear only in balanced pairs delimiting pseudo-text.

6. The punctuation character @ is a separator. An opening @ character must be preceded immediately by one of the separators space, comma, semicolon, or left parenthesis; a closing @ character must be immediately followed by one of the separators space, comma, semicolon, period, or right parenthesis.

   At signs (@) may appear only in balanced pairs delimiting hexadecimal literals.

7. The separator space may optionally immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

   Any punctuation character which appears as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

   The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment-entries, or comment lines.

Character-Strings

A character-string is a character or sequence of contiguous characters which forms a COBOL74 word, literal, PICTURE character-string, or comment-entry. A character-string is delimited by separators.

DEFINITION OF WORDS

A COBOL74 word is created from a combination of not more than 30 characters, selected from the following:

A through Z

0 through 9

– hyphen

A word is ended by a space, period, comma, or semicolon. A word may not begin or end with a hyphen. (A literal constitutes an exception to these rules, as explained in a paragraph entitled Literals in this section.)

A user-defined word is a COBOL74 word that must be supplied by the user to satisfy the format of a clause or statement.

Types of Words

COBOL74 contains the following word types: nouns (user-defined words), verbs, and reserved words.

Nouns

Nouns are divided into special categories:

File-name	Family-name
Record-name	Cd-name
Data-name	Text-name
Condition-name	Library-name
Mnemonic-name	Program-name
Index-name	Alphabet-name
Paragraph-name	Section-name

The length of a noun must not exceed 30 characters. For purposes of readability, a noun may contain one or more hyphens. However, the hyphen must neither begin nor end the noun (this does not apply to literals).

All nouns within a given category must be unique, either because no other noun in the same source program has identical spelling or punctuation, or because uniqueness can be insured by qualification. With the exception of paragraph-name, section-name, text-name, library-name, and family-name, all user-defined words must contain at least one alphabetic character.

File-Name

A file-name is a noun containing at least one alphabetic character assigned to designate a set of data items. The contents of a file are divided into logical records made up of any consecutive set of data items.

Record-Name

A record-name is a noun containing at least one alphabetic character assigned to identify a logical record. A record can be subdivided into several data items, each distinguishable by a data-name.

Data-Name

A data-name is a noun assigned to identify elements within a record or work area and is used in COBOL74 to refer to an element of data, or to a defined data area containing data elements. Each data-name must contain at least one alphabetical character.

Condition-Name

A condition-name is the name assigned to a specific value, set of values, or range of values within the complete set of values that a data item may assume. The data item is a conditional variable. The condition-name must contain at least one alphabetic character and must be unique, or be able to be referenced uniquely through qualification. A conditional variable may be used as a qualifier for any of its condition-names. If references to a conditional variable require indexing, subscripting, or qualification, then references to any of its condition-names also require the same combination of indexing, subscripting, or qualification. A condition-name is used in conditions as an abbreviation for the relation condition; its value is TRUE if the associated conditional variable is equal to one of the set values to which that condition-name is assigned.

Condition-names may be defined in the DATA DIVISION, or in a SPECIAL-NAMES paragraph within the ENVIRONMENT DIVISION where a condition-name must be assigned to the ON STATUS or OFF STATUS, or both, of defined switches.

Mnemonic-Name

The use of mnemonic-names provides a means of relating certain hardware equipment names to problem-oriented names the programmer may wish to use. These associations are established in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

Index-Name

An index-name is a word with at least one alphabetic character that names an index associated with a specific table (refer to Indexing in this section). An index is a register, the contents of which represent the ~~character~~ position of the first character of an element of a table with respect to the beginning of the table.

computational digit

Paragraph-Name

A paragraph-name is a word which names a paragraph in the PROCEDURE DIVISION. Paragraph-names are equivalent only if composed of the same sequence of the same number of digits and/or characters.

Section-Name

A section-name is a word which names a section in the PROCEDURE DIVISION. Section-names are equivalent only if composed of the same sequence of the same number of digits and/or characters.

Other Categories

See the glossary in appendix D for definitions of all other types of user-defined words.

Verbs

A verb in COBOL74 is a single word that denotes action, such as ADD, WRITE, or MOVE. All allowable verbs in COBOL74, with the exception of the word IF, are English verbs. The usage of the COBOL74 verbs takes place primarily within the PROCEDURE DIVISION.

Reserved Words

A reserved word is a COBOL74 word that is one of a specified list of words which may be used in COBOL74 source programs, but must not appear in the programs as user-defined words. Refer to appendix A, Reserved Words.

These rules apply to the entire COBOL74 source program; no exceptions exist for specific divisions, sections, or statements.

There are six types of reserved words:

Key words
Connectives
Optional words
Figurative constants
Special registers
Special-character words

Key Words

A key word is a word whose presence is required in a source program. Within each format, such words are upper-case and underlined.

Key words are of three types:

1. Verbs such as ADD and READ.
2. Required words which appear in statement and entry formats.
3. Words which have a specific functional meaning such as NEGATIVE and SECTION.

Connectives

Connectives are used to indicate the presence of a qualifier or to form compound conditional statements. The connectives OF and IN are used for qualification. The connectives AND, AND NOT, OR, or NOT are used as logical connectives in conditional statements. The comma is used as a series connective to separate two or more operands.

Optional Words

Optional words are included in the COBOL74 language to improve the readability of the statement formats. These optional words may be included or omitted. For example, IF A IS GREATER THAN B... is equivalent to IF A GREATER B... ; the inclusion or omission of the words IS and THEN does not influence the logic of the statement.

Figurative Constant

A figurative constant is a reserved word used to reference specific constant values and must never be enclosed in quotation marks except when the word, rather than the value, is desired. The figurative constant names and meanings are:

ZERO ZEROS ZEROES	Represents the value 0, or one or more of the character '0', depending on the context.
SPACE SPACES	Represents one or more spaces (blanks).
HIGH-VALUE HIGH-VALUES	Represents one or more occurrences of the character that has the highest ordinal position in the program collating sequence, except in the alphabet-name clause of the SPECIAL-NAMES paragraph, where it represents the highest ordinal position in the native collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more occurrences of the character that has the lowest ordinal position in the program collating sequence, except in the alphabet-name clause of the SPECIAL-NAMES paragraph, where it represents the lowest ordinal position in the native collating sequence.
QUOTE QUOTES	Represents one or more occurrences of the character '"'. The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD". If, however, the full "ABD" is desired in a DISPLAY statement, it can be achieved by writing QUOTE "ABD" QUOTE, in which case the object program will display "ABD".
ALL<literal>	When followed by a hexadecimal literal, a nonnumeric literal, or a figurative constant, the word ALL represents a series of that literal. For example, if the COBOL74 statement is MOVE ALL literal TO ERROR-CODE, then the resultant ERROR-CODE would take on the following values:

ALL literal	Size of ERROR-CODE	ERROR-CODE
ALL "ABC"	7 characters	ABCABCA
ALL "2" or ALL 2	5 characters	22222
ALL QUOTE	3 characters	"""
ALL SPACES	8 characters	(eight spaces)

NOTE

The use of ALL with figurative constants, as illustrated in the last two instances, is redundant. MOVE ALL SPACES and MOVE SPACES yields the same result.

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

1. When a figurative constant is associated with another data item, 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. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, STRING, STOP, or UNSTRING statement, the length of the string is one character.

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to numeric characters only, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

When the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual character associated with each figurative constant depends upon the program collating sequence specified. Refer to OBJECT-COMPUTER and SPECIAL-NAMES in Section 5 for additional information.

Special Registers

Certain reserved words are used to name and reference special registers. Special registers are certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL74 features. These special registers include the following: LINAGE-COUNTER, LINE-COUNTER, PAGE-COUNTER, and DEBUG-ITEM.

Special-Character Words

The arithmetic operators and relation characters are reserved words. Refer to the glossary in appendix D for additional information.

Literals

A literal is an item of data whose value is implied by an ordered set of characters of which the literal is composed, or by specification of a reserved word which references a figurative constant. There are three classes of a literal: numeric, nonnumeric, and hexadecimal.

Numeric Literal

A numeric literal is a character-string whose characters are selected from the digits 0 through 9, the plus sign (+), the minus sign (-), and/or the decimal point. Numeric literals may be from 1 to 18 digits in length. The rules for the formation of numeric literals are as follows:

1. A numeric literal must contain at least one digit.

2. A numeric literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

3. A numeric literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer. An integer is a numeric literal which contains no decimal point.

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

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal belongs to category numeric. Refer to the PICTURE clause in section 6 for additional information. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user. The following are examples of numeric literals:

   51679
   .005
   +2.629
   -.8479
   6287.92

Nonnumeric Literal

A nonnumeric literal may be composed of any allowable character. The beginning and ending of a nonnumeric literal are both denoted by a quotation mark. Any character enclosed within quotation marks is part of the nonnumeric literal. Subsequently, all spaces enclosed within the quotation marks are considered part of the literal. Two consecutive quotation marks within a nonnumeric literal cause a single quotation mark to be inserted into the literal string. Four consecutive quotation marks result in a single " literal.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals belong to category alphanumeric. Refer to the PICTURE clause in section 6.

A nonnumeric literal cannot exceed 160 characters. Examples of nonnumeric literals are:

Literal on Source Program Level	Literal Stored by Compiler
"THE TOTAL PRICE"	THE TOTAL PRICE
"-2080.479"	-2080.479
"""LIMITATIONS"""	"LIMITATIONS"
""""	"
"A""B"	A"B

NOTE

Literals that are used for arithmetic computation must be expressed as numeric literals and must not be enclosed in quotation marks as nonnumeric literals. For example, "4.4" and 4.4 are not equivalent. The compiler stores the nonnumeric literal as 4.4, whereas the numeric literal would be stored as 0044 if the PICTURE were 999V9 DISPLAY, with the assumed decimal point located between the two fours.

Hexadecimal Literals

A hexadecimal literal is a character-string consisting of characters selected from the hexadecimal digits '0' through '9' and 'A' through 'F'. The beginning and ending of a hexadecimal literal are each denoted by an @ sign. For example, a binary 12 would be expressed @C@.

The category of a hexadecimal literal (4-bit numeric or 8-bit alphanumeric) is determined by the category of the data item with which it is associated in a COBOL74 statement. A hexadecimal literal is handled as a 4-bit numeric when the category of the associated data item is numeric whether USAGE is COMPUTATIONAL or DISPLAY.

A hexadecimal literal is handled as if it were numeric if:

1. In the VALUE clause, the category of the associated data item is numeric.
2. In the MOVE statement, the category of the receiving data item is numeric or numeric edited.
3. In the conditional expression of an IF, PERFORM, or SEARCH statement, the category of the other relational operand is numeric.

A hexadecimal literal is handled as 8-bit alphanumeric when the category of the associated data item is nonnumeric. Each character is represented by two hexadecimal digits. This requires an even number of digits in the hexadecimal literal. A hexadecimal literal is handled as if it were alphanumeric if:

1. In the VALUE clause, the category of the associated data item is not numeric.
2. In the MOVE statement, the category of the receiving data item is alphanumeric, alphabetic, or alphanumeric edited.
3. In the conditional expression of an IF, PERFORM, or SEARCH statement, the category of the other relational operand is not numeric.
4. It appears in an INSPECT, STRING, UNSTRING, DISPLAY, STOP, DISABLE, or ENABLE statement.
5. It appears in the ALL figurative constant.

A hexadecimal literal may also appear in a COPY statement, in which case the hexadecimal literal does not have a type associated with it.

The following restrictions apply to hexadecimal literals:

1. A hexadecimal literal is not allowed as an arithmetic operand in an ADD, SUBTRACT, MULTIPLY, or DIVIDE statement, nor in an arithmetic expression in a COMPUTE statement or conditional expression.

2. A hexadecimal literal is not allowed as a subscript or index.

3. A hexadecimal literal is not allowed as a program name in a CALL or CANCEL statement.

4. An identifier assigned a hexadecimal literal will not, in most cases, compare as either numeric or alphabetic in a class condition test.

5. When a hexadecimal literal is handled as if its category were computational, then the length of the literal must be from 1 to 18 digits. When a hexadecimal literal is handled as if it were nonnumeric, the length of the literal must be from 2 to 320 digits.

LOGICAL RECORD AND FILE CONCEPTS

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

Physical Aspects of a File

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

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

Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL74 program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL74 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 are adapted 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 has no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; several logical records may be contained within a single physical unit; or, in the case of mass storage files, a logical record may require more than one physical unit. There are several source language methods available for describing the relationship of logical records and physical units. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit must be provided by the interaction of the object program on the hardware and/or software system. In this manual, references to records indicate records, unless the phrase 'physical record' is specifically used.

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

Record Concepts

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

DATA DESCRIPTION CONCEPTS

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 has been specified, it may be further subdivided to permit more detailed data referral.

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

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups. 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 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers 66, 77, and 88, which are exceptions to this rule. 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. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item.

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

1. Entries that specify elementary items or groups introduced by a RENAMES clause.
2. Entries that specify noncontiguous working storage and linkage data items.
3. Entries that specify condition-names.

Entries describing items by means of RENAMES clauses for the purpose of regrouping data items have been assigned the special level-number 66.

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

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

CONCEPT OF CLASSES OF DATA

The five categories of data items (refer to the PICTURE clause in section 6) are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (without editing). Every elementary item, except for an index data item, belongs to one of the classes and also to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to that group item. Table 2-1 shows the relationship of the class and categories of data items.

Table 2-1. Classes of Data

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric

ALGEBRAIC SIGNS

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items and signed numeric literals to indicate algebraic properties; and editing signs, which appear on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly the location of the operational sign. The clause is optional; if it is not used, operational signs are represented as defined under symbol 'S' of the PICTURE clause. Refer to the PICTURE clause, General Rule 8, the 'S' symbol in section 6.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

DATA DESCRIPTION CONCEPTS

STANDARD ALIGNMENT RULES

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

   a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

   b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following the rightmost character and is aligned as in step 1a above.

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

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

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in the JUSTIFIED clause description in section 6.

UNIQUENESS OF REFERENCE

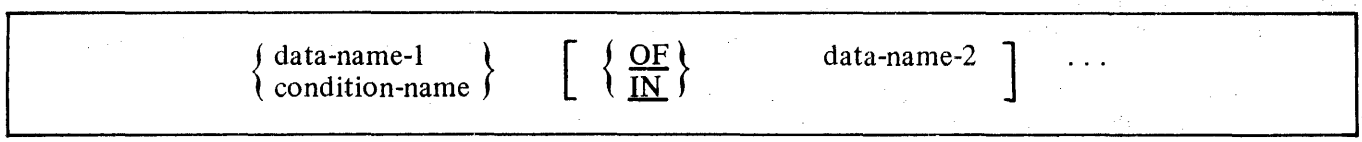
Uniqueness of reference for identifiers and condition-names, if not unique in the program, can be accomplished through the use of qualification, subscripting, or indexing.

Identifier

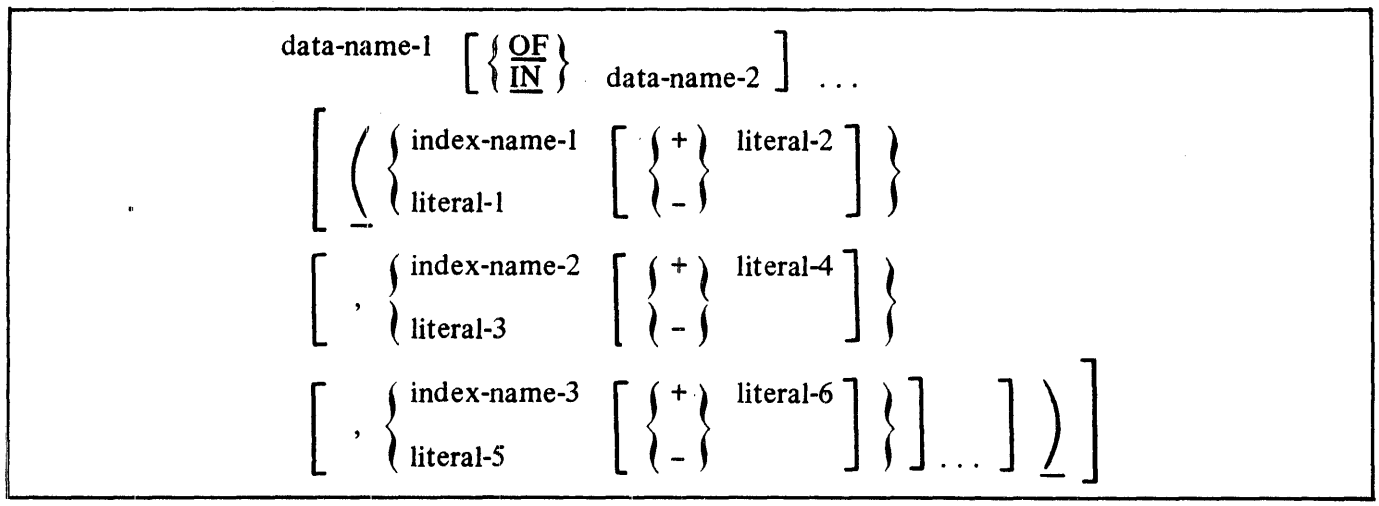
An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or indices necessary to ensure uniqueness.

General Formats:

Format 1:



Format 2:



Restrictions on qualification, subscripting, and indexing are:

1. A data-name must not be subscripted or indexed when that data-name is being used as an index, subscript, or qualifier.

2. Indexing is not permitted where subscripting is not permitted.

3. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data. Refer to the USAGE clause in section 6. Such data items are called index data items.

4. Literal-1, literal-3, literal-5, ... in the previous format example, must be positive numeric integers. Literal-2, literal-4, literal-6, ... must be unsigned numeric integers.

Condition-Name

Each condition-name must be unique, or made unique through qualification and/or indexing, or subscripting.

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

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

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names is exactly that of 'identifier' except that data-name-1 is replaced by 'condition-name-1'.

In the general formats, 'condition-name' refers to a condition-name qualified, indexed or subscripted, as necessary.

Qualification

Every user-specified name that defines an element in a COBOL74 source program must be unique, either because no other name has the identical spelling and hyphenation, or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and the process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the DATA DIVISION, all data-names used for qualification must be associated with a level indicator or a level-number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification. In the PROCEDURE DIVISION, two identical paragraph-names must not appear in the same section.

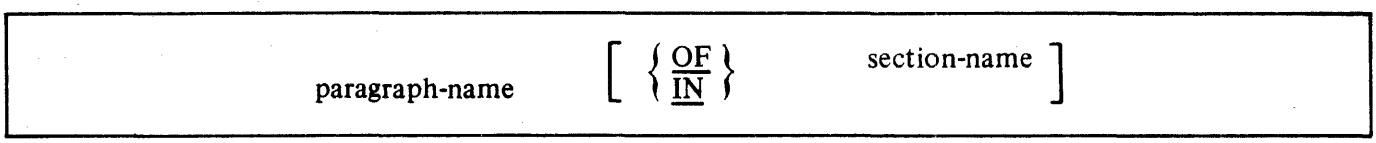
In the hierarchy of qualification, names associated with a level indicator are the most significant, followed by those names associated with level-number 01, and finally the names associated with level-number 02, ... , 49. A section-name is the highest and only qualifier available for a paragraph-name. The most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and procedure-name.

Qualification is performed by following a data-name, a condition-name, a paragraph-name, or a text-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

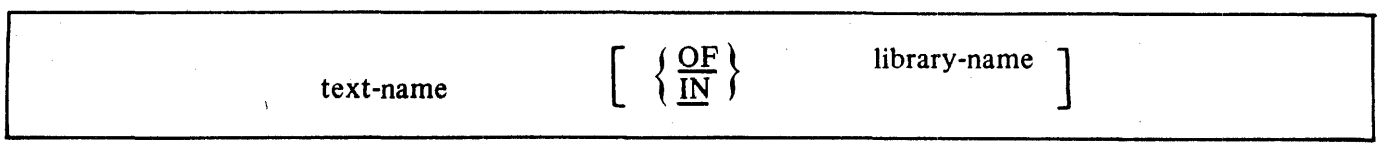
General Format:

Format 1:

Format 2:



Format 3:



QUALIFICATION

General Rules:

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

2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referenced in the PROCEDURE, ENVIRONMENT, and DATA DIVISIONS (except in the REDEFINES clause where qualification is unnecessary and must not be used.)

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

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

6. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used. The complete set of qualifiers for a data-name must not be the same as any partial set of qualifiers for another data-name. Qualified data-names may have any number of qualifiers up to and including 49.

7. If more than one COBOL74 library is available to the compiler during compilation, text-name must be qualified each time it is referenced.

Examples:

In the following file descriptions all items are unique except the data-name TECH. In order to refer to either TECH item, qualification must be used. Otherwise, if reference is made to TECH only, the compiler would not know which of the two is desired. Therefore, in order to move the contents of one TECH into the other TECH, the PROCEDURE DIVISION must be coded with one of the following sentences:

```
MOVE TECH IN CITY-NO TO TECH OF STATE-NO.

MOVE TECH OF CITY-NO TO TECH IN STATE-NO.

MOVE TECH IN AREA-NO TO TECH OF RADIUS-NO.

MOVE TECH OF AREA-NO TO TECH IN RADIUS-NO.
```

```
01  AREA-NO . . .
    03  CITY-NO . . .
        05  TECH . . .
        05  BRANCH . . .
    03  DISTRICT . . .

01  RADIUS-NO . . .
    03  STATE-NO . . .
        05  TECH . . .
        05  DIST-BR . . .
    03  REGION . . .
```

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names (refer to the OCCURS clause in section 6).

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

The subscript may be signed and, if signed, must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, and so forth. The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

At the time of execution of a statement which refers to a subscripted table element, each subscript specified is validated. That is, its value must not be less than one or more than the maximum number of occurrences as specified by the corresponding OCCURS clause (as modified by the DEPENDING ON clause, if any). If the subscript value is not within this range, an abnormal termination of the program occurs.

The subscript or set of subscripts that identifies the table element is delimited by the balanced pair of separators, left parenthesis and right parenthesis, following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier.

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

General Format:

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

Example:

In the following file description, to reference the first department, DEPT (1) is written. If data-name X contains the number of the department desired, DEPT (X) is written. If the data item GROUP contains the specific group desired, then POSITION (X, GROUP) would reference the exact employee.

```
01  EMPLOYEE-JOBS.
    05  DEPT                   OCCURS 50 TIMES.
        10  DEPT-NAME                          PIC X(10).
        10  ALL-JOBS           OCCURS  20 TIMES.
            15  POSITION                       PIC X(15).
```

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 phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name can be given a value by the execution of a SET statement, a SEARCH ALL statement, or a Format 4 PERFORM statement.

An index-name has the same internal representation as an index data item. Refer to General Rule 9, the USAGE clause, in section 6. If a value to be stored in an index-name or in an index data name exceeds the largest value that can be held in that index-name or index data name, the value is truncated according to the rules for the occurrence of a size error condition in an arithmetic statement without a SIZE ERROR phrase.

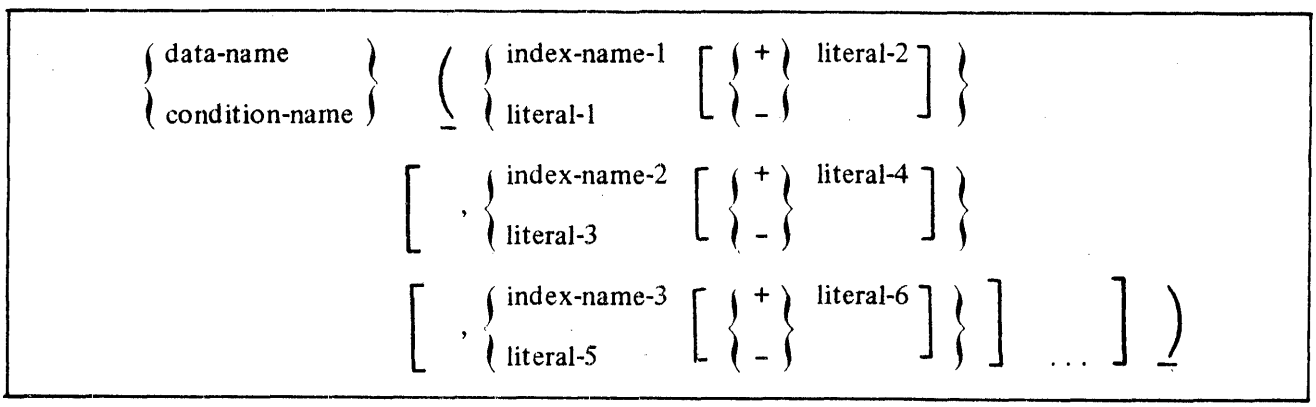
An index-name assigned to one table may not be used to index another table.

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 integer numeric literal, all of which is delimited by the matching pair of separators, left parenthesis and right parenthesis, following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (when the operator − is used), by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

At the time of execution of a statement which refers to an indexed table element, the value of each direct or relative index must not be less than a value which corresponds to the beginning of the first occurrence of the table element. Also, the index must not be greater than a value which corresponds to the beginning of the last occurrence of the table element as specified by the corresponding OCCURS clause. If the index value is not within this range, the execution of the program is terminated. The index value need not precisely address the beginning of a table element in order to pass the range check. This may occur when an index-name is set to the value of an index data item which has been set to the value of another index-name, as such assignments are made without conversion.

Subscripting is permitted where indexing is permitted.

General Format:



EXPLICIT AND IMPLICIT SPECIFICATIONS

There are three types of explicit and implicit specifications that occur in COBOL74 source programs:

1. Explicit and implicit PROCEDURE DIVISION references.
2. Explicit and implicit transfers of control.
3. Explicit and implicit attributes.

Explicit and Implicit PROCEDURE DIVISION References

A COBOL74 source program can reference data items either explicitly or implicitly in PROCEDURE DIVISION statements. An explicit reference occurs when the name of the referenced item is written in a PROCEDURE DIVISION statement or when the name of the referenced item is copied into the PROCEDURE DIVISION by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a PROCEDURE DIVISION statement without the name of the referenced item being written in the source statement. An implicit reference also occurs, during the execution of a PERFORM statement, when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER, or UNTIL phrase is initialized, modified, or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs if the data item contributes to the execution of the statement.

Explicit and Implicit Transfers of Control

The mechanism that controls program flow transfers control from statement to statement in the sequence in which the statements were written in the source program, unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without the writing of an explicit PROCEDURE DIVISION statement, and therefore, is an implicit transfer of control.

COBOL74 provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL74 provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

1. If a paragraph is being executed under control of another COBOL74 statement (for example, PERFORM, USE, SORT, and MERGE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement which causes iterative execution and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

2. When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

3. When any COBOL74 statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Another implicit transfer of control occurs after execution of the declarative section, as described in step 1 above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching statement ALTER does not constitute an explicit transfer of control, but affects the explicit transfer of control that occurs when the associated GO TO statement is executed. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control when the statement is executed in a called program.

In this manual, the term 'next executable statement' is used to refer to the next COBOL74 statement to which control is transferred according to the rules above and the rules associated with each language element in the PROCEDURE DIVISION.

There is no next executable statement following:

1. The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL74 statement.

2. The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL74 statement.

Explicit and Implicit Attributes

Attributes may be implicitly or explicitly specified. Any attribute which has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute assumes the default specification. Such an attribute is known as an implicit attribute.

For example, the usage of a data item need not be specified, in which case, a data item's usage is DISPLAY.

SECTION 3
CODING FORM

GENERAL

The format of the COBOL74 coding form (figure 3-1) has been defined by CODASYL and ANSI, and by common usage. The B 1000 COBOL74 Compiler accepts this standard format. Should program interchange be a major consideration, the user is directed to the ANSI standard.

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

FIELD DEFINITIONS

The same coding form is used for all four divisions of a COBOL74 program. These divisions must appear in proper order: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE. The following paragraphs describe the various fields of this coding form.

Sequence Area (Record Positions 1-6)

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

Indicator Area (Record Position 7)

Column 7 has the following functions:

1. A $ symbol in column 7 indicates that the record is used to specify options for compiler operation. Refer to Section 11 for additional information.

2. If column 7 contains an asterisk (*), the remainder of the record is considered to be a comment and, is not "compiled" to produce object code.

3. If column 7 contains a slash (/), the listing is advanced to channel 1 before printing, and the record is considered to be a comment record.

4. The presence of a hyphen (-) indicates that the last word or literal on the previous record is not complete and is continued on this record beginning in Area B (positions 12 through 72).

   Words and numeric literals may be split at any point by placing a hyphen in column 7 of the following record. Any rightmost blank spaces on a record are ignored as are the leftmost blank spaces on the continuation record.

   Nonnumeric literals are split in a slightly different fashion than words and numeric literals. On the initial record starting from the quotation mark, all information through position 72 is taken as part of the literal, and on the next record a quotation mark must be used to indicate the start of the second part of the literal.

   If there is no hyphen in column 7 of a coding line, it is assumed that the last character in the preceding line is followed by a space.

Burroughs COBOL CODING FORM

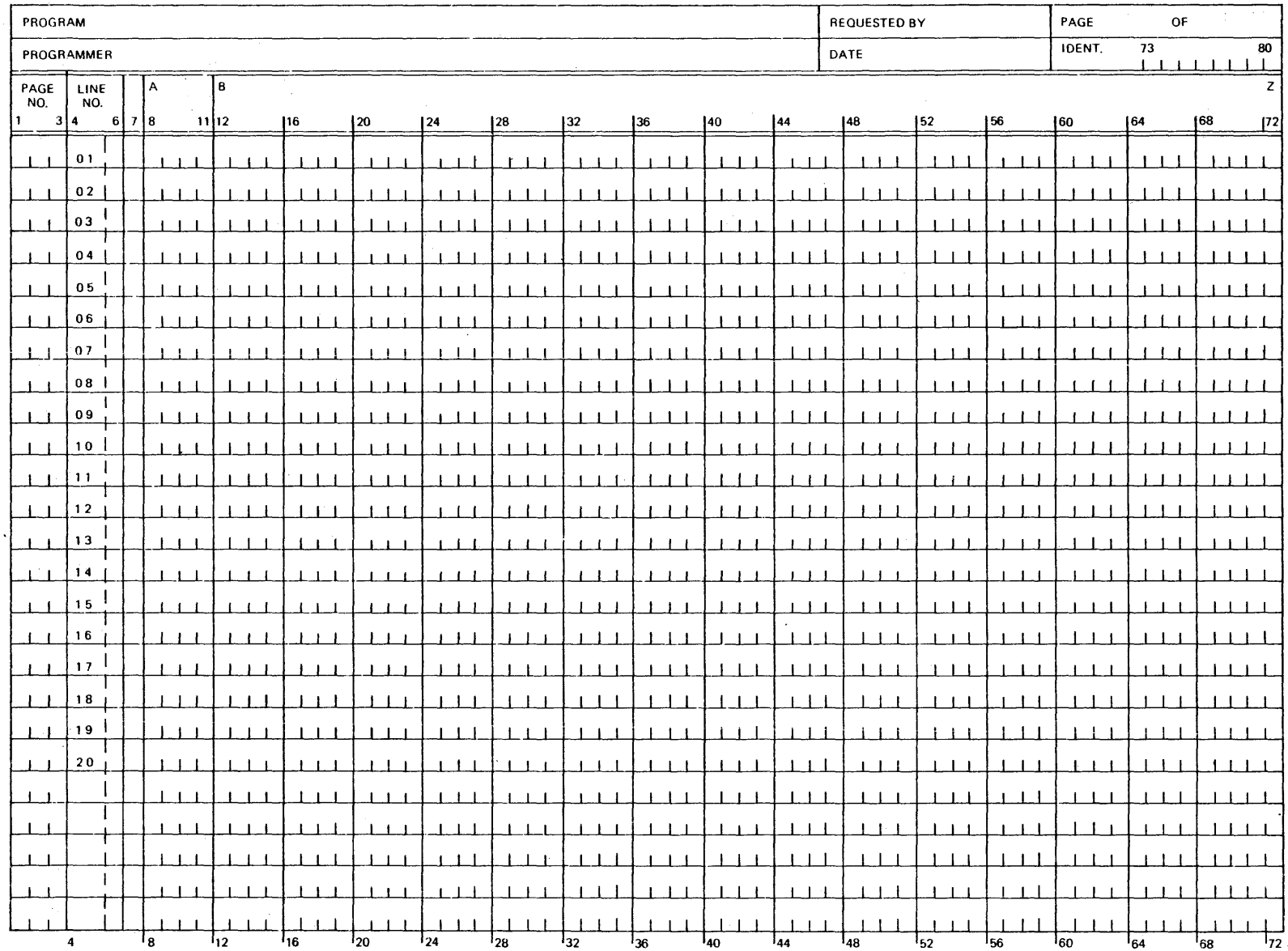


Figure 3-1. COBOL Coding Form

5. The letter D in column 7 specifies a debugging line. Any debugging line that consists solely of spaces from positions 8 through 72 is considered the same as a blank line.

   A debugging line is considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph. Therefore, 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 is only permitted in the program after the OBJECT-COMPUTER paragraph.

   Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain the letter D in position 7, and character-strings may not be broken across two lines.

Area A (Positions 8 through 11)

DIVISION, SECTION, and PARAGRAPH headers must begin in Area A. A division header consists of the division name (IDENTIFICATION, ENVIRONMENT, DATA, or PROCEDURE), followed by a space, then the word DIVISION followed by a period.

In the ENVIRONMENT and DATA DIVISIONS, a section header consists of the section-name, followed by a space, and then the word SECTION followed by a period.

In the PROCEDURE DIVISION, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period.

A paragraph header consists of the paragraph-name followed by a period. The first sentence of the paragraph may appear on the same line as the paragraph header.

Within the IDENTIFICATION and ENVIRONMENT divisions, the section and paragraph headers are fixed and only the headers shown in this manual are permitted. Within the PROCEDURE DIVISION, the section and paragraph headers are defined by the user.

Within the DATA DIVISION, the level indicators (FD, CD, SD) and the level numbers 01 and 77 must each begin in Area A, followed by the associated name and appropriate descriptive information.

The key words DECLARATIVES and END DECLARATIVES that precede and follow the declaratives portion of the PROCEDURE DIVISION, must appear on separate lines. Each must begin in Area A and must be followed by a period and a space.

Area B (Positions 12 through 72)

All entries which are not DIVISION, SECTION, or PARAGRAPH headers; level numbers 01 and 77, or level indicators (FD, SD, CD), must start in Area B.

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

Right Margin (Position 72)

The text of the program must appear between positions 8 and 72, inclusive. A word or statement may end in position 72.

Identification (Positions 73 through 80)

The identification field may contain any information desired by the user. The field is ignored but is reproduced on the output listing by the compiler. This field is normally used for the program name.

BLANK LINES

A blank line is one that contains no entries in the Indicator Area, Area A, and Area B. A blank line may appear anywhere in the source program except immediately preceding a continuation line.

PUNCTUATION

The following rules of punctuation apply to COBOL74 source programs for the B 1000 system.

1. A sentence must be terminated by a period followed by a space. A period must not appear within a sentence unless it is within a nonnumeric literal or is a decimal point in a numeric literal or PICTURE string.

2. Two or more names in a series must be separated by a space or by a comma. If used, commas must appear only where allowed.

3. Semicolons (;) are used only for readability and are never required.

4. A space must never be embedded in a name; hyphens are to be used instead. A hyphen must not start or terminate a name. For example:

   PAY-DAY (correct)

   - PAYDAY (wrong)

SAMPLE CODING

An extract sample from a source program, showing the continuation of both words and nonnumeric literals, is illustrated in figure 3-2.

Burroughs COBOL CODING FORM

PROGRAM: Continuation | REQUESTED BY | PAGE 2 OF 10
PROGRAMMER: John Q. Programmer | DATE | IDENT. 73 CONTIN 80

PAGE NO. 1 3 | LINE NO. 4 6 | 7 | A 8 11 | B 12 72

```
01  FILE-CONTROL. SELECT PRINTING-FILE ASSIGN TO TAPE RESERVE
02      2 AREAS.
03          SELECT MASTER-INPUT ASSIGN TO DISK
04  -       RESERVE 1 AREA.
05
06
07
08  /
09  WORKING-STORAGE SECTION.
10  01  A-FEW-LITERALS.                    05  NUM-
11  -       LIT-123                             VALUE 123
12  -       456.78                  PIC 99
13  -                                   999
14  -                                   9V99.
15      05  NON-NUM-LIT                PIC X(30)   VALUE "ALL C
16  -   HARACTERS VALID. Ø 1 +1 -1 ".
17
18
19
20
```

G12327

Figure 3-2. Example of Continuation of Words and Literals

SECTION 4
IDENTIFICATION DIVISION

GENERAL

The first division of the COBOL74 source program is the IDENTIFICATION DIVISION whose function is to identify the source program and the resultant output of compilation. In addition, the date the program was written, the date the source program was compiled, and other pertinent information can be included in the IDENTIFICATION DIVISION.

IDENTIFICATION DIVISION STRUCTURE

The structure of this division follows:

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

The following rules must be observed in the formation of the IDENTIFICATION DIVISION:

1. The IDENTIFICATION DIVISION must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

2. All paragraph-names must begin in positions 8 through 11 (Area A) of the coding form.

3. The comment-entry can consist of any combination of the characters from the B character set. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted; however, the comment-entry may be contained on one or more lines. A period must be present to denote the end of the comment entry.

PROGRAM-ID Paragraph

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

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

The following rules must be observed to form PROGRAM-ID paragraphs.

1. The program-name must conform to the rules for formation of a user-defined word.

2. The PROGRAM-ID paragraph contains the name of the program and must be present in every program.

3. The program-name identifies the source program and all listings pertaining to a particular program.

DATE-COMPILED Paragraph

The DATE-COMPILED paragraph provides the compilation date in the IDENTIFICATION DIVISION source program listing.

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

The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a one-line paragraph of the form:

DATE-COMPILED. current date.

Current date is composed of the elements year, month, day of month, hour, and minute and represents the date and time at which the compilation of the source program started.

Year is presented as four digits, starting in the position on the printed line corresponding to column 25 of a source line.

Month is presented as the name of the month in English, starting in the position on the printed line corresponding to column 30 of a source line.

Day of month is presented as two digits, starting in the position on the printed line two places to the right of the last character of the month entry.

Time is presented as four digits, with a colon between the second and third digits, and represents the time on a 24-hour clock. Time is presented in the position on the printed line five places to the right of the second digit of day of month.

Any leading zeros in the numeric fields are presented as the character '0' (zero).

If a compilation commences at 11:03 p.m., February 3, 2001, the current date would be presented as:

2001 FEBRUARY 03 23:03

CODING THE IDENTIFICATION DIVISION

Figure 4-1 provides an example of how the IDENTIFICATION DIVISION may be coded in the source program. Continued lines must begin in Area B and must not include a hyphen in the indicator area.

Burroughs COBOL CODING FORM

PROGRAM: Identification Division | REQUESTED BY: | PAGE 1 OF 8
PROGRAMMER: Bob | DATE: | IDENT. 73–80: ID-Divisn

PAGE NO.	LINE NO.	A (8–11) / B (12–72)
	01	IDENTIFICATION DIVISION.
	02	PROGRAM-ID. PAYROLL.
	03	AUTHOR. JOHNTNO.
	04	INSTALLATION. BURROUGHS CORPORATION,
	05	GOLETA, CA.
	06	DATE-WRITTEN. June 28, 1979.
	07	DATE-COMPILED.
	08	
	09	
	10	
	11	
	12	
	13	
	14	
	15	
	16	
	17	
	18	
	19	
	20	

G12328

Figure 4-1. Coding the IDENTIFICATION DIVISION

SECTION 5
ENVIRONMENT DIVISION

GENERAL

The ENVIRONMENT DIVISION is the second division of a COBOL74 source program. Its function is to specify the computer being used for the program compilation, specify the computer to be used for object program execution, associate files with the computer hardware devices, and provide the compiler with pertinent information about disk storage files defined within the program. Furthermore, this division is also used to specify input-output areas to be utilized for each file declared in a program.

ENVIRONMENT DIVISION ORGANIZATION

The ENVIRONMENT DIVISION consists of two sections. The CONFIGURATION SECTION contains the overall specifications of the computer. The INPUT-OUTPUT SECTION deals with files to be used in the object program. .

ENVIRONMENT DIVISION STRUCTURE

The structure of this division follows:

```
 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 following rules must be observed in the formulation of the ENVIRONMENT DIVISION.

1. The ENVIRONMENT DIVISION must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

2. All entries must begin in Area A (columns 8 through 11) of the coding form.

CONFIGURATION SECTION

The CONFIGURATION SECTION contains information concerning the system to be used for program compilation (SOURCE-COMPUTER), the system to be used for program execution (OBJECT-COMPUTER), and the SPECIAL-NAMES paragraph. The SPECIAL-NAMES paragraph relates hardware names used by the B 1000 COBOL74 Compiler to the mnemonic-names in the source program, and alphabet-names to character sets and/or collating sequences.

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled. It also contains an optional clause for use in debugging COBOL74 programs. Refer to section 10 in this manual for further information on the compile-time debugging switch.

General Format:

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

Syntax Rule:

1. The computer-name is any COBOL74 word and is handled as a comment entry which describes the computer upon which the source program is to be compiled. This computer name is for documentation only.

General Rule:

1. The computer-name is treated as a comment and ignored.

2. 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 all debugging lines are compiled.

3. 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 comment lines.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed.

General Format:

```
OBJECT-COMPUTER.  computer-name [ , MEMORY SIZE integer { WORDS      } ]
                                                        { CHARACTERS }
                                                        { MODULES    }
        [ , STACK SIZE IS integer-2 ]
        [ , PROGRAM COLLATING SEQUENCE IS alphabet-name ]
        [ , SEGMENT-LIMIT IS segment-number ] .
```

Syntax Rule:

1. Computer-name is a system name.
2. Segment-number must be an integer whose value is within the range of 1 through 49.

General Rules:

1. The computer-name is any COBOL74 word and is handled as a comment entry which describes the computer upon which the object program is to be executed. This computer name is for documentation only.
2. If the PROGRAM COLLATING SEQUENCE clause is specified, the collating sequence associated with alphabet-name is used to determine the truth value of any nonnumeric comparisons:
   a. Explicitly specified in relation conditions. Refer to Relation Condition in section 7 for additional information.
   b. Explicitly specified in condition-name conditions. Refer to Condition-Name Condition (Conditional Variable) in section 7 for additional information.
3. If the PROGRAM COLLATING SEQUENCE clause is not specified, the EBCDIC collating sequence is used.
4. If the PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet-name specified in that clause.
5. The PROGRAM COLLATING SEQUENCE clause is also applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase of the respective MERGE or SORT statement is specified. Refer to the MERGE and SORT statement in section 7.
6. The PROGRAM COLLATING SEQUENCE clause applies only to the program in which it is specified.

OBJECT-COMPUTER

7. If the STACK SIZE clause is specified, integer-2 indicates the number of entries in the perform stack. The default value is 25.

8. The SEGMENT-LIMIT clause specifies the limit of the fixed segment for sections numbered from 0 to 49. Refer to Segmentation in section 7 for further discussion.

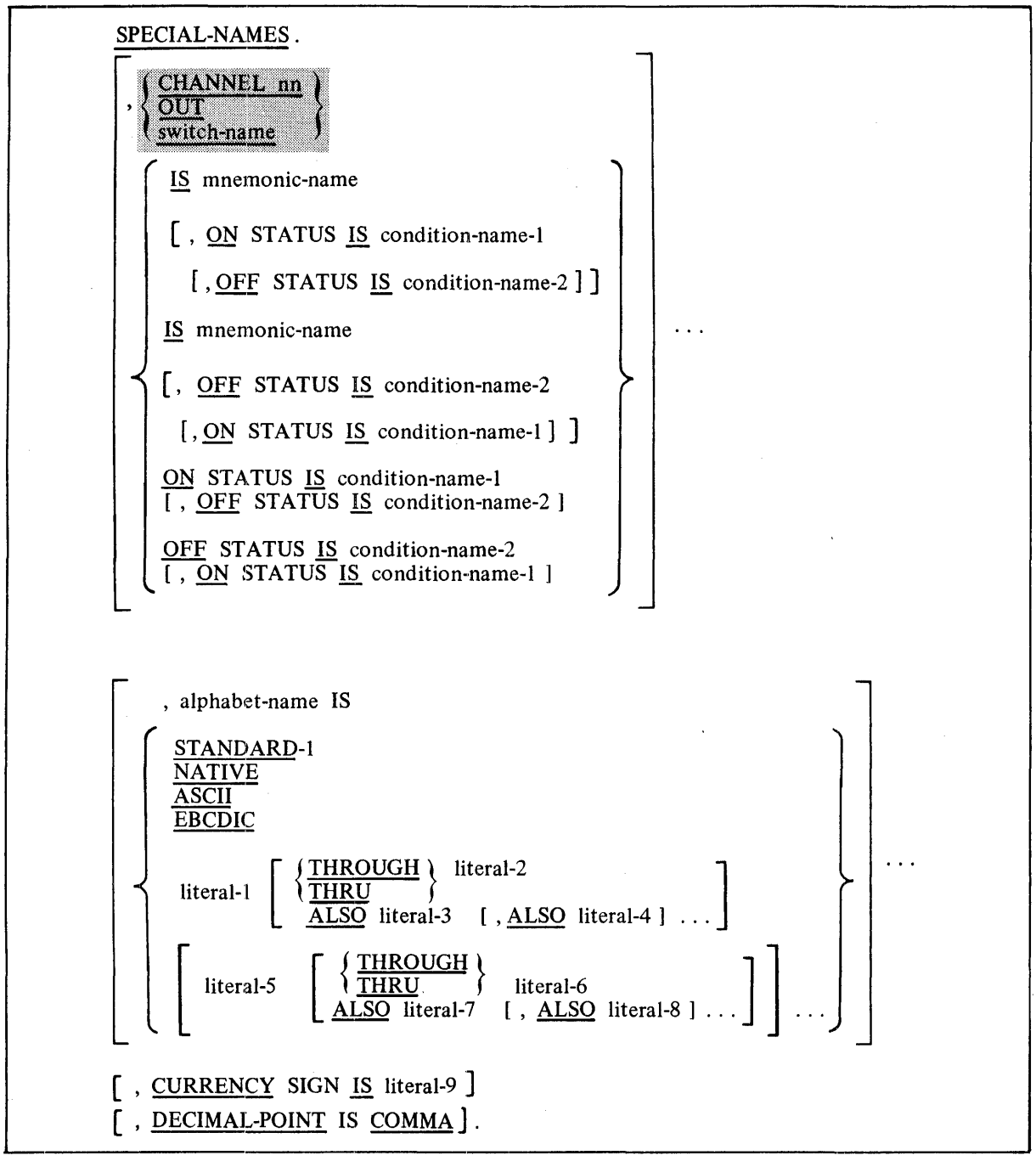
9. The MEMORY SIZE clause is used to increase the amount of dynamic memory.

10. WORDS and MODULES are equivalent to CHARACTERS.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph provides a means of relating names to user-specified mnemonic-names and of relating alphabet-names to character sets and/or collating sequences.

General Format:



SPECIAL-NAMES

Syntax Rules:

1. The literals specified in the literal phrase of the alphabet-name clause:

   a. If numeric, must be unsigned integers and must have a value within the range of 1 through the maximum number of characters in the EBCDIC character set.

   b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.

2. If the literal phrase of the alphabet-name clause is specified, a given character must not be specified more than once in an alphabet-name clause.

3. The words THRU and THROUGH are equivalent.

General Rules:

1. If switch-name is not specified, the associated mnemonic-name may be used in the ACCEPT, DISPLAY, SEND, and WRITE statements.

2. If CHANNEL nn was specified, the mnemonic-name is to be used in a WRITE or SEND statement. The nn must be an integer within the range 01 through 99.

3. If ODT was specified, the mnemonic-name is to be used in an ACCEPT or DISPLAY statement.

4. If switch-name is used, at least one condition-name must be associated with it. The status of the switch is specified by condition-names and interrogated by testing the condition-names. Refer to Switch-Status Condition in section 7 for additional information. Switch ON STATUS is any non-zero value.

The valid switches are SW1, SW2, SW3, SW4, SW5, SW6, SW7, and SW8. Switches are provided as a means of communication with the external environment. The meaning associated with each switch is entirely user defined. Switches may be set at program initiation and/or set or reset through the ODT during program execution.

5. The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause (refer to OBJECT-COMPUTER Paragraph in this section) or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (refer to MERGE and SORT in section 7), the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (refer to the File Description Structure in section 6), the alphabet-name clause specifies a character code set.

   a. ASCII is a synonym for STANDARD-1. If the STANDARD-1 or ASCII phrase is specified, the character code set and collating sequence identified is that defined in the American National Standard Code for Information Interchange, X3.4-1968.

   b. If the NATIVE phrase is specified, the native character code set and native collating sequence will be identified with the alphabet-name. The native character code set is EBCDIC and is the character code set associated with DISPLAY usage.

c. The correspondence between characters of the ASCII character code set and characters of the EBCDIC character code set is determined by standard translation tables for EBCDIC to ASCII and ASCII to EBCDIC translation. Refer to appendix C.

d. If the literal phrase is specified, the alphabet-name may not be referenced in a CODE-SET clause. Refer to the CODE-SET clause in section 6. The collating sequence identified is that defined according to the following rules:

Rule 1: The value of each literal specifies:

1). The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

2). The actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

Rule 2: The order in which the literals appear in the alphabet-name clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

Rule 3: Any characters within the native collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

Rule 4: If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the native character set in either ascending or descending sequence.

Rule 5: If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1, literal-3, literal-4, ..., are assigned to the same position in the collating sequence being specified.

6. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

7. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

SPECIAL-NAMES

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

   a. Digits 0 through 9.

   b. Alphabetic characters:

A	D	R	X
B	L	S	Z
C	P	V	space

   c. Special characters:

*	,	(	/
+	.	)	=
–	;	"	

   If the CURRENCY SIGN IS clause is not present, the default value dollar sign ($) is used in the PICTURE clause.

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

INPUT-OUTPUT SECTION

The INPUT-OUTPUT section contains information concerning files to be used by the object program, the manner of recording used or to be used, and the presence of any multiple-file tape or disk.

FILE CONCEPTS

In the following paragraphs, concepts of File Types, Organization, Access Mode, Current Record Pointer, I-O Status, INVALID KEY, AT END, and LINAGE-COUNTER are discussed pertaining to Sequential, Indexed, Relative, and Sort-Merge files.

Sequential I-O

Sequential I-O provides a capability to access records of a file in established sequence. The sequence is established as a result of writing the records to the file. It also provides for the sharing of memory areas among files.

Sequential I-O provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this manual. Within the PROCEDURE DIVISION, Sequential I-O provides full capabilities for the CLOSE, OPEN, READ, REWRITE, USE, and WRITE statements . Additional features available include: OPTIONAL files, the RESERVE clause, SAME RECORD AREA, REVERSED, and EXTEND options.

Relative I-O

Relative I-O provides the capability to access records of a mass storage file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's logical ordinal position in the file.

Relative I-O has full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this manual. Within the PROCEDURE DIVISION, the Relative I-O provides full capabilities for the CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements. Additional features available include: the RESERVE clause, DYNAMIC accessing, SAME RECORD AREA, READ NEXT, and the START statement.

For more information on the structure of the Relative file, see appendix G in this manual.

Indexed I-O

Indexed I-O provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one or more keys within that record.

Indexed I-O provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats for this manual. Within the PROCEDURE DIVISION, the Indexed I-O provides full capabilities for the CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements as specified in the formats for this manual. Additional features include: the RESERVE clause, DYNAMIC accessing, ALTERNATE KEYS, SAME RECORD AREA, READ NEXT, and the START statement.

Appendix G in this manual includes information on the structure and concepts of Indexed Sequential Access Method files, as well as efficient use and maintenance of the ISAM file.

INPUT-OUTPUT SECTION
FILE CONCEPTS

Queue Files

Queue files can be used by a program to reference a queue of messages. An OPEN statement logically connects a program to a queue file and a CLOSE statement logically disconnects a program from a queue file.

Message queueing is done on a first in, first out basis. A WRITE statement adds a message to a queue file and a READ statement deletes a message from a queue file.

The queue characteristics are declared through file attributes MAXSUBFILES and MAXCENSUS. For a description of these attributes refer to section 8. See appendix F for further information.

A single level of subfiles, termed a "queue file family," can be established for a queue file.

When an OPEN statement is executed for the file, all subfiles in the family are opened. The index of the subfile that is to be referenced is specified by the contents of the ACTUAL KEY data item associated with the file.

When a READ statement is executed for the file and the value of the ACTUAL KEY data item is not equal to zero, the ordinal subfile corresponding to that value is read. If no ACTUAL KEY data item is specified or if the contents of the ACTUAL KEY data item are zero, a message from one of the subfiles is read. The members of the queue family are examined beginning with queue number one, and the first queue member found not empty is read.

When a WRITE statement is executed for the file, the ordinal subfile corresponding to the value in the ACTUAL KEY data item is written. Note that a value of zero is not valid.

Remote Files

Remote files can be used in a COBOL74 program by specifying the file type in the SELECT clause. When more than one station is desired for a remote file, a value greater than one must be specified for the MAXSTATIONS file attribute for the remote file. COBOL74 supports the variable record concept for remote files. To read messages from and write messages to specific terminals, the LASTSTATION file attribute is used.

The file attributes can be referenced in section 8 and programming examples can be found in appendix F of this manual.

Port Files

Communication between user processes across a Burroughs Network Architecture (BNA) network is accomplished through the standard I-O file mechanism using a special kind of file called a port file. A user can communicate with a foreign process by performing read and write operations to a port file. A port file has one or more associated subports termed "subfiles," each of which can be connected to a different process. Communication between local processes cannot use port files without going through a BNA network.

A subfile provides a two-way, point-to-point logical communication path between two programs. In order to establish this path, each program must describe the desired connection. These descriptions are declared using file attributes.

The ACTUAL KEY clause specifies the subfile index of a port file. If the ACTUAL KEY is zero, the OPEN statement opens all subfiles, the READ statement performs a "non-selective" read operation, the WRITE statement performs a "broadcast" write operation, and the CLOSE statement closes all open subfiles associated with the port file.

If no ACTUAL KEY is specified, the file must contain a single subfile. That subfile is assumed in the associated I-O statements.

For more information concerning BNA, refer to the B 1000 Systems Burroughs Network Architecture (BNA) Installation and Operation Manual, as well as appendix F in this manual.

Sort-Merge

The Sort-Merge module provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the SORT or after the records have been combined by the MERGE.

Sort-Merge provides the facility for sorting one or more files, or combining two or more files, one or more times within a given execution of a COBOL74 program.

Relationship with Sequential I-O

The files specified in the USING phrase of the SORT and MERGE statements must be described implicitly or explicitly in the FILE-CONTROL paragraph as having sequential organization.

The file specified in the GIVING phrase of the SORT and MERGE statements must be described implicitly or explicitly in the FILE-CONTROL paragraph as having sequential organization.

No input-output statement may be executed for the file named in the Sort-Merge file description.

Organization

Sequential Files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

Relative File organization is permitted only on mass storage devices. A Relative File consists of records which are identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

A file whose organization is Indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

INPUT-OUTPUT SECTION
FILE CONCEPTS

The data item named in the RECORD KEY clause of the file control entry for an Indexed File is the prime record key for that file. For purposes of inserting, updating, and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record.

A data item named in the ALTERNATE RECORD KEY clause of the file control entry for an Indexed File, is an alternate record key for that file. The value of an alternate record key may be nonunique if the DUPLICATES phrase is specified. These keys provide alternate access paths for retrieval of records from the file.

Access Mode

The ACCESS MODE clause specifies the manner in which records are accessed in a file. There are three access modes: sequential, random, and dynamic. The allowable access modes, based upon the specified organization of the file, are discussed in the following paragraphs.

Sequential Files

In the sequential access mode, the sequence in which records are accessed is by the ascending order of ordinal location within the file. This order is established when the records are originally written to the file.

In the random access mode, the sequence in which records are accessed is specified by the contents of the ACTUAL KEY data item at the time the READ or WRITE statement is executed. The value of the ACTUAL KEY data item supplies the ordinal record number of the record to be accessed.

Relative File

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the random access mode, the sequence in which records are retrieved is controlled by the programmer. The desired record is accessed by placing its relative record number in a relative key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Indexed Files

In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key values. The order of retrieval of records within a set of records having duplicate record key values is the order in which the records were written into the set.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing the value of the record key in a record key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Current Record Pointer

For all file types, and for each user, the current record pointer is a conceptual entity used in selection of the next record to be accessed within a given file. The setting of the current record pointer is affected only by the OPEN, START, and READ statements. The WRITE statement for a sequentially organized file may also affect the setting of the current record pointer.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement and before any applicable USE procedure is executed, to indicate to the COBOL74 program the status of that input-output operation. The specification of the FILE STATUS clause (or a USE procedure) for a file indicates that the program is capable of determining and correcting any errors encountered during an I-O operation on that file.

Interrogation and proper interpretation of the FILE STATUS data item after an I-O operation on a file helps to insure the integrity of that file and can be an aid when debugging the program.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to a value which indicates one of the following conditions upon completion of the input-output operation.

Value	Condition
0	Successful Completion
1	At End
2	Invalid Key
3	Permanent Error
8	Burroughs-Defined Condition
9	Burroughs-Defined Condition

The above conditions are defined in following text.

Successful Completion

The input-output statement was successfully executed.

At End

The sequential READ statement was unsuccessfully executed as a result of:

1. An attempt to read other than a queue or port file record when no next logical record exists in the file.

2. The first READ statement being executed for a file described with the OPTIONAL clause, when that file was not available to the program at the time its associated OPEN statement was executed.

INPUT-OUTPUT SECTION
FILE CONCEPTS

3. An attempt to read an empty queue with no other queue files connected to this queue by way of an OPEN OUTPUT or an OPEN I-O statement.

4. An attempt to read a subfile when no next logical record exists and the communication path with the connected process is no longer established.

Invalid Key

The input-output statement was unsuccessfully executed as a result of one of the following:

1. For a Format 2 READ statement, on other than a queue or port file, the contents of the ACTUAL KEY data item were less than 1 or greater than the original number of the last record ever written to the file.

2. For a Format 2 WRITE statement, on other than a queue or port file, the contents of the ACTUAL KEY data item were less than 1 or greater than the last record allowed to be written because of the specification of a maximum file size.

3. For a Format 2 READ statement on a queue or port file, the contents of the ACTUAL KEY data item were less than zero or greater than the number of subfiles in the queue or port file family.

4. For a Format 2 WRITE statement on a queue or port file, the contents of the ACTUAL KEY data item were less than 1 or greater than the number of subfiles in the queue or port file family.

Permanent Error

The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error. When there is no FILE STATUS clause and no USE procedure specified for a file, detection of a Permanent Error condition will cause the program to terminate abnormally.

Burroughs-Defined Condition

The input-output statement encountered conditions other than those already defined and may have been unsuccessfully executed, depending on the value of status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character contains a value as follows:

1. If no further information is available concerning the input-output operation, then status key 2 contains a value of 0.

2. When status key 1 contains a value of 0 indicating a successful completion, status key 2 may contain a value of 2 indicating a duplicate key. This condition indicates:

   a. For a READ statement, the key value for the current key of reference is equal to the value of that same key in the next record within the current key of reference.

b. For a WRITE or REWRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

3. When status key 1 contains a value of 2 indicating an INVALID KEY condition, status key 2 is used to designate the case of that condition as follows:

   a. A value of 1 in status key 2 indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated (refer to WRITE in section 7), or the prime record key value has been changed by the COBOL74 program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

   b. A value of 2 in status key 2 indicates a duplicate key value. An attempt was made to write or rewrite a record that would create a duplicate key in an indexed file.

   c. A value of 3 in status key 2 indicates no record found. An attempt is made to access a record, identified by a key, but that record does not exist in the file.

   d. A value of 4 in status key 2 indicates a boundary violation. An attempt was made to write beyond the externally defined boundaries of an indexed file. The compiler specifies the manner in which these boundaries are defined.

4. When status key 1 contains a value of 3 indicating a permanent error condition, status key 2 may contain a value of 4 indicating a boundary violation. This condition indicates that an attempt was made to write beyond the externally defined boundaries of a sequential file. The compiler specifies the manner in which these boundaries are defined.

5. When status key 1 contains a value of 8 indicating a Burroughs-defined condition and the value of status key 2 equals 1, a FILE NOT OPEN condition is indicated. If the AVAILABLE phrase is specified in an OPEN statement, this status key value indicates that the file was not opened before control was returned to the next statement.

6. When status key 1 contains a value of 9 indicating a Burroughs-defined condition, the value of status key 2 indicates the condition as follows:

Status Key 2 Value	Condition
1	Short Block
2	Data Error
4	Q-Empty or No Data
5	Q-Full or No Buffer
6	Timeout
7	Break on Output
9	Unexpected I-O Error

Short Block

Because of the limitation of the physical recording medium, the system is unable to determine whether the logical record returned had been written to the file. Determination of the validity of the data record is the responsibility of the programmer.

INPUT-OUTPUT SECTION
FILE CONCEPTS

Data Error

When logical records are declared variable in length and the logical record length is supplied by the programmer (by means of the RECORD CONTAINS clause), a data error occurs on a READ, WRITE, or REWRITE statement if the logical record length supplied is less than the minimum record size or greater than the maximum record size declared for the file. This condition initiates no input-output operation nor does it cause data to be transferred to or from the record area.

Q-Empty or No Data

This condition indicates that the queue file is empty or that no data is available when the WITH NO WAIT clause is used with the READ statement.

Q-Full or No Buffer

This condition indicates that no buffer is available when the WITH NO WAIT clause is used with the WRITE statement.

Timeout

A time limit has elapsed prior to the transfer of data to or from the hardware device.

Break on Output

For an output or input-output file, this condition occurs if the physical hardware device is equipped with a break such that an operator can halt the transfer of data in process.

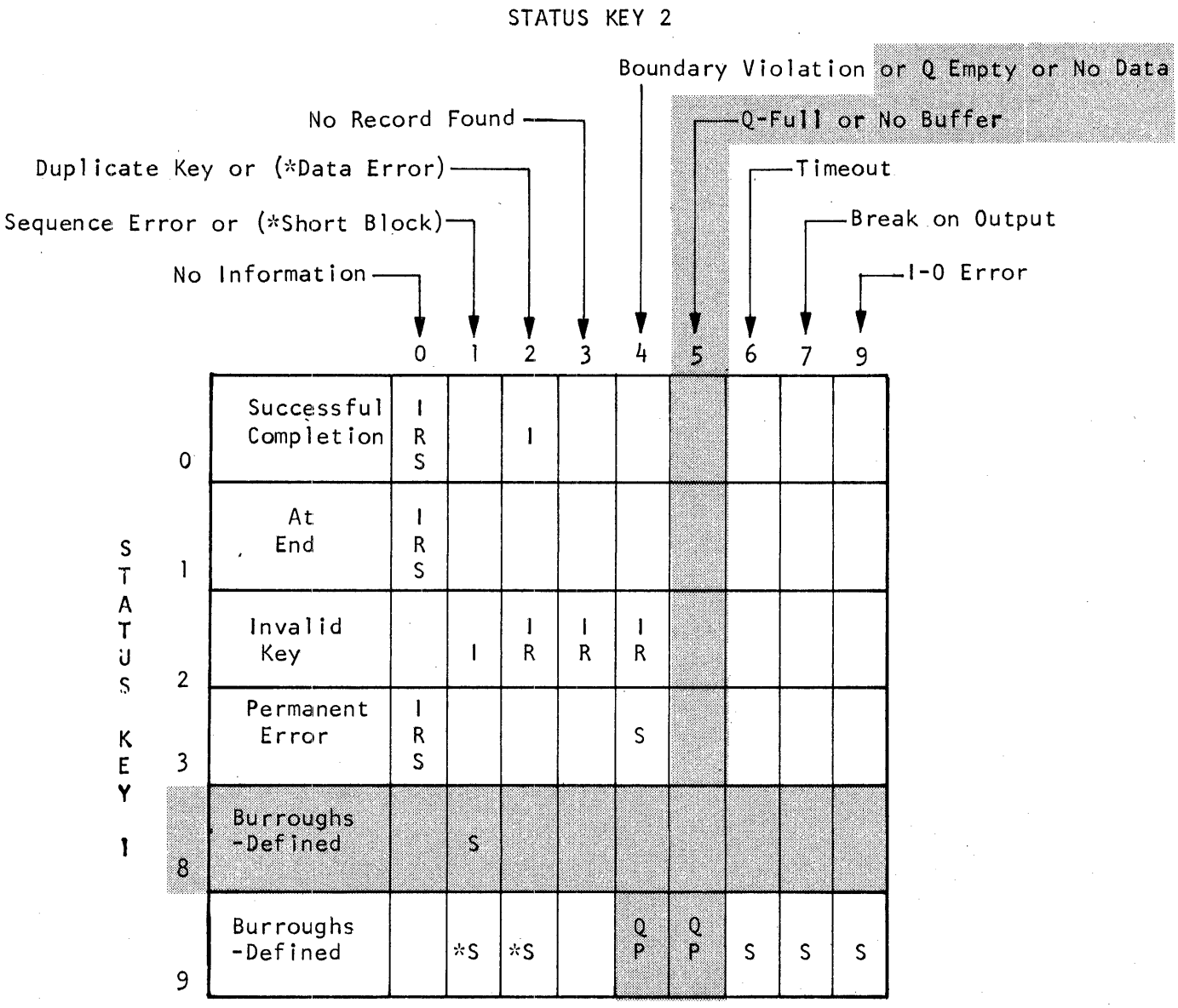
Unexpected I-O Error

An error may have occurred in the input-output operation but its nature cannot be determined.

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in table 5-1. The letter I (Indexed), P (Port), R (Relative), S (Sequential), or Q (Queue) at an intersection indicates a valid permissible combination.

Table 5-1. Status Key Combinations



The (*) distinguishes which error occurred when there are two with the same value.

INPUT-OUTPUT SECTION
FILE CONCEPTS

Invalid Key

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE, or DELETE statement. For details of the causes of the condition, refer to the START, READ, WRITE, REWRITE, and DELETE statements in section 7.

When the INVALID KEY condition is recognized, these actions are taken in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. Refer to I-O Status in this section for additional information.

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, but a USE procedure is specified for this file, either explicitly or implicitly, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

At End

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition refer to the READ statement in section 7.

Linage-Counter

For Sequential Files only, the reserved word LINAGE-COUNTER is a name for a special register generated by the presence of a LINAGE clause in a file description entry. The implicit description is that of an unsigned integer whose size is equal to the size of integer-1 or the data item referenced by data-name-1 in the LINAGE clause. Refer to the LINAGE clause in section 6 for the rules governing the LINAGE-COUNTER.

FILE-CONTROL PARAGRAPH

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format:

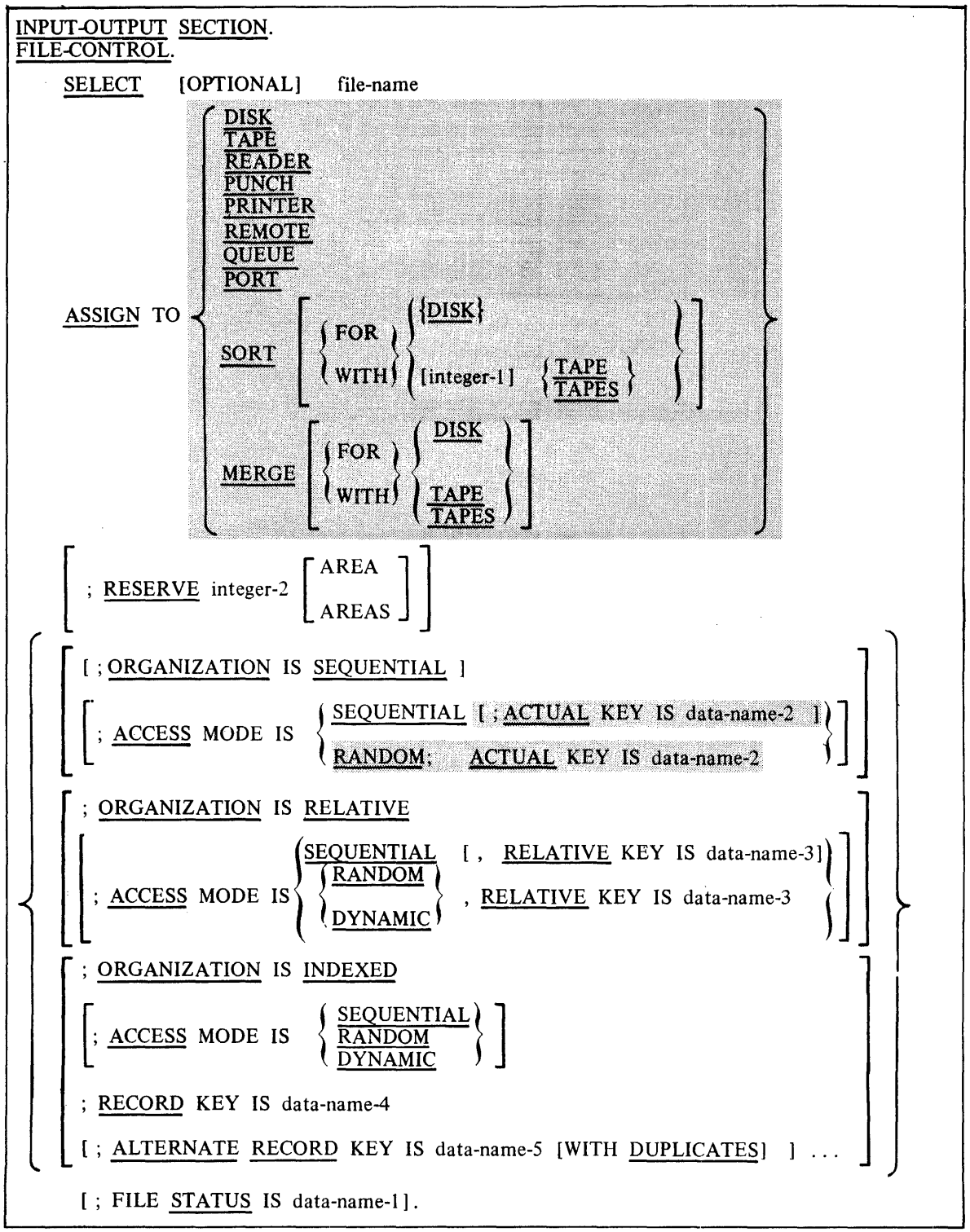
```
FILE-CONTROL.  { file-control-entry } ...
```

File Control Entry

The file control entry names a file and may specify other file-related information. If using the sort-merge features, the file control entry names a sort or merge file and specifies the association of the file to a storage medium.

FILE-CONTROL

General Format:



FILE-CONTROL

Syntax Rules:

1. The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

2. Each file described in the DATA DIVISION must be named only once with a file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the DATA DIVISION. For an Indexed File, the first eight letters of the file-name must be unique.

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause without the ACTUAL KEY phrase is implied.

4. Data-name-1 must be defined in the DATA DIVISION as a two-character data item of the category alphanumeric and must not be defined in the FILE SECTION or the COMMUNICATION SECTION.

5. Each sort or merge file described in the DATA DIVISION must be named only once with a file-name in the FILE-CONTROL paragraph. Each sort or merge file specified in the file control entry must have a Sort-Merge File description entry in the DATA DIVISION.

6. If file-name represents a sort or merge file, only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph.

7. Integer-1 must have a value within the range of 3 through 8 inclusive.

8. Data-name-1, data-name-3, data-name-4, and data-name-5 may be qualified.

9. The ORGANIZATION IS SEQUENTIAL clause applies only to the program in which it is specified.

10. The OPTIONAL phrase may only be specified for sequential input files. Its specification is required for input files that are not necessarily present each time the object program is executed.

11. The ACTUAL KEY phrase may be specified only for mass storage files, port files, and queue files.

12. Data-name-2 must be defined in the DATA DIVISION as an elementary numeric item that describes an unsigned integer.

13. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.

14. Data-name-3 must not be defined in a record description entry associated with that file-name.

15. The data item referenced by data-name-3 must be defined as an unsigned integer.

16. The data items referenced by data-name-4 and data-name-5 must each be defined as a data item of the category alphanumeric within a record description entry associated with that file-name.

FILE-CONTROL

17. Neither data-name-4 nor data-name-5 can describe an item whose size is variable. Refer to the OCCURS clause in section 6 for more information.

18. Data-name-5 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-4 or by any other data-name-5 associated with this file.

19. A REMOTE file type cannot have an associated ACTUAL KEY phrase specified. A QUEUE file type can have an ACTUAL KEY phrase specified.

General Rules:

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. For Relative and Indexed Files, the storage medium must be DISK.

2. If TAPE or TAPES is specified as the primary work medium of the SORT and integer-1 is not specified, the default number of tapes is 3.

3. The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-2. If the RESERVE clause is not specified, one input-output area is allocated. This clause has no meaning in the processing of Indexed files.

4. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

5. Records in a sequentially organized file may be accessed in either a sequential or a random manner, depending upon the ACCESS MODE clause. Random access mode may be specified only for mass storage files.

6. When the access mode of a Relative File is sequential, records in the file are accessed in the order of ascending relative record numbers of existing records in the file.

7. When the access mode of an Indexed File is sequential, records in the file are accessed in the order of ascending record key values within a given key of reference.

8. When the FILE STATUS clause is specified, a value is moved by the operating system into the data item specified by data-name-1 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement. Refer to I-O Status in this section for additional information.

9. If the access mode of a Relative File is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

10. If the access mode of an Indexed File is random, the value of the RECORD KEY data item indicates the record to be accessed.

11. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. Refer to General Rules 5 and 8, or 6 and 9 under the FILE-CONTROL statement.

12. All records stored in a Relative File are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, and so forth.

13. In a Relative File, the data item specified by data-name-3 is used to communicate a relative record number between the program and the MCP.

14. The RECORD KEY clause specifies the prime record key for the file. The values must be unique among records of the file. The prime record key provides an access path to records in an Indexed File.

15. An ALTERNATE RECORD KEY clause specifies an alternate record key for the file and provides an alternate access path to records in an Indexed File.

16. In an Indexed File, the data descriptions of data-name-4 and data-name-5 as well as the relative locations within a record must be the same as that used when the file was created. The number of alternate keys for the file must also be the same as that used when the file was created.

17. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

18. DISK specifies that mass storage is the storage medium of the file. A more precise specification of the medium may be made in the VALUE OF clause in the File Description entry or by means external to the language.

19. If the ACTUAL KEY clause is specified, the following rules apply:

    a. For mass storage files specifying an ACTUAL KEY, the value of the ACTUAL KEY data item specifies the logical ordinal position of the record in the file.

    b. For port files, the value of the ACTUAL KEY data item specifies the subfile index of the port file.

20. The ACTUAL KEY clause must be specified for a port file that contains more than one subfile.

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph specifies the memory area which is to be shared by different files, and the location of files on a multiple file reel.

General Format:

```
I-O-CONTROL.

[ ; SAME [ RECORD     ] AREA FOR file-name-3 { , file-name-4 } ... ] ... .
         [ SORT       ]
         [ SORT-MERGE ]
```

Syntax Rules:

1. The I-O-CONTROL paragraph is optional.
2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.
3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.
4. The four formats of the SAME clause (SAME AREA, SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following description.

More than one SAME clause may be included in a program; however, the following restrictions apply:

a. A file-name must not appear in more than one SAME AREA clause.

b. A file-name must not appear in more than one SAME RECORD AREA clause.

c. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

d. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

e. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clause(s).

5. The files referenced in the SAME AREA, SAME RECORD AREA, SAME SORT AREA, or SAME SORT-MERGE AREA clause need not all have the same organization or access type.

General Rules:

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

   a. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies a memory area which is made available for use in sorting or merging each sort or merge file named. Thus, any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

   b. In addition, storage areas assigned to files that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause. The extent of such allocation is specified by the MCP.

   c. Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, a SAME AREA or SAME RECORD AREA clause naming these files must also be included in the program.

   d. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any files named in this clause that are not sort-merge files must not be open.

2. The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same memory area during processing. The area being shared includes all storage area assigned to the files specified. It is not valid to have more than one of the files open at the same time. Refer to syntax rule 4c under the I-O-CONTROL paragraph.

3. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area; records are aligned on the leftmost character position.

ENVIRONMENT DIVISION CODING

CODING THE ENVIRONMENT DIVISION

Figure 5-1 provides an example of how the ENVIRONMENT DIVISION may be coded in the source program.

Burroughs COBOL CODING FORM

PROGRAM ENVIRONMENT DIVISION CODING | REQUESTED BY | PAGE 7 OF 20
PROGRAMMER KOSBY A. | DATE | IDENT. 73 ENVIRON 80

PAGE NO. 1 3	LINE NO. 4 6	7	A 8 11	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72	Z
	01		ENVIRONMENT DIVISION.		
	02		CONFIGURATION SECTION.		
	03		SOURCE-COMPUTER. B-1900.		
	04		OBJECT-COMPUTER. B-1900, SEGMENT-LIMIT IS 5.		
	05		SPECIAL-NAMES. DECIMAL-POINT IS COMMA		
	06		SW1 IS SR0		
	07		SW1 ON STATUS IS SWITCH-1-ON		
	08		SW2 ON STATUS IS SWITCH-2-ON		
	09		OFF STATUS IS SWITCH-2-OFF		
	10		REC-MODE IS ASCII.		
	11				
	12		INPUT-OUTPUT SECTION.		
	13		FILE-CONTROL. SELECT OPTIONAL ALL-TAPE ASSIGN TO TAPE.		
	14		SELECT FILE1 ASSIGN TO DISK		
	15		ORGANIZATION IS RELATIVE		
	16		ACCESS MODE DYNAMIC		
	17		RELATIVE KEY IS REL-KEY.		
	18		SELECT OTHER-TAPE ASSIGN TO TAPE.		
	19		I-O-CONTROL.		
	20		SAME RECORD AREA FOR ALL-TAPE, OTHER-TAPE.		

G12329

Figure 5-1. Coding the ENVIRONMENT DIVISION

SECTION 6
DATA DIVISION

GENERAL

The DATA DIVISION describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed belongs to these three categories:

1. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

2. That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

3. Constants which are defined by the user.

DATA DIVISION ORGANIZATION

The DATA DIVISION, which is one of the required divisions in a program, is subdivided into sections. These are FILE, WORKING-STORAGE, LINKAGE, and COMMUNICATION SECTIONS.

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry.

The WORKING-STORAGE SECTION describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are preassigned in the source program.

The LINKAGE SECTION appears in the called program and describes data items that are to be referred to by the calling program and the called program. The structure is the same as the WORKING-STORAGE SECTION.

The COMMUNICATION SECTION describes the data items in the source program that serve as the interface between the Data Communication Subsystem and the program.

DATA DIVISION STRUCTURE

The following structure shows the general format of the sections of the DATA DIVISION, and defines the order of presentation in the source program.

```
DATA DIVISION.
[ FILE SECTION.                                                          ]
[   [ file-description-entry [record-description-entry]  ...            ]
[   [ sort-merge-file-description-entry (record-description-entry) ... ] ... ]

[ WORKING-STORAGE SECTION.                                               ]
[   [ 77-level-description-entry ]                                       ]
[   [ record-description-entry   ]  ...                                  ]

[ LINKAGE SECTION.                                                       ]
[   [ 77-level-description-entry ]                                       ]
[   [ record-description-entry   ]  ...                                  ]

[ COMMUNICATION SECTION.                                                 ]
[   [ communication-description-entry    [record-description-entry] ... ] ... ]
```

FILE SECTION

In a COBOL74 program, the file description entry (FD) represents the highest level of organization in the FILE SECTION. The FILE SECTION header is followed by a file description entry consisting of a level indicator (FD), a file-name, and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label records, the value of label items, the names of the data records which comprise the file, and the number of lines to be written on a logical printer page. The entry is terminated by a period.

The Sort-Merge file description (SD) gives information about the size and the name of the data records associated with the file to be sorted or merged. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

Record Description

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

Examples:

```
01 DATA-ITEM-ONE     PICTURE X(10).
03 LINE-COUNT        PICTURE 999 VALUE ZEROES.
```

A record description has a hierarchical structure and, therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in Concepts of Levels, section 2, while the elements allowed in a record description are shown in the data description structure.

File Description Structure

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

FILE DESCRIPTION

General Format:

```
[FD file-name
    [ ; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS / CHARACTERS} ]
    [ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
        [ DEPENDING ON data-name-1 ] ]
    [ ; LABEL {RECORD IS / RECORDS ARE} {STANDARD / OMITTED} ]
    [ ; VALUE OF attribute-name-1 IS {data-name-2 / literal-1}
        [ , attribute-name-2 IS {data-name-3 / literal-2} ] ... ] ...
    [ ; DATA {RECORD IS / RECORDS ARE} data-name-4 [, data-name-5] ... ]
    [ ; LINAGE IS {data-name-6 / integer-5} LINES [, WITH FOOTING AT {data-name-7 / integer-6} ]
        [ , LINES AT TOP {data-name-8 / integer-7} ] [ , LINES AT BOTTOM {data-name-9 / integer-8} ] ]
    [ ; CODE-SET IS alphabet-name ].
    { record-description-entry } ... ] ...
```

Syntax Rules:

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.
2. The clauses which follow the name of the file are optional in many cases, and order of appearance is immaterial.
3. One or more record description entries must follow the file description entry.

Sort-Merge File Description Structure

The Sort-Merge file description entry furnishes information concerning the physical structure, identification, and record names of the file to be sorted or merged.

General Format:

```
[ SD file-name
    [ ; RECORD CONTAINS [ integer-1 TO ] integer-2 CHARACTERS
        [ DEPENDING ON data-name-1 ] ]
    [ ; DATA { RECORD IS  } data-name-2 [ , data-name-3 ] ... ]
             { RECORDS ARE }
    [ VALUE OF attribute-name-1 IS { data-name-4 } ]
                                   { literal-1   }
      [ , attribute-name-2 IS { data-name-5 } ... ] .
                              { literal-2   }
    { record-description-entry } ... ] ...
```

Syntax Rules:

1. The level indicator SD identifies the beginning of the Sort-Merge file description entry and must precede the file-name.

2. The clauses which follow the name of the file are optional and their order of appearance is immaterial.

3. One or more record description entries must follow the Sort-Merge file description entry; however, no input-output statements may be executed for this file.

CODING THE FILE SECTION

Figure 6-1 illustrates the coding of the FILE SECTION and shows a File Description and a Sort-Merge File Description.

Burroughs COBOL CODING FORM

PROGRAM FILE SECTION EXAMPLE | REQUESTED BY | PAGE 3 OF 33
PROGRAMMER CHRISTINE | DATE | IDENT. 73 FILESECT 80

PAGE NO. 1 3	LINE NO. 4 6	7	A 8 11	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 Z
	01		FILE SECTION.	
	02		FD MASTER-IN-FILE	
	03		BLOCK CONTAINS 3 RECORDS	
	04		RECORD CONTAINS 50 CHARACTERS	
	05		VALUE OF TITLE IS "MASTER/INFILE ON DSKNAME".	
	06			
	07		01 EMPLOY-REC.	
	08		05 EMP-NAME PIC X(20).	
	09		05 EMP-NUMBER PIC 9(8).	
	10		05 EMP-DEPT PIC X(20).	
	11		05 EMP-DEPT-CODE PIC 99.	
	12			
	13		SD MY-SORT-FILE	
	14		RECORD CONTAINS 80 CHARACTERS.	
	15			
	16		01 SORT-REC.	
	17		05 SR-EMP-NUMBER PIC 9(8).	
	18		05 SI-DEPT-CODE PIC 99.	
	19		05 FILLER PIC X(70).	
	20			

G12330

Figure 6-1. Coding the FILE SECTION

BLOCK CONTAINS

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

General Format:

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

Syntax Rules:

1. When the word RECORDS is not specified, the value of integer-2 must not be less than the largest record specified for the file.

General Rules:

1. This clause is required when the physical record contains more than one logical record. If this clause is not specified, the physical record is presumed to contain one logical record that is as large as the largest record specified for this file.

2. When the word RECORDS is specified, the physical record size is considered to be integer-2 times the largest record specified for this file.

3. When the word CHARACTERS is specified, the physical record size is considered to be integer-2 characters. If integer-2 is not a multiple of the largest record specified for this file, the physical record size is adjusted so that it is a multiple of the largest record specified, but must not exceed the value of integer-2.

4. Integer-2 represents the exact number of records or characters in the physical record unless the size is adjusted as stated in General Rule 3.

5. If logical records of differing sizes are grouped into one physical record, the amount of data transferred from the record area to the physical record depends on the size of the record named in the WRITE or REWRITE statement. In this case, the logical records are aligned on maximum record-size boundaries. If the size of the record named is not equal to the maximum record size specified for the file, the data is transferred to the physical record according to the rules specified for the MOVE statement without the CORRESPONDING phrase, with the sending area considered to be a group item.

6. For Index Sequential data files, the maximum number of logical records that can be contained in a physical record (a block) is 255. The maximum number of characters that can be contained in a physical record of an Index Sequential data file is 8,192. Disregarding these limits can cause unpredictable system halts. The physical record size (blocksize) of an Index Sequential data file is calculated as follows.

   (RECORD SIZE * RECORDS PER BLOCK) + (INDEX SEQUENTIAL DATA FILE BLOCK CONTROL INFORMATION)

   INDEX SEQUENTIAL DATA FILE BLOCK CONTROL INFORMATION = (RECORDS PER BLOCK + 7) / 8

The fractional part of the number in the calculation for the Index Sequential data file block control information (BCI) is truncated.

CODE-SET

The CODE-SET clause specifies the character code set used to represent data on the external media.

General Format:

CODE-SET IS alphabet-name

Syntax Rules:

1. When the CODE-SET clause is specified for a file, all data in that file must be described as USAGE IS DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.

2. The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase, and must be declared in the SPECIAL-NAMES Paragraph.

General Rules:

1. If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation. Refer to the SPECIAL-NAMES paragraph in section 5.

2. If the CODE-SET clause is not specified, the native character code set, EBCDIC, is assumed for data on the external media.

3. For the creation of the translate table to be used by the COBOL74 compiler, refer to the CREATE/TABLE utility. For example, when ASCII is the character code set, the COBOL74 compiler requires a file called TRANSLATE/ASCII to perform the translation. The file, TRANSLATE/ASCII, is created by using the CREATE/TABLE utility.

EXAMPLE:

The following example shows how the character code set desired is declared in the SPECIAL-NAMES paragraph and then used in the CODE-SET clause.

```
SPECIAL-NAMES.
    ODT IS SPO
    SW1 ON STATUS IS SWITCH-1-ON
    MY-REC-MODE IS ASCII.


FD  IN-FILE
    BLOCK CONTAINS 5 RECORDS
    CODE-SET IS MY-REC-MODE.

01  DATA-ITEM-ONE                PIC X(80).
```

DATA RECORDS

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

General Format:

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

Syntax Rules:

1. Data-name-1 and data-name-2 are the names of data records and must have 01 level-number record descriptions, with these same names, associated with them.

General Rules:

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of different sizes, different formats, and so forth. The order in which they are listed is not significant.

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

LABEL RECORDS

The LABEL RECORDS clause specifies whether labels are present.

General Format:

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

Syntax Rules:

1. When this clause is not specified, the STANDARD option is assumed.

General Rules:

1. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned. For Indexed and Relative Files, OMITTED is documentation only.

2. In a Sequential File, STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the label specifications.

3. The LABEL RECORDS clause is ignored for mass storage files. All mass storage files are labeled.

LINAGE

The LINAGE clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

General Format:

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

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

LINAGE

The relationship of the page components is shown in figure 6-2.

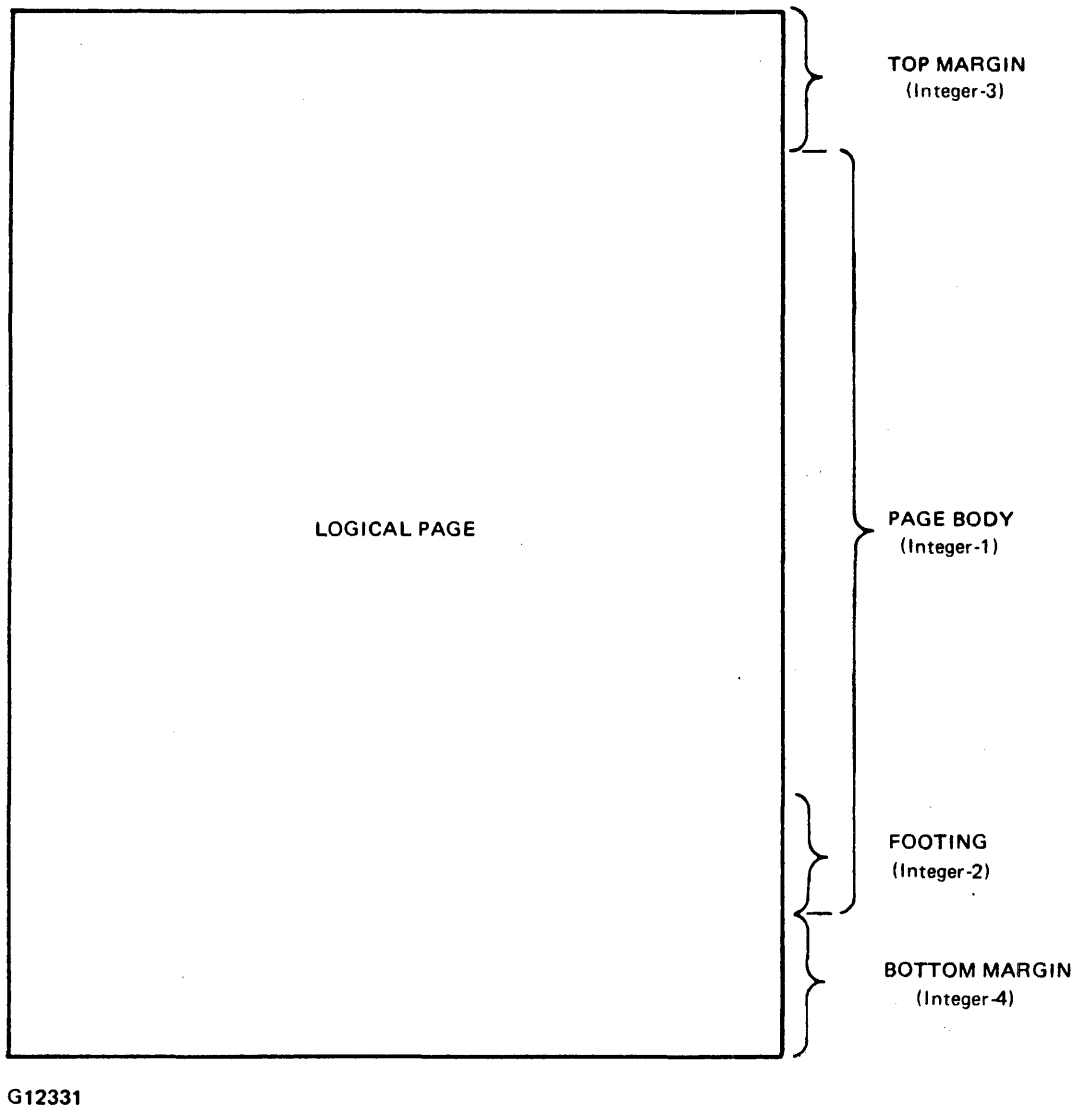


Figure 6-2. Linage Page Relationship

Syntax Rules:

1. Data names must be elementary unsigned numeric items.
2. The value of integer-1 must be greater than zero.
3. The value of integer-2 must not be greater than integer-1.
4. Integer-3 and integer-4 may be equal to zero.

LINAGE

General Rules:

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of the number of lines.

Example:

Logical page = LINAGE LINES + LINES AT TOP + LINES AT BOTTOM

If LINES AT TOP or LINES AT BOTTOM are not specified, the values for these functions are 0.

If the FOOTING phrase is not specified, the assumed value is equal to integer-1 (or data-name-1).

There is not necessarily any relationship between the size of the logical page and the size of a physical page.

2. The value of integer-1 (data-name-1) specifies the number of lines that can be written and/or spaced on the logical page. That part of the logical page in which these lines can be written and/or spaced is called the page body.

3. The value of integer-3 (data-name-3) specifies the number of lines that comprise the top margin on the logical page.

4. The value of integer-4 (data-name-4) specifies the number of lines that comprise the bottom margin on the logical page.

5. The value of integer-2 (data-name-2) specifies the line number within the page body at which the footing area begins, where 0 < integer-2 < integer-1.

6. The values of integer-1, integer-2, and integer-4, if specified, are used at the time the file is opened with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page.

   The value of integer-2, if specified, is used at that time to define the footing area.

   These values are used for all logical pages written for the file during a given execution of the program.

7. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, are used as follows:

   a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, are used to specify the number of lines that are to comprise each of the indicated sections of the FIRST logical page.

   b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs, are used to specify the number of lines that are to comprise each of the indicated sections for the next logical page.

LINAGE

8. The value of data-name-2, if specified at the time an OPEN statement with the OUTPUT phrase is executed for the file, is used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, data-name-2 is used to define the footing area for the next logical page.

LINAGE-COUNTER

A LINAGE-COUNTER is generated by the presence of a LINAGE clause The value in the LINAGE-COUNTER at any given time represents the line number at which the logical device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

1. A separate LINAGE-COUNTER is supplied for each file described in the FILE SECTION whose file description entry contains a LINAGE clause.

2. LINAGE-COUNTER may be referenced but not modified by PROCEDURE DIVISION statements.

   Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file name when necessary.

3. LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement:

   a. When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to 1.

   b. When the ADVANCING identifier-2 or the integer phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer or the value of identifier-2.

   c. When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value of 1.

   d. The value of LINAGE-COUNTER is automatically reset to 1 when the device is repositioned to the first line that can be written on for each of the succeeding logical pages.

4. The value of LINAGE-COUNTER is automatically set to 1 at the time an OPEN statement is executed for the associated file.

Example:

The following example shows the LINAGE clause used to define a logical printer page (standard 11-inch form). Vertical height is made up of a top margin, a page body, and a bottom margin. All actual printing is done in the page body, including a commonly required footing area of five lines near the bottom of the page body.

Now equate:

```
          page body to integer-1
            footing to integer-2
         top margin to integer-3
      bottom margin to integer-4
```

the statement would be

```
LINAGE IS 50 LINES,
WITH FOOTING AT 45,
LINES AT TOP 6,
LINES AT BOTTOM 10.
```

RECORD CONTAINS

The RECORD CONTAINS clause specifies the size of the data records.

General Format:

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

Syntax Rules:

1. Data-name-1 must reference an elementary unsigned numeric integer four characters in length.

General Rules:

1. The size of each data record is completely defined within the record description entry; therefore, this clause is never required unless variable length records are desired. If this clause is not included, the size of the data record will be defined as the maximum number of characters in the largest size data record that has been included in the record description entries for the file.

When present, the following items apply:

   a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case, integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

   b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record.

2. For an Indexed File integer-2 cannot be less than four.

3. If integer-1 and integer-2 are both shown, it is implied that the logical records are variable in length, in which case, the following notes apply:

   a. If the DEPENDING clause is not specified, the logical record length is automatically derived by the compiler and supplied as the first four characters of the record when it is written and may not be referenced.

b. If the DEPENDING clause is specified, the logical record length is supplied by the programmer in data-name-1 at the time the record is written and the following notes apply:

   1) Data-name-1 must occupy the first four characters of each record of the file.

   2) The logical record length includes the four characters needed to contain the length.

4. Variable length records are only implemented in files that are described (explicitly or implicitly) with sequential organization. In B 1000 Mark 11.0, remote files may be variable in length.

VALUE OF

The VALUE OF clause specifies the description of an item in the label records associated with a file.

General Format:

```
VALUE OF attribute-name-1 IS { data-name-1 }
                             { literal-1   }

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

Syntax Rules:

1. Data-name-1, data-name-2, and so on, should be qualified when necessary, but cannot be subscripted, indexed, or items described with the USAGE IS INDEX clause.

2. Data-name-1, data-name-2, and so forth, must be in the WORKING-STORAGE SECTION.

3. Attribute-name-1, attribute-name-2, and so on, are discussed in File Attributes, in section 8.

General Rules:

1. At execution time, for an input file, the appropriate label routine checks to see if the value of attribute-name-1 is equal to the value of literal-1, or of data-name-1, whichever has been specified.

   During execution of the program, for an output file, the value of attribute-name-1 is made equal to the value of literal-1, or data-name-1, whichever has been specified.

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

3. During compilation of the program, the compiler makes no attempt to verify the value of data-name-1, data-name-2, etc. Reference the B 1000 System Software Operation Guide, Volume I, for acceptable values.

One of the most common attribute names is TITLE, which specifies the external file name. Examples of the TITLE attribute using the literal option are:

```
VALUE OF TITLE IS "FAMILY/MYFILE ON MYPACK".
VALUE OF TITLE IS "FAMILY/MYFILE".
VALUE OF TITLE IS "MYFILE".
```

When the data-name option is used the clause is:

```
VALUE OF TITLE IS DATA-NAME-1.
```

VALUE OF

DATA-NAME-1 is defined as an alphanumeric data item in the DATA DIVISION. The programmer builds the literal string in the data item (DATA-NAME-1) prior to the OPEN of the file.

```
01 DATA-NAME-1 PIC X(35).
```

The following are examples of statements which may be used in the PROCEDURE DIVISION to create the literal string.

```
MOVE "MYFILE" TO DATA-NAME-1.
MOVE "FAMILY/MYFILE" TO DATA-NAME-1.
MOVE "FAMILY/MYFILE ON MYPACK" TO DATA-NAME-1.

STRING "FAMILY", "/", "MYFILE", DELIMITED BY SIZE
      INTO DATA-NAME-1.

STRING "MYFILE", " ON ", "MYPACK" DELIMITED BY SIZE
      INTO DATA-NAME-1.
```

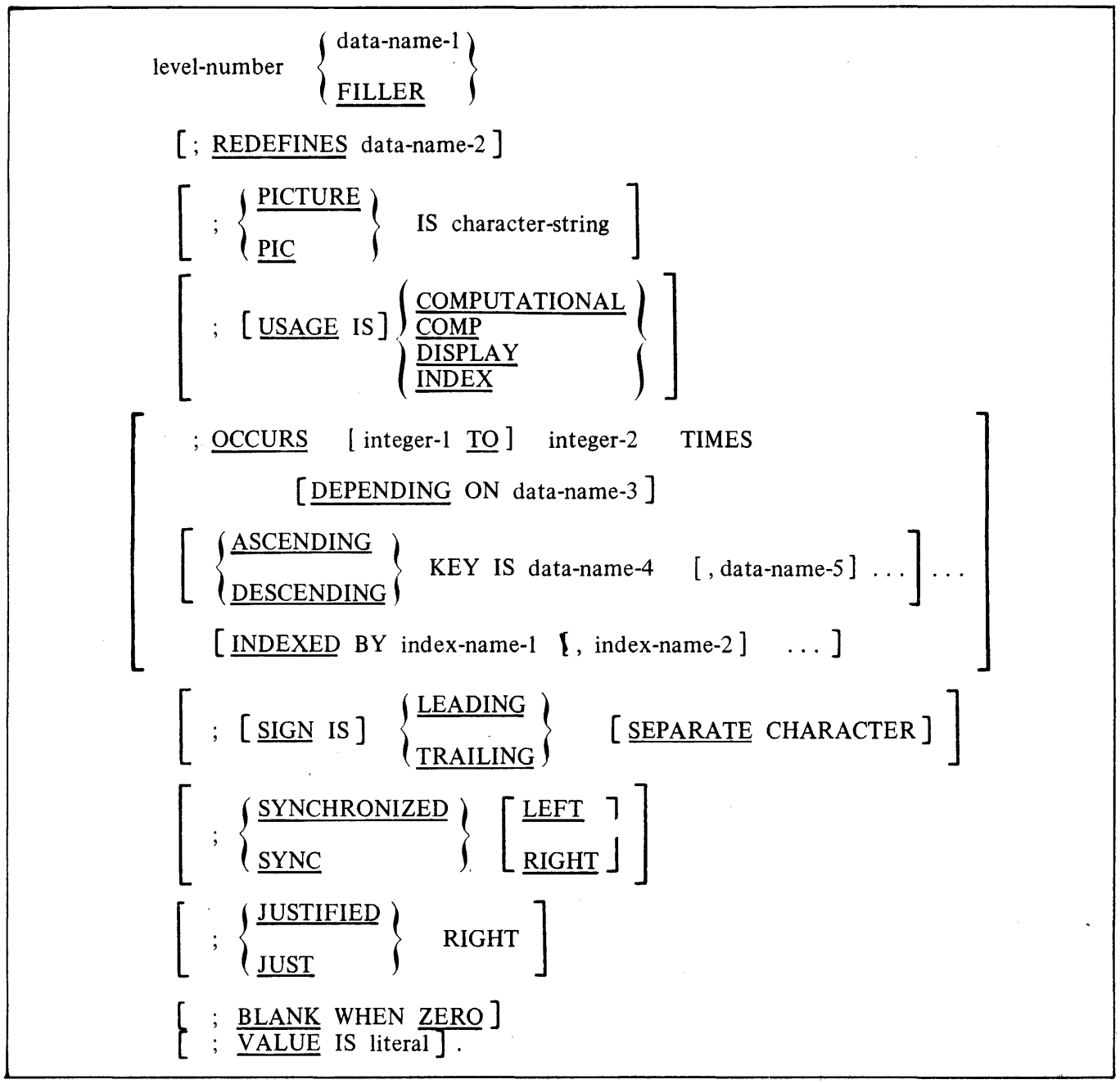
Use of the STRING statement satisfies the requirement that there are no spaces either side of the slash (/) when creating the title of the file.

DATA DESCRIPTION STRUCTURE

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

General Format:

Format 1:



Format 2:

```
66 data-name-1; RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                      { THRU    }
```

Format 3:

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

[ , literal-3 [ { THROUGH } literal-4 ] ] ... .
              { THRU    }
```

Syntax Rules:

1. The level-number in Format 1 may be any number from 01 through 49 or 77.
2. The clauses may be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.
3. The PICTURE clause must be specified for every elementary item except an index data item, in which case, use of this clause is prohibited.
4. The words THRU and THROUGH are equivalent.

General Rules:

1. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses must not be specified except for an elementary data item.

DATA DESCRIPTION STRUCTURE

2. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

   a. Another condition-name.

   b. A level 66 item.

   c. A group item containing items for which USAGE (other than USAGE IS DISPLAY) is specified either explicitly or implicity.

   d. An index data item (Refer to the USAGE IS INDEX clause in this section).

BLANK WHEN ZERO

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

General Format:

BLANK WHEN ZERO

Syntax Rules:

1. The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited. Refer to PICTURE in this section.

General Rules:

1. When the BLANK WHEN ZERO clause is used, the item will contain spaces if the value of the item is zero.
2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

DATA-NAME or FILLER

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

General Format:

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

Syntax Rules:

1. In the FILE, WORKING-STORAGE, COMMUNICATION, and LINKAGE SECTIONS, a data-name or the key word FILLER must be the first word following the level-number in each data description entry.

General Rules:

1. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly. However, the key word FILLER may be used as a conditional variable because such use does not require explicit reference to the FILLER item, but to its value.

JUSTIFIED

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format:

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

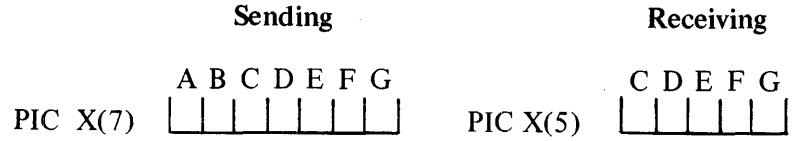
Syntax Rules:

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.
3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

General Rules:

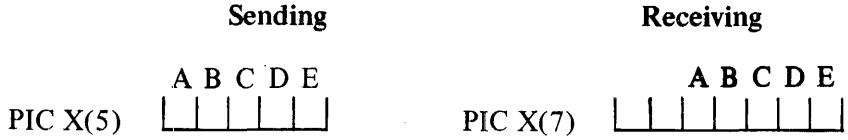
1. When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated.

Example:



When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.

Example:



2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. Refer to Standard Alignment Rules in section 2.

LEVEL-NUMBER

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items, linkage items, condition-names, and the RENAMES clause.

General Format:

level-number

Syntax Rules:

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

2. Data description entries subordinate to an FD, SD, or CD entry must have level-numbers with the values 01 through 49, 66 or 88. Refer to the FD, SD, and CD file descriptions in the paragraphs entitled File Description Structure, Sort-Merge File Description Structure, and Communication Description Structure in this section.

3. Data description entries in the WORKING-STORAGE SECTION and LINKAGE SECTION must have level-numbers with the values 01 through 49, 66, 77 or 88.

General Rules:

1. The level-number 01 identifies the first entry in each record description. Less inclusive groupings are given higher numbers (not necessarily successive) up to a limit of 49.

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

   a. Level-number 77 is assigned to identify noncontiguous working storage data items, and noncontiguous linkage data items, and can be used only as described by Format 1 of the data description structure.

   b. Level-number 66 is assigned to identify RENAMES entries and can be used only as described in Format 2 of the data description structure.

   c. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only as described in Format 3 of the data description structure.

3. Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area.

Examples:

The following is an example of a record layout which corresponds to figure 6-3 showing a record description and the use of level numbers.

STUDENT RECORD				
STUDENT NO.	NAME		GRADE	BIRTH DATE
	Last	First		mth day yr

Burroughs COBOL CODING FORM

PROGRAM Level Numbers | REQUESTED BY | PAGE 3 OF 15
PROGRAMMER Heather | DATE | IDENT. 73 Level1S1 80

PAGE NO. 1 3	LINE NO. 4 6	7	A 8 11	B 12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	Z 72
	01		01	STUDENT-REC.															
	02			03	STUDENT-NO							PIC	9(16).						
	03			03	STUDENT-NAME.														
	04				05	LAST-NAME						PIC	X(8).						
	05				05	FIRST-NAME						PIC	X(5).						
	06			03	GRADE							PIC	99.						
	07				88	FIRST-GRADE		VALUE	1.										
	08				88	SECOND-GRADE		VALUE	2.										
	09				88	THIRD-GRADE		VALUE	3.										
	10																		
	11			03	BIRTH-DATE.														
	12				05	BIRTH-MONTH						PIC	99.						
	13				05	BIRTH-DAY						PIC	99.						
	14				05	BIRTH-YEAR						PIC	99.						
	15																		
	16																		
	17			66	REC-TWO RENAMES STUDENT-NO.														
	18																		
	19			66	REC-THREE RENAMES BIRTH-MONTH THRU BIRTH-DAY.														
	20																		

4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

G12332

Figure 6-3. Level Numbers

OCCURS

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

General Format:

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 ] ... ]
```

Syntax Rules:

1. When both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2.
2. The data description of data-name-1 must describe a positive integer and, therefore, cannot be zero.
3. Data-name-1, data-name-2, data-name-3, ... may be qualified.
4. Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.
5. Data-name-3, and so forth, must be the name of an entry subordinate to the group item which is the subject of this entry.
6. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing.

OCCURS

7. A data description entry that contains Format 2 of the OCCURS clause may only be followed, within that record description, by data description entries which are subordinate to it.

8. The OCCURS clause cannot be specified in a data description entry that:

   a. Has an 01, 66, 77, or 88 level-number.

   b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.

9. In Format 2, the data item defined by data-name-1 must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

10. If data-name-2 is not the subject of this entry, then:

    a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

    b. Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.

    c. There must not be any entry that contains an OCCURS clause between the items identified by the data-names in the KEY IS phrase and the subject of this entry.

11. Index-name-1, index-name-2, ... must be unique words within the program.

12. Integer-2 in Format 1 and integer-1 in Format 2 cannot be zero.

General Rules:

1. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than SEARCH or USE FOR DEBUGGING. Further, if the data-name associated with the OCCURS clause is the name of a group item, then all data-names belonging to the group must be subscripted or indexed when used as operands, except as the object of a REDEFINES clause. Refer to Subscripting, Indexing, and Identifier in section 2.

2. Except for the OCCURS clause, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

3. The number of occurrences of the subject entry is defined as follows:

   a. In Format 1, the value of integer-2 represents the exact number of occurrences.

b. In Format 2, the current value of the data item referenced by data-name-1 represents the number of occurrences.

   Format 2 specifies that the data-name associated with the OCCURS clause has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable. When the OCCURS clause is subordinate to an 01 level-number in the File Description of a file, the RECORD CONTAINS <integer-3> to <integer-4> clause indicates the minimum and maximum number of characters for the variable length records.

   When a table element is referenced, the value of data-name-1 must fall within the range integer-1 through integer-2, inclusive. If the value of data-name-1 is outside this range, the execution of the program is terminated. The value of data-name-1 is used to determine the last table element that may be referenced. When the value of data-name-1 is less than integer-2, the data items whose occurrence numbers exceed the value of data-name-1 are inaccessible. Reducing the value of the data item referenced by data-name-1 has no effect upon the contents of data items whose occurrence numbers now exceed the value of the data item referenced by data-name-1.

4. When a group item that has a subordinate entry that specifies Format 2 of the OCCURS clause is referenced, only that part of the table area that is specified by the value of data-name-1 is used in the operation.

5. The KEY IS phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, and so forth. The ascending or descending order is determined according to the rules for comparison of operands (refer to Relation Condition in section 7 for the Comparison of Numeric Operands and the Comparison of Nonnumeric Operands). The data-names are listed in descending order of significance.

Examples:

The following data descriptions illustrate the use of an OCCURS clause to provide equivalent data mapping with fewer data names.

```
01 DATA-REC.
    03 AMOUNT-ONE       PIC 9(8).
    03 AMOUNT-TWO       PIC 9(8).
    03 AMOUNT-THREE     PIC 9(8).
    03 DESC-ITEM        PIC X(20).
    03 DESC-NO          PIC 9(5).
```

```
01 DATA-REC.
    03 AMOUNT       PIC 9(8)
           OCCURS 3 TIMES.
    03 DESC-ITEM    PIC X(20).
    03 DESC-NO      PIC 9(5).
```

OCCURS

This example illustrates a use of the OCCURS clause to provide nested descriptions. A reference to ITEM-4 requires the use of three levels of subscripting, for example, ITEM-4 (2, 5, 4). A reference to ITEM-3 requires two subscripts, for example, ITEM-3 (I,J).

```
01  TABLE-REC.
    03  ITEM    OCCURS 2 TIMES.
        05  ITEM-1                  PIC X(10).
        05  ITEM-2  OCCURS 5 TIMES.
            07  ITEM-3              PIC X(4).
            07  ITEM-4  OCCURS 5 TIMES.
                09  ITEM-5          PIC XX.
                09  ITEM-6          PIC 99.
```

PICTURE

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

General Format:

```
{PICTURE}
{PIC    }  IS character-string
```

Syntax Rules:

1. A PICTURE clause can be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL74 character set used as symbols. The allowable combinations determine the category of the elementary item.
3. The maximum number of characters allowed in the character-string is 30.
4. The PICTURE clause must appear in every elementary item except those items whose USAGE is declared as INDEX.
5. PIC is an abbreviation for PICTURE.
6. The asterisk, when used as the zero suppression symbol, and the clause BLANK WHEN ZERO may not appear in the same entry.

General Rules:

1. There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.
2. To define an item as alphabetic:
   a. The PICTURE character-string can only contain the symbols 'A', 'B'.
   b. The item contents, when represented in standard data format, must be any combination of the 26 letters of the English alphabet and the space from the COBOL74 character set.
3. To define an item as numeric:
   a. The PICTURE character-string can only contain the symbols '9', 'P', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive.

PICTURE

b. If unsigned, the item contents, when represented in standard data format, must be a combination of the numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign. Refer to the SIGN clause in this section.

4. To define an item as alphanumeric:

   a. The PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item.

   b. The item contents, when represented in standard data format, are allowable characters in the B character set.

5. To define an item as alphanumeric edited:

   a. The PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and '/'.

      1) The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X' or at least one '/' (stroke) and at least one 'X'.

      2) The character-string must contain at least one '0' (zero) and at least one 'A' or at least one '/' (stroke) and at least one 'A'.

   b. The contents, when represented in standard data format, are allowable characters in the computer character set.

6. To define an item as numeric edited:

   a. The PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', 'P', 'V', 'Z', "0", "9", ",", ".", "*", "+", "-", "CR", "DB", and the currency symbol ($). The allowable combinations are determined from the order of precedence of symbols and the editing rules.

      1) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive.

      2) The character-string must contain at least one '0', 'B', '/', 'Z', '*', '+', ',', '.', '-', 'CR', 'DB', or currency symbol.

   b. The contents of the character positions of these symbols that are allowed to represent a digit in standard data format must be one of the numerals.

7. The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An integer which is enclosed in parentheses following the symbols: "A", ",", "X", "9", "P", "Z", "*", 'B', '/', '0', '+', '-', or '$' indicates the number of consecutive occurrences of the symbol. The following symbols may appear only once in a given PICTURE: "S", "V", '.', 'CR', and 'DB'.

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

'A'

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

'B'

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

'P'

Each letter 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items. The scaling position character P can appear only to the left or right as a continuous string of P's within a PICTURE description; since the scaling position character P implies an assumed decimal point (to the left if P's are leftmost PICTURE characters and to the right if 'P's are rightmost PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description. The character 'P' and the insertion character '.' (period) cannot both occur in the same PICTURE character-string. If, in any operation involving conversion of data from one form of internal representation to another, the data item being converted is described with the PICTURE character 'P', each digit position described by a 'P' is considered to contain the value zero, and the size of the data item is considered to include the digit positions so described.

'S'

The letter 'S' is used in a character-string to indicate the presence of an operational sign in the internal representation of a numeric data item. It must be the first (leftmost) character in the character-string and may be used in the PICTURE character-string of any data item whose USAGE is DISPLAY or COMPUTATIONAL. The SIGN clause may be used to specify the exact representation and position of the operational sign. Refer to the SIGN clause in this section for additional information.

When an operational sign is specified for a DISPLAY data item and a SIGN clause is not specified, the sign is maintained and expected in the zone of the most significant (leftmost) character. When the data item is the receiving field in an arithmetic statement and since the native character set is EBCDIC, the four zone bits are set to binary 1101 for negative values and to binary 1100 for positive values. When the data item is used in an algebraic comparison or operation to supply an algebraic value, only the most significant zone being a binary 1101 will cause the value of the data item to be considered negative. Only the zone values 1100 and 1101 will qualify the data item as being NUMERIC if tested by the NUMERIC class condition. For DISPLAY data items, the presence or absence of an operational sign has no effect upon the amount of storage required to contain the data item.

When an operational sign is specified for a COMPUTATIONAL data item and a SIGN clause is not specified, the sign is maintained and expected as a leading separate 4-bit character to the left of the most significant digit position. Since the native character set is EBCDIC, the binary pattern of the sign character is 1101 for negative values and 1100 for positive values. Like DISPLAY data items, only these values allow the item to be considered NUMERIC in the class condition test. Unlike DISPLAY data items, the specification of an operational sign for COMPUTATIONAL data items increases by one the number of 4-bit character positions occupied by the data item in storage.

PICTURE

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

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

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

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

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

'/'
Each stroke '/' in the character-string represents a character position into which the stroke character is inserted. The '/' is counted in the size of the item.

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

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

'+' '-' 'CR' 'DB'
These symbols are used as editing sign control symbols and when used, represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

'*'

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

'cs'

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

Editing Rules:

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

   a. Simple insertion

   b. Special insertion

   c. Fixed insertion

   d. Floating insertion

   There are two types of suppression and replacement editing:

   a. Zero suppression and replacement with spaces

   b. Zero suppression and replacement with asterisks

2. The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. Table 6-1 specifies which type of editing may be performed upon a given category:

Table 6-1. Editing for Each Item Category

Category	Type of Editing
Alphabetic	Simple insertion 'B' only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B', and '/'
Numeric Edited	All, subject to Editing Rule 3

3. Floating insertion, and editing by zero suppression and replacement, are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

PICTURE

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

   Any character which is not a defined PICTURE character may be used as a simple insertion character. Precedence rules for these characters and rules for determining category are the same as those for the simple insertion character "0".

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

6. Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. The symbols 'CR' or 'DB' always represent two character positions in determining the size of the item and must represent the rightmost character positions counted in the size of the item. The symbol '+' or '-', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as in the PICTURE character-string. Editing sign control symbols produce the results shown in table 6-2 depending upon the value of the data item.

Table 6-2. Editing of Sign Control Symbols

Editing Symbol in PICTURE Character-String	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	–
–	space	–
CR	2 spaces	CR
DB	2 spaces	DB

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

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right. These simple insertion characters are part of the floating string.

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

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

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character is placed into the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

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

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

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

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

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

PICTURE

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

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

The leftmost suppression symbol ('Z' or '*') in the PICTURE character-string can only be preceded by the currency symbol or the sign symbols ('+' or '-').

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

Precedence Rules

Figure 6-4 shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol 'cs'.

At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or 'cs' must be present in a PICTURE string.

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ −}	{+ −}	{CR DB}	cs	{Z *}	{Z *}	{+ −}	{+ −}	cs	cs	9	A X	S	V	P	P
Non-Floating Insertion Symbols	B	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	0	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	/	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	,	X	X	X	X	X	X			X	X	X	X	X	X	X	X			X		X
	.	X	X	X	X		X			X	X		X		X		X					
	{+ −}																					
	{+ −}	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	{CR CB}	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	cs						X															
Floating Insertion Symbols	{Z *}	X	X	X	X		X			X	X											
	{Z *}	X	X	X	X	X	X			X	X	X								X		X
	{+ −}	X	X	X	X					X			X									
	{+ −}	X	X	X	X	X				X			X	X						X		X
	cs	X	X	X	X		X								X							
	cs	X	X	X	X	X	X								X	X				X		X
Other Symbols	9	X	X	X	X	X	X			X	X		X		X		X	X	X	X		X
	A X	X	X	X													X	X				
	S																					
	V	X	X	X	X		X			X	X		X		X		X		X		X	
	P	X	X	X	X		X			X	X		X		X		X		X		X	
	P						X			X									X	X		X

G12333

Figure 6-4. PICTURE Character Precedence Chart

PICTURE

Nonfloating insertion symbols '+' and '-', floating insertion symbols "Z", "*", "+", "-", and "cs", and the other symbol "P" appear twice in the PICTURE character precedence chart, figure 6-4. The leftmost column and uppermost row for each symbol represent use on the left of the decimal point position. The second appearance of the symbol in the chart represents use to the right of the decimal point position.

Examples:

Table 6-3 demonstrates the editing function of the PICTURE Clause.

Table 6-3. Editing Application of the PICTURE Clause

Source Area		Receiving Area	
Picture	Data	Editing Picture	Edited Data
9(5)	12345	$ZZ,ZZ9.99	$12,345.00
V9(5)	12345	$$$,$$9.99	$0.12
V9(5)	12345	$ZZ,ZZ9.99	$ 0.12
9(5)	00000	$$$,$$9.99	$0.00
9(3)V99	12345	$ZZ,ZZ9.99	$ 123.45
9(5)	00000	$$$,$$$.$$	
9(5)	01234	$**,**9.99	$*1,234.00
9(5)	00000	$**,***.**	*******.**
9(5)	00123	$**,**9.99	$***123.00
9(3)V99	00012	$ZZ,ZZ9.99	$ 0.12
9(3)V99	12345	$$$,$$9.99	$123.45
9(3)V99	00001	$ZZ,ZZZ.99	$ .01
9(5)	12345	$$$,$$9.99	$12,345.00
9(5)	00000	$ZZ,ZZZ.ZZ	
9(3)V99	00001	$$$,$$$.$$	$.01
S9(5)	(+) 12345	ZZZZ9.99+	12345.00+
S9(5)	(-) 00123	--99999.99	-00123.00
9(3)V99	12345	999.00	123.00
S9(5)	(-) 12345	ZZZZ9.99-	12345.00-
S9(5)	(+) 12345	ZZZZ9.99-	12345.00
9(5)	12345	BBB99.99	45.00
S9(5)V	(-) 12345	-ZZZZ9.99	-12345.00
S9(5)	(-) 12345	$$$$$$.99CR	$12345.00CR
S99V9(3)	(-) 12345	------.99	-12.34
S9(5)	(+) 12345	$$$$$$.99CR	$12345.00
9(3)V99	12345	999.BB	123.
9(5)	12345	00999.00	00345.00
V9(5)	12345	PPP99	00045
9(5)	12345	999PP	12300

REDEFINES

The REDEFINES clause allows the same computer storage area to be described by different data de-
entries.

General Format:

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

NOTE
Level-number, data-name-1 and the semicolon are shown in the general for-
to improve clarity. Level-number and data-name-1 are not part of the
REDEFINES clause.

Syntax Rules:

1. The REDEFINES clause, when specified, must immediately follow data-name-1.

2. The level-numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88.

3. This clause must not be used in level 01 entries in the FILE SECTION. Refer to the DATA RECORDS clause, General Rule 2, in this section.

4. The REDEFINES clause must not be used in level 01 entries in the COMMUNICATION SEC-

5. The data description entry for data-name-2 cannot contain a REDEFINES clause. Data-name-2 may be subordinate to an entry which contains a REDEFINES clause. The data description entry for data-name-2 cannot contain an OCCURS clause. However, data-name-2 may be sub-
to an item whose data description entry contains an OCCURS clause. In this case,
the reference to data-name-2 in the REDEFINES clause may not be subscripted or indexed. Nei-
the original definition nor the redefinition can include an item whose size is variable
as defined in the OCCURS clause. Refer to the OCCURS clause in this section.

6. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 may occur between the data description entries of data-name-2 and data-name-1.

REDEFINES

General Rules:

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

2. The REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area. Therefore, the usage of data-name-1 need not be the same as the usage of data-name-2, except that DISPLAY or group data items may not redefine elementary COMPUTATIONAL or INDEX data items which do not begin on a byte boundary. When redefinition occurs at other than the 01 level, the amount of storage allocated for data-name-2 must be the same as the amount of storage implied by the declared size and usage of data-name-1, with the following exceptions:

   a. A DISPLAY or group data item may redefine an elementary COMPUTATIONAL data item that begins, but does not end, on a byte boundary if the difference in size can be accounted for by the generation of a 4-bit filler so that the redefining item ends on a byte boundary.

   b. A DISPLAY or group data item may be redefined by an elementary COMPUTATIONAL data item even though the actual size (including sign position, if described) is one 4-bit character less than the number of 4-bit characters in the storage area. The redefining item is aligned to begin on a byte boundary and end at the middle of the last byte of storage.

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

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

5. Multiple 01 level entries subordinate to any given level indicator represent implicit redefinitions of the same area.

Example:

The following example illustrates the uses of the REDEFINES clause.

```
01  WHOLE-ITEM.
    03  PART-ONE            PIC X(40).
    03  PART-TWO REDEFINES PART-ONE.
        05  A               PIC X(20).
        05  B               PIC X(20).
        05  B2 REDEFINES B  PIC 9(20).
    03  PART-THREE REDEFINES PART-ONE  PIC X(40).
    03  PART-FOUR  REDEFINES PART-ONE  PIC 9(40).
```

RENAMES

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

General Format:

```
66 data-name-1; RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                     { THRU    }
```

NOTE
Level-number 66, data-name-1, and the semicolon are shown in the general format to improve clarity. Level-number and data-name-1 are not part of the RENAMES clause.

Syntax Rules:

1. All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.

2. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same 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, 88, or 01 level entry.

3. Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the asso- level 01, FD, or SD entries. Neither data-name-2 nor data-name-3 may have an OC- clause in the data description entry nor be subordinate to an item that has an OC- clause in the data description entry. Refer to the OCCURS clause in this section.

4. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. Data-name-3, therefore, cannot be subordinate to data-name-2.

5. Data-name-2 and data-name-3 may be qualified.

6. The words THRU and THROUGH are equivalent.

7. None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in the OCCURS clause.

General Rules:

1. One or more RENAMES entries can be written for a logical record.
2. When data-name-3 is specified, data-name-1 is 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).

RENAMES

When data-name-3 is specified: If data-name-2 is an elementary COMPUTATIONAL or IN- data item, it must be positioned to begin at an 8-bit character boundary; if data-name-3 is an elementary COMPUTATIONAL or INDEX data item, it must be positioned to end at the end of an 8-bit character boundary.

3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

   Data-name-1 assumes all the characteristics of data-name-2 as determined from the data de- of data-name-2, including usage, justification, synchronization, editing require- and so on.

Examples:

The following are examples of the RENAMES clause.

```
01  PART-ONE.
    03  SUB-PART-ONE.
        05  SP-1                PIC X.
        05  SP-2                PIC X(5).
        05  SP-3                PIC XXX.
        05  SP-4                PIC X.

    03  SUB-PART-TWO.
        05  SP2-1               PIC XX.
        05  SP2-2               PIC X(10).
        05  SP2-3               PIC XXX.

66  SUB-PT-THREE RENAMES SUB-PART-ONE.
66  SUB-PT-FOUR  RENAMES SUB-PART-ONE.
66  SUB-PT-5 RENAMES SP-2 THRU SP-4.
66  SUB-PT-6 RENAMES SP-3 THROUGH SP2-2.
66  SUB-PT-7 RENAMES SP-1 THRU SUB-PART-TWO.
```

SIGN

The SIGN clause specifies the position and the mode of representation of the operational sign when necessary to describe these properties explicitly.

General Format:

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

Syntax Rules:

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

2. The numeric data description entries to which the SIGN clause applies must be described as usage DISPLAY or COMPUTATIONAL.

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

4. If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

General Rules:

1. The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'; the 'S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

2. A numeric data description entry whose PICTURE contains the character 'S', but to which no optional SIGN clause applies, has an operational sign which is positioned and represented according to the standard default position and representation of operational signs. Refer to the PICTURE clause, General Rule 8 (the 'S' symbol), in this section.

3. If a SIGN clause, without a SEPARATE CHARACTER phrase, applies to a numeric data description entry, then:

   a. If the usage of the data item is DISPLAY, the operational sign is maintained and expected as a binary 1100 or 1101 in the zone of the LEADING or TRAILING character, and will not cause additional storage to be allocated for the data item.

b. If the usage of the data item is COMPUTATIONAL, the operational sign is maintained and expected as a binary 1100 or 1101 LEADING or TRAILING character, increasing by one 4-bit character the amount of storage allocated for the data item above that which would be allocated for an unsigned COMPUTATIONAL data item. The presence or absence of the SEPARATE CHARACTER phrase has no effect upon the position or representation of the operational sign for COMPUTATIONAL data items.

4. If a SIGN clause with a SEPARATE CHARACTER phrase applies to a numeric data description entry, then:

   a. If the usage of the data item is DISPLAY, the operational sign is maintained and expected as a LEADING or TRAILING character separate from, and in addition to, the numeric character positions. The operational sign for negative values is the character "-" and for positive values the character "+".

   b. If the usage of the data item is COMPUTATIONAL, the rules listed above in General Rule 3b apply.

5. Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

Examples:

The following are examples of the SIGN clause.

```
WORKING-STORAGE SECTION.
77  NUMB1       PIC S9(11) VALUE ZEROES SIGN IS TRAILING.
77  NUMB2       PIC S9(6)  VALUE ZEROES SIGN IS LEADING.


01  GROUP-ITEM  SIGN IS LEADING.
    03  ITEM-1     PIC X(15).
    03  ITEM-2     PIC S9(4) COMP.
    03  ITEM-3     PIC X(25).
    03  ITEM-4     PIC S9999.
```

SYNCHRONIZED

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

General Format:

```
{SYNCHRONIZED}  [LEFT ]
{SYNC        }  [RIGHT]
```

Syntax Rules:

1. This clause may only appear with an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.

General Rules:

1. This clause specifies that the subject data item is to be aligned in the computer to a byte boundary. If the previous data item did not end on a byte boundary an implicit 4-bit FILLER is generated. This unused filler digit is included in:
   a. The size of any group item(s) to which the elementary item belongs.
   b. Any redefinition of the previous data item when it is the object of a REDEFINES clause. Refer to the REDEFINES and the USAGE clauses in this section.
2. Neither RIGHT nor LEFT following SYNCHRONIZED has effect on the positioning of the data item. RIGHT and LEFT are handled as comment entries.
3. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justification, truncation, or overflow.
4. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position.
5. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:
   a. Each occurrence of the data item is SYNCHRONIZED.
   b. Any implicit FILLER generated for other data items within that same table are generated for each occurrence of those data items.

USAGE

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

General Format:

```
[USAGE IS] { COMPUTATIONAL }
           { COMP          }
           { DISPLAY       }
           { INDEX         }
```

Syntax Rules:

1. The PICTURE character-string of a COMPUTATIONAL item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', and one or more 'P's. Refer to the PICTURE clause in this section.

2. COMP is an abbreviation for COMPUTATIONAL.

3. The USAGE clause may be written in any data description entry with a level number other than 66 or 88.

4. If the USAGE is written in the data description entry for a group item, it may also be written in the data description entry for any subordinate elementary item or group item, but the same USAGE must be specified by both entries. Items of different USAGE may appear in the same record.

5. An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a PROCEDURE DIVISION header, or the USING phrase of a CALL statement.

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

General Rules:

1. If the USAGE clause is written at a group level, it applies to each elementary item in the group.

2. This clause specifies the manner in which a data item is represented in the storage of a 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 data item.

   DISPLAY data items are represented internally as contiguous 8-bit characters represented in the native character set, EBCDIC.

   Elementary COMPUTATIONAL data items are represented internally as contiguous 4-bit characters or digits.

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

   The group item is considered to be a group data item whose class is alphanumeric and whose usage is DISPLAY and may be referenced any place in the syntax acceptable for such an item. The size of the group item is considered to be in terms of DISPLAY characters aligned according to the rules for DISPLAY, one character for every two 4-bit characters or digits that form a part of it.

4. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.

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

6. Every occurrence of a DISPLAY data item begins and ends on a byte boundary. Within a record description, the declaration of a DISPLAY data item immediately following a COMPUTATIONAL or INDEX data item that does not end on a byte boundary causes an automatic generation of a 4-bit filler between the two items. This filler area between the two data items is not included in the size of either item, but is included in the size of all group items to which the two items are subordinate. Similarly, if the last item declared within a group item at the next lowest hierarchical level is a COMPUTATIONAL or INDEX data item that does not end on a byte boundary, an automatic generation of a 4-bit filler occurs. This filler is included in the size of the group item.

7. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. If a group item is described with the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group is not an index data item and cannot be used in the SEARCH or SET statement.

   The group item is considered to be a group data item whose class is alphanumeric and whose usage is DISPLAY and may be referenced any place in the syntax acceptable for such an item. The size of the group item is considered to be in terms of DISPLAY characters, four characters for each subordinate index data item.

8. An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion takes place.

9. An index data item may contain a signed value. An index data item occupies the same space and has the same alignment as an item declared PICTURE S9(7) USAGE IS COMPUTATIONAL.

VALUE

The VALUE clause defines the value of constants, the initial value of working-storage items, the initial value of data items in the COMMUNICATION SECTION, and the values associated with a condition-name.

General Format:

Format 1:

```
VALUE IS    literal
```

Format 2:

```
{VALUE IS   }
{VALUES ARE } literal-1 [ { THROUGH } literal-2 ]
                        { THRU    }
[ , literal-3 [ { THROUGH } literal-4 ] ] ...
              { THRU    }
```

Syntax Rules:

1. The words THRU and THROUGH are equivalent.
2. The VALUE clause cannot be stated for any item whose size is variable. Refer to the OCCURS clause in this section.
3. A signed numeric literal must have an associated signed numeric PICTURE character-string.
4. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

General Rules:

1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:
   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working-storage item, the literal is aligned in the data item according to the standard alignment rules. Refer to Standard Alignment Rules in section 2.

b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (Refer to Standard Alignment Rules in section 2.) Editing characters in the PICTURE clause are included in determining the size of the data item (refer to the PICTURE clause in this section) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.

c. Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

2. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

Condition-Name Rules

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of the conditional variable.

2. Format 2 can only be used in connection with condition-names. Refer to Condition-Name in section 2. Wherever the THRU phrase is used, literal must be less than literal-2, literal-3 less than literal-4, and so forth.

Data Description Entries Other Than Condition-Names

1. Rules governing the use of the VALUE clause differ with the respective sections of the DATA DIVISION:

   a. In the FILE SECTION, the VALUE clause may be used only in condition-name entries.

   b. In the WORKING-STORAGE SECTION and the COMMUNICATION SECTION, the VALUE clause must be used in condition-name entries. The VALUE clause may also be used to specify the initial value of any other data item, in which case the clause 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 is undefined.

   c. In the LINKAGE SECTION, the VALUE clause may be used only in condition-name entries.

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

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

VALUE

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

5. The VALUE clause must not be written for a group containing items for which USAGE (other than USAGE IS DISPLAY) is specified either explicitly or implicity.

WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is optional and is that part of the DATA DIVISION set aside for intermediate processing of data. The difference between the WORKING-STORAGE and FILE SECTIONS is that the former deals with data that is not associated with an input or output file. All clauses which are used in normal input or output record descriptions can be used in a WORKING-STORAGE record description.

WORKING-STORAGE Structure

Whereas the FILE SECTION is composed of file description (FD or SD) entries and associated record description entries and noncontiguous items, the WORKING-STORAGE SECTION is composed only of record description entries and noncontiguous items. The WORKING-STORAGE SECTION begins with a section-header and a period, followed by data description entries for noncontiguous WORKING-STORAGE items, and/or record description entries for WORKING-STORAGE records.

Each WORKING-STORAGE SECTION record name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

General Format:

```
WORKING-STORAGE SECTION.
77  data-name-1
    88  condition-name-1
          .
          .
77  data-name-n
01  data-name-2
    02  data-name-3

          .
          .
    66  data-name-m RENAMES data-name-3
01  data-name-4
    02  data-name-5
        03  data-name-n
            88  condition-name-2
```

Noncontiguous WORKING-STORAGE

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

The following record description clauses are required in each entry:

Level-number 77
Data-name
The PICTURE clause or the USAGE IS INDEX clause.

The OCCURS clause is not meaningful on a 77 level item and causes an error at compilation time if used. Other data description clauses are optional and can be used to complete the description of the item if necessary.

WORKING-STORAGE Records

Data elements and constants in WORKING-STORAGE which have a definite hierarchic relationship to one another must be grouped into records according to the rules for the formation of record descriptions. All clauses which are used in normal input or output record descriptions can be used in a WORKING-STORAGE description, including REDEFINES, OCCURS, and COPY.

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 with the data item. The initial value of any index data item is unpredictable.

Condition-Names

Any WORKING-STORAGE item may be a conditional variable with which one or more condition-names are associated. Entries defining condition-names must immediately follow the conditional variable entry. Both the conditional variable entry and the associated condition-name entries may contain VALUE clauses.

CODING THE WORKING-STORAGE SECTION

Figure 6-5 illustrates the coding of the WORKING-STORAGE SECTION.

Burroughs COBOL CODING FORM

PROGRAM WORKING-STORAGE SECTION | REQUESTED BY | PAGE 13 OF 56
PROGRAMMER JOHNNY | DATE | IDENT. 73 WRKING 80

```
PAGE NO. 1 3 | LINE NO. 4 6 | 7 | A 8 11 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 | Z
01  WORKING-STORAGE SECTION.
02  01  HDG-LINE.
03      03  FILLER  PIC A(52) VALUE SPACES.
04      03  FILLER  PIC A(17) VALUE "SALES PERFORMANCE".
05
06  77  DISK-CONTROL  PICTURE 9(8) COMPUTATIONAL.
07  77  TOTAL-SALES  PIC  9(11) VALUE ZERO.
08  77  SALES-QUOTA  PIC  9(10).
09
10  01  STATE-TABLE.
11      05  STATES.
12          10  CALIF  PIC 9999.
13          10  NEVADA  PIC 9(A).
14          10  ORE  PIC 9(A).
15
16      05  STATE-KEY REDEFINES STATES OCCURS 3.
17          10  STATE-CODE  PIC 99.
18          10  COUNTY  PIC 9.
19          10  CITY  PIC 9.
20
```

G12334

Figure 6-5. Coding the WORKING-STORAGE SECTION

LINKAGE SECTION

The LINKAGE SECTION is used for Inter-Program Communication to provide 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 the ability for both programs to have access to the same data items.

The LINKAGE SECTION is an optional part of the DATA DIVISION. It is used when the object program is to be a called program and the CALL statement in the calling program contains a USING phrase.

The LINKAGE 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-names 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 indices.

LINKAGE SECTION Structure

The structure of the LINKAGE SECTION is the same as that previously described for the WORKING-STORAGE SECTION. It begins with a section header, followed by data description entries for noncontiguous data items and/or record description entries. But, each 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 may be referenced within the PROCEDURE DIVISION of the called program only if 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. Condition-names and/or index-names associated with such data-names and/or subordinate data items, may also be referenced in the PROCEDURE DIVISION.

General Format:

```
LINKAGE SECTION.
77  data-name-1
    88  condition-name-1
          .
          .
77  data-name-n
01  data-name-2
    02  data-name-3

          .
          .
    66  data-name-m RENAMES data-name-3
01  data-name-4
    02  data-name-5
        03  data-name-n
            88  condition-name-2
```

Noncontiguous LINKAGE Storage

Items in the LINKAGE SECTION that have no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary 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 or the USAGE IS INDEX 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 which have a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause which is 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).

CODING THE LINKAGE SECTION

Figure 6-6 illustrates the coding of the LINKAGE SECTION.

Burroughs COBOL CODING FORM

PROGRAM Linkage Section | REQUESTED BY | PAGE 5 OF 20
PROGRAMMER Elizabeth | DATE | IDENT. 73 LINKSECT 80

PAGE NO. 1 3	LINE NO. 4 6	7	A 8 11	B 12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	Z 72
	01		LINKAGE	SECTION.															
	02		77	PGM1-ITEM							PIC	X(15).							
	03		77	PGM2-ITEM							PIC	X(10).							
	04																		
	05		01	PGM3-ITEMS.															
	06			03	PGM3-A						PIC	X(2).							
	07			03	PGM3-B						PIC	X(7).							
	08			03	PGM3-C						PIC	X(40).							
	09																		
	10																		
	11		77	PGM4-ITEM							PIC	X(30).							
	12		77	PGM3-DUMMY							PIC	X(25).							
	13																		
	14		01	PGM4-DATA.															
	15			03	PGM4-NAME						PIC	X(16).							
	16			03	PGM4-AGE						PIC	X(2).							
	17																		
	18																		
	19																		
	20																		

4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

G12335

Figure 6-6. Coding the LINKAGE SECTION

Examples:

The following two programs, IPCCALL and STATE, are sample programs using Inter-Program Communication (IPC). The calling program, IPCCALL, through the use of the CALL statement, passes a data item to STATE. The called program, STATE, has a LINKAGE SECTION, and the USING phrase specified in the PROCEDURE DIVISION heading in which to accept the shared data. When the program STATE has completed and executes the EXIT PROGRAM statement, control and the new value of the data item is returned to IPCCALL.

```
BURROUGHS B1800/B1700 COBOL74 COMPILER, MARK IX.0.1 (05/07/80 16:27)
IPCCALL

000200    IDENTIFICATION DIVISION.
000300    PROGRAM-ID.     IPCCALL.
000400    AUTHOR.         MAURA ALFORD.
000500    DATE-WRITTEN.   04/28/80.
000600    DATE-COMPILED.    1980 JUNE  27      11:04
000700    SECURITY. NONE.
000800  * REMARKS.  I CALL A PROGRAM CALLED STATE.
000900    ENVIRONMENT DIVISION.
001000    CONFIGURATION SECTION.
001100    SOURCE-COMPUTER. B1985.
001200    OBJECT-COMPUTER. B1985.
001600    DATA DIVISION.
002200    WORKING-STORAGE SECTION.
002300
002400    01  U-STATE-AREA.
002500        05  U-STATE.
002510            10  U-NUMERIC-STATE      PIC 99.
002520        05  U-STATE-NAME             PIC X(20).
002530        05  U-STATE-RETURN-STATUS    PIC 9.
002600
002700    PROCEDURE DIVISION.
002800    BEGIN.
002900        MOVE  "AL" TO U-STATE.
003000        CALL "STATE" USING U-STATE-AREA.
003100        DISPLAY "U-STATE-AREA = " U-STATE-AREA.
003900        MOVE  "CT" TO U-STATE.
004000        CALL "STATE" USING U-STATE-AREA.
004100        DISPLAY "U-STATE-AREA = " U-STATE-AREA.
004200        CANCEL "STATE".
004400        STOP RUN.
```

LINKAGE SECTION

```
BURROUGHS B1800/B1700 COBOL74 COMPILER. MARK IX.0.1 (05/07/80 16:27)
STATE

000100   IDENTIFICATION DIVISION.
000200   PROGRAM-ID.        STATE.
000300   AUTHOR.             TIC.
000310 * REMARKS.  I AM CALLED BY A PROGRAM NAMED IPCCALL.
000400   ENVIRONMENT DIVISION.
000500   CONFIGURATION SECTION.
000600     SOURCE-COMPUTER. B-1855.
000700     OBJECT-COMPUTER. B-1855.
000800   DATA DIVISION.
000900   WORKING-STORAGE SECTION.
001000   01  NUMBER-OF-STATE-CODES-IN-TABLE    PIC 9(2)  COMP VALUE 10.
001100   01  WS-STATE-ABBREV-AND-NAMES.
001200       03  STATE-01 PIC X(23) VALUE "AL-ALABAMA              ".
001300       03  STATE-02 PIC X(23) VALUE "AK-ALASKA               ".
001400       03  STATE-03 PIC X(23) VALUE "AZ-ARIZONA              ".
001500       03  STATE-04 PIC X(23) VALUE "AR-ARKANSAS             ".
001600       03  STATE-05 PIC X(23) VALUE "CA-CALIFORNIA           ".
001700       03  STATE-06 PIC X(23) VALUE "CO-COLORADO             ".
001800       03  STATE-07 PIC X(23) VALUE "CT-CONNECTICUT          ".
001900       03  STATE-08 PIC X(23) VALUE "DE-DELAWARE             ".
002000       03  STATE-09 PIC X(23) VALUE "DC-DISTRICT OF COLUMBIA".
002100       03  STATE-10 PIC X(23) VALUE "FL-FLORIDA              ".
002110
003700   01  WS-STATE-TABLE REDEFINES WS-STATE-ABBREV-AND-NAMES.
003800       03  WS-STATE  OCCURS 10 TIMES
003900                     INDEXED BY WS-STATE-NUMBER.
004000           05  WS-STATE-ABBREVIATION          PIC X(2).
004100           05  FILLER                         PIC X(1).
004200           05  WS-STATE-NAME                  PIC X(20).
004210
004300   LINKAGE SECTION.
004400    01  U-STATE-AREA.
004500        05  U-STATE.
004600            10  U-NUMERIC-STATE     PIC 9(2).
004700        05  U-STATE-NAME            PIC X(20).
004800        05  U-STATE-RETURN-STATUS   PIC 9(1).
004810
004900   PROCEDURE DIVISION USING U-STATE-AREA.
005000
005100   000-ENTRY.
005200       MOVE ZERO TO U-STATE-RETURN-STATUS.
005300       MOVE SPACES TO U-STATE-NAME.
005400       IF U-STATE IS NUMERIC
005500           PERFORM 010-CONVERT-NUMERIC-CODE
005600         ELSE
005700           PERFORM 020-CONVERT-ALPHA-CODE.
005800
005900   005-EXIT-PROGRAM.
006000       EXIT PROGRAM.
006100
006200   010-CONVERT-NUMERIC-CODE.
006300       IF U-NUMERIC-STATE > ZERO AND
006400          NOT > NUMBER-OF-STATE-CODES-IN-TABLE
006500           SET WS-STATE-NUMBER TO U-NUMERIC-STATE
006600           MOVE WS-STATE-ABBREVIATION (WS-STATE-NUMBER)
006700                TO U-STATE
006800           MOVE WS-STATE-NAME (WS-STATE-NUMBER) TO U-STATE-NAME
006900         ELSE
007000           IF U-NUMERIC-STATE = 99
007100               MOVE "XX" TO U-STATE
007200             ELSE
007300               MOVE 1 TO U-STATE-RETURN-STATUS.
007400
007500   020-CONVERT-ALPHA-CODE.
007600       IF U-STATE = "XX"
007700           MOVE 98 TO U-NUMERIC-STATE
007800         ELSE
007900           SET WS-STATE-NUMBER TO 1
008000           SEARCH WS-STATE
008100              AT END MOVE 1 TO U-STATE-RETURN-STATUS
008200           WHEN U-STATE = WS-STATE-ABBREVIATION (WS-STATE-NUMBER)
008300              SET U-NUMERIC-STATE TO WS-STATE-NUMBER
008400              MOVE WS-STATE-NAME (WS-STATE-NUMBER) TO U-STATE-NAME.
```

COMMUNICATION SECTION

The COMMUNICATION SECTION is that part of the DATA DIVISION wherein the interface areas necessary to communicate with data communication devices declared in the Network Controller (NC) are defined.

In a COBOL74 program the communication description entries (CD) represent the highest level of organization in the COMMUNICATION SECTION. The COMMUNICATION SECTION header is followed by a communication description entry consisting of a level indicator (CD), a data-name, and a series of independent clauses. These clauses indicate the queues and subqueues, the message date and time, the source, the text length, the status and end keys, and message count of input. Additional clauses specify the destination count, the text length, the status and error keys, and destinations for the output. The actual entry is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

B 1000 COBOL74 supports four levels of queue and subqueue names, and also provides full capabilities for the ENABLE, DISABLE, RECEIVE and SEND statements. Full Communication Section implementation is possible only if the Data Communication Subsystem is a participating MCS and supports these features. For a non-participating MCS, B COBOL74 provides limited capabilities for the ENABLE, DISABLE, RECEIVE and SEND statements as described in section 7 of this manual. A non-supportive MCS does not allow use of the SYMBOLIC SUB-QUEUES or the INITIAL clause as described in this section. Valid syntax that is not supported, however, is ignored during program execution.

COMMUNICATION SECTION

Communication Description Structure

General Format:

Format 1:

```
CD cd-name;

                                [ [ [ ; SYMBOLIC QUEUE IS data-name-1]
                                    [ ; SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                                    [ ; SYMBOLIC SUB-QUEUE-2 IS data-name-3]
                                    [ ; SYMBOLIC SUB-QUEUE-3 IS data-name-4]
FOR [ INITIAL ] INPUT               [ ; MESSAGE DATE IS data-name-5]
                                    [ ; MESSAGE TIME IS data-name-6]
                                    [ ; SYMBOLIC SOURCE IS data-name-7]
                                    [ ; TEXT LENGTH IS data-name-8]
                                    [ ; END KEY IS data-name-9]
                                    [ ; STATUS KEY IS data-name-10]
                                    [ ; MESSAGE COUNT IS data-name-11] ]
                                  [ data-name-1, data-name-2, . . . , data-name-11 ] ]
```

COMMUNICATION SECTION

Format 2:

```
CD cd-name; FOR OUTPUT
    [ ; DESTINATION COUNT IS data-name-1 ]
    [ ; TEXT LENGTH IS data-name-2 ]
    [ ; STATUS KEY IS data-name-3 ]
    [ ; DESTINATION TABLE OCCURS integer-2 TIMES
        [ ; INDEXED BY index-name-1 [ , index-name-2 ] ... ] ]
    [ ; ERROR KEY IS data-name-4 ]
    [ ; SYMBOLIC DESTINATION IS data-name-5 ] .
```

Syntax Rules:

Format 1:

1. A CD must appear only in the COMMUNICATION SECTION.

2. Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the PROCEDURE DIVISION Header.

3. Except for the INITIAL clause, the optional clauses may be written in any order.

4. If neither option in the format is specified, a level 01 data description entry must follow the CD description entry. Either option may be followed by a level 01 data description entry.

5. For each INPUT CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined as follows:

   a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1 through 12 in the record.

   b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.

   c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25 through 36 in the record.

   d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37 through 48 in the record.

COMMUNICATION SECTION

e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49 through 54 in the record.

f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign occupying character positions 55 through 62 in the record.

g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63 through 74 in the record.

h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 75 through 78 in the record.

i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.

j. The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80 through 81 in the record.

k. The MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 82 through 87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

Implicit Description			Comment
01	data-name-0.		
02	data-name-1	PICTURE X(12).	SYMBOLIC QUEUE
02	data-name-2	PICTURE X(12).	SYMBOLIC SUBQUEUE 1
02	data-name-3	PICTURE X(12).	SYMBOLIC SUBQUEUE 2
02	data-name-4	PICTURE X(12).	SYMBOLIC SUBQUEUE 3
02	data-name-5	PICTURE 9(06).	MESSAGE DATE
02	data-name-6	PICTURE 9(08).	MESSAGE TIME
02	data-name-7	PICTURE X(12).	SYMBOLIC SOURCE
02	data-name-8	PICTURE 9(04).	TEXT LENGTH
02	data-name-9	PICTURE X.	END KEY
02	data-name-10	PICTURE XX.	STATUS KEY
02	data-name-11	PICTURE 9(06).	MESSAGE COUNT

NOTE

The information under the heading Comment is for clarification and is not part of the description.

6. Record description entries following an INPUT CD implicitly redefine this record and must describe a record of exactly 87 characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. The record is always referenced according to the data descriptions defined in Syntax Rule 5.

7. Data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

Format 2:

8. A CD must appear only in the COMMUNICATION SECTION.

9. If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

10. For each OUTPUT CD, a record area of contiguous standard data format characters is allocated according to the formula: (10 + 13 x integer-2).

    a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer without an operational sign occupying character positions 1 through 4 in the record.

    b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 5 through 8 in the record.

    c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of 2 characters occupying positions 9 through 10 in the record.

    d. Character positions 11 through 23 and every set of 13 characters thereafter form table items of the following description:

        1) The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character.

        2) The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

Implicit Description	Comment
01 data-name-0.	
02 data-name-1 PICTURE 9(04).	DESTINATION COUNT
02 data-name-2 PICTURE 9(04).	TEXT LENGTH
02 data-name-3 PICTURE XX.	STATUS KEY
02 data-name OCCURS integer-2 TIMES.	DESTINATION TABLE
03 data-name-4 PICTURE X.	ERROR KEY
03 data-name-5 PICTURE X(12).	SYMBOLIC DESTINATION

COMMUNICATION SECTION

NOTE
The information under the heading Comment is for clarification and is not part of the description.

11. Record descriptions following an OUTPUT CD implicitly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. That record is always referenced according to the data descriptions defined in Syntax Rule 10.

12. Data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

13. If the DESTINATION TABLE OCCURS clause is not specified, one (1) ERROR KEY and one (1) SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

14. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may only be referred to by subscripting or indexing.

15. There is no restriction on the value of the data item referenced by data-name-1 and integer-2.

General Rules:

Format 1:

1. The INPUT CD information constitutes the communication between the Data Communication Subsystem and the COBOL74 program as information about the message being transmitted. This information does not come from the terminal as part of the message.

2. Subqueues can be implemented in B 1000 COBOL74 but if these fields are not used, ANSI$74 rules specify that these unused fields must contain spaces. Since COBOL74 initializes data to hexadecimal zeros, not spaces, it is necessary to explicitly define a value of spaces for each of these unused fields to avoid a run-time error.

3. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, subqueues, ... respectively. All symbolic names must have been previously defined in the Data Communication Subsystem. In accordance with the B naming conventions, positions 11 and 12 in each data-name must contain spaces.

   The SYMBOLIC QUEUE field in the INPUT CD is mandatory, unless the SQN option is used at run time. Refer to the B 1000 Systems Software Operation Guide, Volume 1. Since the SQN option must be sent by an MCS, use is restricted and thus, designation of the SYMBOLIC QUEUE field in the INPUT CD is the easiest method of indicating the file name.

4. A RECEIVE statement causes the serial return of the next message or portion of a message from the queue as specified by the entries in the CD.

If during the execution of a RECEIVE statement, a message from a more specific source is needed, the contents of the data item referenced by data-name-1 can be made more specific by the use of the contents of the data items referenced by data-name-2, data-name-3, and in turn data-name-4. When a given level of the queue structure is specified, all higher levels must also be specified.

If less than all the levels of the queue hierarchy are specified, the Data Communication Subsystem determines the "next" message or portion of a message to be accessed.

After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 will contain the symbolic names of all the levels of the queue structure.

5. The INITIAL clause is specified in an INPUT CD, whenever a COBOL74 program is initiated by the Data Communication Subsystem as a result of receiving a message. The INITIAL clause requires a Data Communication Subsystem that supports this concept. The symbolic names of the queue structure that demanded this activity are placed in the data items referenced by data-name-1 through data-name-4, the SYMBOLIC QUEUE. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

   The symbolic names are inserted, or the initialization to spaces is completed prior to the execution of the first PROCEDURE DIVISION statement.

   The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 will return the actual message that caused the program to be scheduled. Only at that time is the remainder of the CD updated.

6. MESSAGE DATE (data-name-5) has the format YYMMDD (year, month, day), with contents representing the date the message was received by the Data Communication Subsystem.

   The contents of the data item referenced by data-name-5 are only updated as part of the execution of a RECEIVE statement.

7. The contents of MESSAGE TIME (data-name-6) have the format HHMMSSTT (hours, minutes, seconds, hundredths of a second) and represent the time the message was received by the Data Communication Subsystem.

   The contents of the data item referenced by data-name-6 are only updated as part of the execution of the RECEIVE statement.

8. During the execution of a RECEIVE statement, the data item referenced by data-name-7 (SYMBOLIC SOURCE) is updated with the station name, as defined in the Station Section of the Network Controller. The SYMBOLIC SOURCE is the communications terminal that is the source of the message being transferred. However, if the station name of the communication terminal is not known, the contents of the data item referenced by data-name-7 will be spaces.

   After every RECEIVE operation, the user program must move the SYMBOLIC SOURCE station name to the appropriate field (SYMBOLIC DESTINATION) in the OUTPUT CD, before initiating the SEND to that station. If this is not done, a run-time error occurs since there is no destination station available for message transfer.

COMMUNICATION SECTION

9. TEXT LENGTH (data-name-8) contains the number of character positions filled as a result of the execution of the RECEIVE statement.

10. The contents of the data item referenced by data-name-9 (END KEY) are set only by the Data Communication Subsystem as part of the execution of a RECEIVE statement according to the following rules:

    a. When the RECEIVE MESSAGE phrase is specified, then:

        1) If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

        2) If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

        3) If less than a message is transferred, the contents of the data item referenced by data-name-9 are set to 0.

    b. When the RECEIVE SEGMENT phrase is specified, then:

        1) If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

        2) If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

        3) If an end of segment has been detected, the contents of the data item referenced by data-name-9 are set to 1;

        4) If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

    c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-9.

11. The STATUS KEY referenced by data-name-10 indicates the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statements.

    Use of this field is optional but highly recommended. Any RECEIVE or SEND error, left undetected because of the absence of a STATUS KEY check, causes the program to fall through to the next executable statement, in most cases, with no indication to the user that a problem exists.

    The actual association between the contents of the data item referenced by data-name-10 and the status condition is defined in table 6-4.

12. The MESSAGE COUNT referenced by data-name-11 indicates the number of messages that exist in a queue structure. The Data Communication Subsystem updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT statement with the COUNT phrase.

Format 2:

13. The nature of the OUTPUT CD information is such that it is not sent to the terminal, but contains information about the message being transmitted.

14. During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 (DESTINATION COUNT) indicate the number of symbolic destinations that are to be used from the area referenced by data-name-5 (SYMBOLIC DESTINATION).

    The first symbolic destination is found in the first occurrence of the area referenced by data-name-5; the second symbolic destination in the second occurrence of the area referenced by data-name-5 ..., up to and including the occurrence of the area referenced by data-name-5 indicated by the contents of data-name-1.

    If, during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the value of DESTINATION COUNT (data-name-1) is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

15. It is the responsibility of the user to ensure that the value of DESTINATION COUNT (data-name-1) is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

16. As part of the execution of a SEND statement, the Data Communication Subsystem interprets the contents of TEXT LENGTH (data-name-2) as user indication of the number of leftmost character positions of the data item referenced by the associated SEND identifier from which data is to be transferred. If TEXT LENGTH is zero, no characters are transferred.

    It is important that TEXT LENGTH be given the correct value prior to each SEND of a message size which varies from that of the previous SEND. If TEXT LENGTH is greater than the size of the message actually being sent, SEND fails with a STATUS KEY value of 50.

17. Each occurrence of the data item referenced by data-name-5 contains a symbolic destination (station name) previously defined in the Network Controller. If the SYMBOLIC DESTINATION (data-name-5) does not contain a valid station name, SEND fails with a STATUS KEY value of 20.

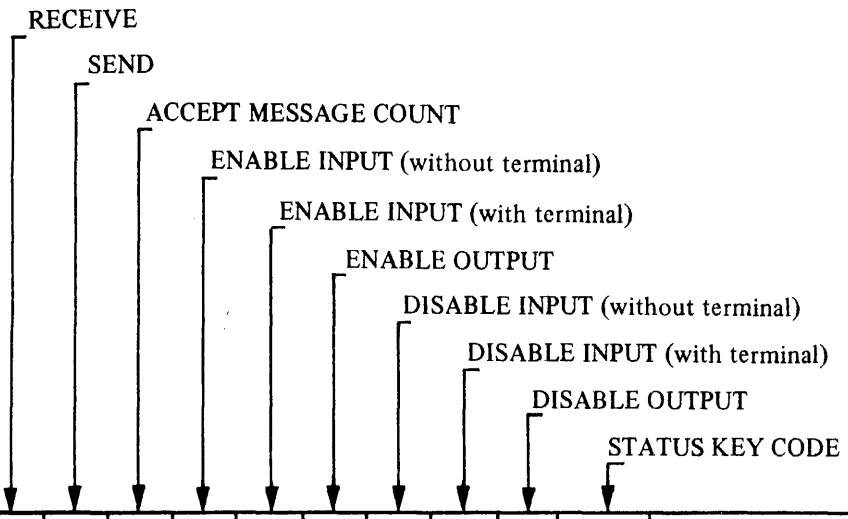
    These symbolic destination names must follow the rules for the formation of system-names. In accordance with the B naming conventions, positions 11 and 12 in data-name-5 must contain spaces.

18. The contents of STATUS KEY (data-name-3) indicates the status condition of the previously executed SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

    The actual association between the contents of the data item referenced by data-name-3 and the status condition is defined in table 6-4.

COMMUNICATION SECTION

Table 6-4. Communication Status Key Condition



*	*	*	*	*	*	*	*	*	00	No error detected. Action completed.
	*								10	One or more destinations are disabled. Action completed.
	*				*			*	20	One or more destinations unknown, or access thereto denied by system. Action completed for known destinations. No action taken for unknown destinations. Data-name-4 (ERROR KEY) indicates known or unknown (includes system denied access).
*		*	*			*			20	One or more queues unknown, or access to queue denied by system. No action taken.
				*			*		20	The source is unknown, or access thereto denied by the system. No action taken.
	*				*			*	30	Content of DESTINATION COUNT invalid. No action taken.
			*	*	*	*	*	*	40	Password invalid. No enabling/disabling action taken.
	*								50	Character count greater than length of sending field. No action taken.
	*								60	Partial segment with either zero character count or no sending area specified. No action taken.
*	*	*	*	*	*	*	*	*	91	No Data Communication Subsystem present. No action taken.

G12336

19. If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the MCP determines that any specified destination is unknown, or the Network Controller chooses to deny the program access to any destination, the contents of the data item referenced by data-name-3 (STATUS KEY) and all occurrences of the data items referenced by data-name-4 (ERROR KEY) are updated.

    The contents of the data item referenced by data-name-4, when equal to 1, indicate that the associated value in the area referenced by data-name-5 (SYMBOLIC DESTINATION) has not been previously defined or has been denied access to this destination. Otherwise, the contents of the data item referenced by data-name-4 are set to zero (0).

All Formats:

20. Table 6-4 indicates the possible contents of the data items referenced by data-name-10 for Format 1, and by data-name-3 for Format 2, at the completion of each statement shown. An asterisk (*) on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

Example:

The following program, ECHO-ECHO, illustrates the COMMUNICATION SECTION in a data communication program. The SYMBOLIC QUEUE, MCSREMOTE, is a name declared in the Network Controller. One copy of this program can be accessed by more than one station if it is executed by means of the SMCS/JOBS file. This program executes under a participating or non-participating, supporting or non-supporting Data Communication Subsystem.

COMMUNICATION SECTION

```
000100   IDENTIFICATION DIVISION.                                          : **    PCN
000200   PROGRAM-ID. ECHO-ECHO.                                            : 08 18 83bo
000300   AUTHOR.  JULIA.                                                   :
000400 *                                                                   : **    PCN
000500 * REMARKS.  THIS PROGRAM IS AN ECHO TEST PROGRAM.  ENTER A          : **    PCN
000600 * MESSAGE UP TO 20 CHARACTERS LONG AND IT WILL BE REPEATED          : 08 18 83bo
000700 * BACK TO YOU.  ENTER "END" AND IT WILL GO TO END-OF-JOB.           : **    PCN
000800 *                                                                   : **    PCN
000900   ENVIRONMENT DIVISION.                                             : **    PCN
001000   CONFIGURATION SECTION.                                            : **    PCN
001100   SOURCE-COMPUTER. B1000.                                           : **    PCN
001200   OBJECT-COMPUTER. B1000.                                           : **    PCN
001300   DATA DIVISION.                                                    : **    PCN
001400   WORKING-STORAGE SECTION.                                          : **    PCN
001500   01  OUT-AREA.                                                     : **    PCN
001600       03  FILLER          PIC X(20) VALUE "I AM YOUR ECHO !!   ".  : **    PCN
001700       03  REC-AREA.                                                : **    PCN
001800           05  END-CODE    PIC X(3).                                : **    PCN
001900           05  FILLER      PIC X(17).                               : 08 18 83bo
002000 *                                                                   : **    PCN
002100   COMMUNICATION SECTION.                                            : **    PCN
002200   CD  INPUT-Q FOR INPUT.                                            : **    PCN
002300   01  IN-Q-SPECS.                                                   : **    PCN
002400 * ********* "MCSREMOTE" IS THE NAME OF A REMOTE FILE DEFINED IN     : **    PCN
002500 * ********* THE NETWORK CONTROLLER. THE SYMBOLIC SUB-QUEUES MUST    : 08 19 83BO
002510 * ********* CONTAIN SPACES WHEN NOT USED.                           : 08 19 83BO
002600       03  SYMBOLIC-QUE            PIC X(12) VALUE "MCSREMOTE".     : **    PCN
002700       03  SY-SUB1-QUE             PIC X(12) VALUE SPACES.          : **    PCN
002800       03  SY-SUB2-QUE             PIC X(12) VALUE SPACES.          : **    PCN
002900       03  SY-SUB3-QUE             PIC X(12) VALUE SPACES.          : **    PCN
003000 * ********* NEXT 5 FIELDS ARE UPDATED WHEN A MESSAGE IS RECEIVED.   : 08 19 83BO
003100 * ********* RECEIVED.                                               : **    PCN
003200       03  MESS-DATE               PIC 9(6).                        : **    PCN
003300       03  MESS-TIME               PIC 9(8).                        : **    PCN
003400       03  IN-STATION              PIC X(12).                       : **    PCN
003500       03  IN-TEXT-LENGTH          PIC 9(4).                        : **    PCN
003600       03  END-KEY                 PIC X.                           : **    PCN
003700 * ********* IN-STATUS-KEY SHOWS STATUS WHEN ONE OF THE FOLLOWING    : **    PCN
003800 * ********* STATEMENTS IS EXECUTED:  RECEIVE, ACCEPT MESSAGE        : **    PCN
003900 * ********* COUNT, ENABLE INPUT, DISABLE INPUT.                     : **    PCN
004000       03  IN-STATUS-KEY           PIC XX.                          : **    PCN
004100 * ********* FOLLOWING FIELD IS UPDATED WHEN THE ACCEPT MESSAGE      : **    PCN
004200 * ********* COUNT STATEMENT IS EXECUTED.                            : **    PCN
004300       03  MSG-COUNT               PIC 9(6).                        : **    PCN
004400 *                                                                   : **    PCN
004500   CD  OUTPUT-Q FOR OUTPUT.                                          : **    PCN
004600   01  OUT-Q-SPECS.                                                  : **    PCN
004700 * ********* NUM-OUT-STATIONS WITH A VALUE OF 1 SENDS TO ONE         : **    PCN
004800 * ********* STATION AT A TIME.                                      : **    PCN
004900       03  NUM-OUT-STATIONS        PIC 9(4) VALUE 1.                : **    PCN
005000 * ********* OUT-TEXT-LENGTH IS UPDATED BY USER.                     : **    PCN
005100       03  OUT-TEXT-LENGTH         PIC 9(4) VALUE 40.               : 08 18 83bo
005200 * ********* FOLLOWING FIELD IS UPDATED BY SYSTEM AFTER EXECUTION    : **    PCN
```

COMMUNICATION SECTION

```
005300 * ********* OF SEND, ENABLE OUTPUT, OR DISABLE OUTPUT.                 : **    PCN
005400       03  OUT-STATUS-KEY          PIC XX.                             : **    PCN
005500 * ********* WHEN THE FOLLOWING TABLE OCCURS ONLY ONCE, THE             : 08 18 83bo
005600 * ********* MESSAGE IS SENT TO ONE  STATION AT A TIME, BASED           : 08 18 83bo
005610 * ********* ON THE VALUE IN THE OUT-STATION FIELD.                     : 08 18 83bo
005620       03  OUT-DESTINATION TABLE OCCURS 1.                             : 08 18 83bo
005700           05  OUT-ERROR-KEY           PIC X.                          : 08 18 83bo
005800 * ********* OUT-STATION IS UPDATED BY USER WITH STATION NAME (AS      : 08 19 83BO
005900 * ********* DECLARED IN NETWORK CONTROLLER) OF THE DESTINATION        : **    PCN
006000 * ********* FOR A MESSAGE.                                            : **    PCN
006100           05  OUT-STATION             PIC X(12).                      : 08 18 83bo
006200 *                                                                      : **    PCN
006300   PROCEDURE DIVISION.                                                 : **    PCN
006400 *                                                                      : 08 18 83bo
006500 * ******** THE ENABLE INPUT STATEMENT IS ONLY NECESSARY IF            : **    PCN
006600 * ******** PASSWORD CHECKING IS DESIRED FOR COBOL74 DATA              : 08 18 83bo
006610 * ******** COMMUNICATION USERS.  THE PASSWORD, REFERRED TO BELOW      : 08 19 83BO
006620 * ******** AS KEY, MUST BE ASSOCIATED WITH THE SMCS PROGRAM VIA       : 08 19 83BO
006630 * ******** THE "PASSWORD" COMMAND FROM THE SYSTEM ODT.                : 08 19 83BO
006640 * ******** SYSTEM ODT.                                               : 08 18 83bo
006650 *                                                                      : 08 18 83bo
006660 ***THE FOLLOWING PARAGRAPH DEMONSTRATES USE OF THE ENABLE VERB**      : 08 18 83bo
006670 *                                                              *      : 08 18 83bo
006680 * ENABLE-EXAMPLE.                                             *      : 08 18 83bo
006700 *     ENABLE INPUT INPUT-Q WITH KEY "BLITZ".                  *      : 08 18 83bo
006800 *     IF  IN-STATUS-KEY NOT = 00                              *      : 08 18 83bo
006900 *         DISPLAY "IN-STATUS-KEY = " IN-STATUS-KEY            *      : 08 18 83bo
007000 *             "  SINCE MY IN-STATUS-KEY DOES NOT = 00,"       *      : 08 18 83bo
007100 *             " I HAD AN ERROR. MY PASSWORD IS NOT VALID.".    *      : 08 18 83bo
007200 *         STOP RUN.                                           *      : 08 18 83bo
007300 ************************************************************          : 08 18 83bo
007310 *                                                                      : 08 18 83bo
007400   BEGINNING-OF-JOB.                                                   : **    PCN
007500       MOVE SPACES TO REC-AREA.                                        : **    PCN
007600       RECEIVE INPUT-Q MESSAGE INTO REC-AREA                           : **    PCN
007700           NO DATA GO TO BEGINNING-OF-JOB.                             : **    PCN
007800       IF  IN-STATUS-KEY NOT = 00                                      : **    PCN
007900           DISPLAY "RECEIVE WAS NOT SUCCESSFUL.  IN-STATUS-KEY = "     : **    PCN
008000               IN-STATUS-KEY                                           : **    PCN
008100           STOP RUN.                                                   : **    PCN
008200       IF  END-CODE = "END"                                            : **    PCN
008300           DISPLAY "THANK YOU FOR TALKING TO ME!!"                     : **    PCN
008400           STOP RUN.                                                   : **    PCN
008500 * ********* SEND MESSAGE TO STATION THAT YOU RECEIVED FROM.           : **    PCN
008600       MOVE IN-STATION TO OUT-STATION(1).                              : 08 18 83bo
008700       SEND OUTPUT-Q FROM OUT-AREA WITH EMI.                           : **    PCN
008800       IF  OUT-STATUS-KEY NOT = 00                                     : **    PCN
008900           DISPLAY "MY SEND WAS NOT SUCCESSFUL."                       : **    PCN
009000               "  MY OUT-STATUS-KEY = " OUT-STATUS-KEY                 : **    PCN
009100               "; MY OUT-ERROR-KEY = " OUT-ERROR-KEY(1)                : 08 18 83bo
009200           STOP RUN.                                                   : **    PCN
009300       IF  OUT-ERROR-KEY (1) NOT = 0                                   : 08 18 83bo
009400           DISPLAY "OUT-STATION HAS BEEN DENIED, OOPS!"                : **    PCN
009500           STOP RUN.                                                   : **    PCN
009600       GO TO BEGINNING-OF-JOB.                                         : **    PCN
```

SECTION 7
PROCEDURE DIVISION

GENERAL

The PROCEDURE DIVISION must be included in every COBOL74 source program. This division may contain declarative and nondeclarative procedures.

Declarative sections must be grouped at the beginning of the PROCEDURE DIVISION preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. Refer to the USE statement in this section.

RULES OF PROCEDURE FORMATION

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

The end of the PROCEDURE DIVISION and the physical end of the program is that physical position in a COBOL74 source program after which no further procedures appear.

A section consists of a section header followed by zero, one, or more successive paragraphs. A section ends immediately before the next section or at the end of the PROCEDURE DIVISION or, in the declaratives portion of the PROCEDURE DIVISION, at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name, followed by a period and a space, followed by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the PROCEDURE DIVISION or, in the declaratives portion of the PROCEDURE DIVISION, at the key words END DECLARATIVES.

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

A statement is a syntactically valid combination of words and symbols beginning with a COBOL74 verb.

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

EXECUTION OF THE PROCEDURE DIVISION

Execution begins with the first statement of the PROCEDURE DIVISION, excluding declaratives. Statements are executed in the order of appearance, except where the user indicates GO TO, PERFORM, CALL, conditional statements, and declarative procedures.

PROCEDURE DIVISION STRUCTURE

The PROCEDURE DIVISION is made up of the PROCEDURE DIVISION header and the PROCEDURE DIVISION body. Descriptions of these follow.

PROCEDURE DIVISION Header

The PROCEDURE DIVISION is identified by and must begin with the following header:

```
PROCEDURE DIVISION [USING data-name-1 [ , data-name-2 ] ... ] .
```

PROCEDURE DIVISION Body

The body of the PROCEDURE DIVISION must conform to one of the following two formats.

Format 1:

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

Format 2:

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

STATEMENTS AND SENTENCES

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

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

Conditional Statements

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

A conditional statement is

1. An IF, SEARCH, or RETURN statement.
2. A READ statement that specifies the AT END or INVALID KEY phrase.
3. A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.
4. A START, REWRITE, or DELETE statement that specifies the INVALID KEY phrase.
5. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
6. A RECEIVE statement that specifies a NO DATA phrase.
7. A STRING, UNSTRING, or CALL statement that specifies the ON OVERFLOW phrase.

Example:

The following syntax for the IF statement is an example of a conditional statement.

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

Statement-1 or statement-2 can be either imperative or conditional statements. If statement-1 or statement-2 is conditional, then these conditions within the conditional statement (IF statement) are considered to be nested.

STATEMENTS AND SENTENCES

Conditional Sentences

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

Examples:

```
IF AGE IS GREATER THAN VOTE-AGE GO TO PARTY-TYPE,
ELSE ADD 1 TO DONT-COUNT, GO TO GET-NEXT-PERSON.

IF SALES IS EQUAL TO QUOTA , MOVE SALESMAN TO
PROMOTION-LIST, ELSE MOVE SALESMAN TO BAD-LIST.
```

Compiler-Directing Statements

A compiler-directing statement consists of a compiler-directing verb and its operands. The compiler-directing verbs are COPY and USE (refer to the COPY statement and the USE statement in this section). A compiler-directing statement causes the compiler to take a specific action during compilation.

Compiler-Directing Sentences

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

Example:

```
MYFILE. COPY "FAMILY/MYFILE ON MYPACK".
```

Imperative Statements

An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement nor a compiler-directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT	GENERATE	SEND
ADD (1)	GO	SET
ALTER	INITIATE	SORT
CALL(3)	INSPECT	START (2)
CANCEL	MERGE	STOP
CLOSE	MOVE	STRING (3)
COMPUTE (1)	MULTIPLY (1)	SUBTRACT (1)
DELETE (2)	OPEN	SUPPRESS
DISABLE	PERFORM	TERMINATE
DISPLAY	READ (5)	UNSTRING (3)
DIVIDE (1)	RECEIVE (4)	WAIT
ENABLE	RELEASE	WRITE (6)
EXIT	REWRITE (2)	

The numbers in parentheses following some of the verbs have the following meaning:

Number	Meaning
1	Without the optional SIZE ERROR phrase.
2	Without the optional INVALID KEY phrase.
3	Without the optional ON OVERFLOW phrase.
4	Without the optional NO DATA phrase.
5	Without the optional AT END phrase or INVALID KEY phrase.
6	Without the optional INVALID KEY phrase or END-OF-PAGE phrase.

When 'imperative-statement' appears in the general format of statements, it refers to a statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements. Imperative statements must be ended by a period, or an ELSE phrase associated with a previous IF statement, or a WHEN phrase associated with a previous SEARCH statement.

Imperative Sentences

An imperative sentence is one or more imperative statements terminated by a period. An imperative statement can contain either a GO TO statement or a STOP RUN statement which, if present, must be the last statement in the sentence.

Examples:

```
ADD 1 TO ITEM-COUNT, GO TO READ-NEXT-ITEM.
DISPLAY "THIS IS THE END ", STOP RUN.
```

CONTROL RELATIONSHIP BETWEEN PROCEDURES

In COBOL74, imperative and conditional sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the rules of the coding form (section 3), to establish the sequence in which the object program is to execute the procedure. In the PROCEDURE DIVISION, names are used so that one procedure can reference another by naming the procedure to be referenced. In this way, the sequence in which the object program is to be executed may be varied simply by transferring control to a named procedure.

In procedure execution, control is transferred only to the beginning of a paragraph or section. Control is passed to a sentence within a paragraph only from the sentence written immediately preceding it. If a procedure is named, control can be passed to it from any sentence which contains a GO TO or PERFORM, followed by the name of the procedure to which control is to be transferred.

Paragraphs

So that the source programmer may group several sentences to convey one idea (procedure), paragraphs have been included in COBOL74. In writing procedures in accordance with the rules of the PROCEDURE DIVISION and the requirements of the coding form (section 3), the programmer begins a paragraph with a name. The name consists of a word followed by a period, and the name precedes the paragraph it names. A paragraph is terminated by the next paragraph-name. The smallest grouping of the PROCEDURE DIVISION which is named is a paragraph.

Programs may contain identical paragraph-names, provided they are resident in different sections. If such paragraph-names are not qualified when used, the current section is assumed. Paragraph-names may be used in GO TO, PERFORM, and ALTER statements.

Sections

A section consists of zero, one, or more successive paragraphs and must be named when designated. The section-name is followed by the word SECTION, a priority number which is optional, and a period. If the section is a DECLARATIVE section, the DECLARATIVE sentence USE or COPY follows the section header and begins on the same line. Under all other circumstances, a sentence may not begin on the same line as a section-name. The section-name applies to all successive paragraphs until another section-name is found.

Since paragraph-names and section-names both have the same designated position on the reference format (position A), section-names, when specified, are written on one line followed by a paragraph name on a subsequent line.

SEGMENTATION

COBOL74 segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

COBOL74 segmentation deals only with segmentation of procedures. As such, only the PROCEDURE DIVISION and the ENVIRONMENT DIVISION are considered in determining segmentation requirements for an object program.

Segmentation provides the facility of intermixing sections with different segment-numbers and allows the fixed portion of the source program to contain segments that may be overlaid.

Program Segments

Although it is not mandatory, the PROCEDURE DIVISION for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire PROCEDURE DIVISION must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation does not affect the need for qualification of procedure-names to ensure uniqueness.

Fixed Portion

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

A fixed permanent segment is the main program segment and may be overlaid in the same manner as if it were a fixed overlayable segment. A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause (refer to SEGMENT-LIMIT in this section). Such a segment, if called for by the program, is always made available in the last used state.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, either a fixed overlayable segment or another independent segment. An independent segment is in initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

3. Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in step 2 below).

On subsequent transfer of control to the segment, an independent segment is in the state when last used.

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraphs 1 and 2 above).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

Refer to Explicit and Implicit Transfers of Control in section 2 for additional information.

Segmentation Classification

Sections which are to be segmented are classified, using a system of segment-numbers and the following criteria.

1. Logic requirements: Sections which must be available for reference at all times, or which are referred to frequently, are normally classified as belonging to one of the permanent segments. Sections which are used less frequently are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.

2. Frequency of use: The more frequently a section is referred to, the lower the segment-number; the less frequently it is referred to, the higher the segment-number.

3. Relationship to other sections: Sections which frequently communicate with one another should be given the same segment-numbers.

Segmentation Control

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. If any reordering of the object program is required to handle the flow from segment to segment, according to the rules, the compiler provides control transfers to maintain the logical flow specified in the source program. The compiler also provides all controls necessary for a segment to operate whenever the segment is used. Control may be transferred within a source program to any paragraph in a section. It is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

Segment-Numbers

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

General Format:

Section-name SECTION [segment-number] .

Syntax Rules:

1. The segment-number must be an integer ranging in value from 0 through 126.
2. If the segment-number is omitted from the section header, the segment-number is assumed to be zero.
3. Sections in the declaratives must contain segment-numbers less than 50.

General Rules:

1. All sections which have the same segment-number constitute a program segment. Sections with the same segment-numbers need not be physically contiguous in the source program.
2. Segments with segment-numbers 0 through 49 belong to the fixed portion of the object program.
3. Segments with segment-numbers 50 through 126 are independent segments.

SEGMENTATION

SEGMENT-LIMIT

Ideally, all program segments having segment-numbers ranging from 0 through 49 should be specified as permanent segments. However, when insufficient memory is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means of reducing the number of permanent segments in a program, while still retaining the logical properties of fixed portion segments (segment-numbers 0 through 49).

General Format:

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION and has the following format:

```
[ , SEGMENT-LIMIT IS segment-number ]
```

Syntax Rules:

1. Segment-number must be an integer ranging in value from 1 through 49.

General Rules:

1. When the SEGMENT-LIMIT clause is specified, only those segments having segment-numbers from 0 up to, but not including, the segment-number designated as the segment-limit, are considered as permanent segments of the object program.

2. Those segments having segment-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

3. When the SEGMENT-LIMIT clause is omitted, all segments having segment-numbers from 0 through 49 are considered as permanent segments of the object program.

Example:

All segments whose priority number is less than that specified in SEGMENT-LIMIT are gathered into a single segment, regardless of physical location in the source program. All other segments equal to or greater than that specified in SEGMENT-LIMIT are gathered into overlayable segments according to equal priority number, regardless of physical location in the source program.

The use of the gathering technique allows programmers to create tailored segments which reduce disk access times.

Example:

Program A: SEGMENT-LIMIT equals 17.

Non-Gathered

Segment	Description	Size in Digits
00-16	Main body of the program	4,000
17	Used frequently	1,000
18	Used frequently	5,000
19	Used infrequently	4,000
20	Used at EOJ only	500
21	Used frequently	2,000
22	Used at BOJ only	1,000
23	Used frequently	500
24	Used for infrequent test	1,500
25	Used infrequently	3,000

Gathered

Segment	Description	Size in Digits
00-16	Main body of the program	4,000
17	Used frequently	1,000
18	Used frequently	5,000
19	Used infrequently	4,000
20	Used at EOJ only	500
17	Used frequently (was segment 21)	2,000
19	Used at BOJ only (was segment 22)	1,000
17	Used frequently (was segment 23)	500
20	Used for infrequent test (was segment 24)	1,500
20	Used infrequently (was segment 25)	3,000

Results of Gathering

Segment	Description	Size in Digits
00-16	Main body of the program	4,000
17	Used frequently	3,500
18	Used infrequently	5,000
19	Used infrequently	5,000
20	Used infrequently	5,000

"Fall through" is performed in the sequence shown in the Non-Gathered example, and not as appears in the Results of Gathering example. This preserves the logical integrity of the original program.

The COBOL74 interpreter automatically checks to see whether an overlay being called for by an object program is already present in memory. If present, no disk access is required and the program is not interrupted. If the overlay is not present, the COBOL74 interpreter interrupts the program and accesses the disk for the desired overlayable portion of the program. The COBOL74 interpreter uses overlay segments directly from the program library where the object program was compiled to, and is called in as an overlay in the initial generated code every time it is required by the operating program.

Restrictions on Program Flow

When segmentation is used, the following restriction is placed on the ALTER statement.

The ALTER Statement

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER statement refers is in a fixed overlayable segment.

THE PROCEDURE DIVISION HEADER

The PROCEDURE DIVISION is identified by, and must begin with, the following header:

```
PROCEDURE DIVISION [USING data-name-1 [ , data-name-2 ] ... ] .
```

The USING phrase is present 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 in the DATA DIVISION of the program in which this header occurs, and must have a 01 or 77 level-number.

Within a called program, LINKAGE SECTION data items are processed according to the data descriptions given in the called program.

When the USING phrase is present, the object program operates as if data-name-1 of the PROCEDURE DIVISION header in the called program and data-name-1 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. The descriptions of data must define an equal number of character positions; however, they need not be the same name. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the PROCEDURE DIVISION header of the called program; however, a given data-name may appear more than once in the same USING phrase of a CALL statement.

If the USING phrase is specified, the INITIAL clause must not be present in any CD entry. Refer to Syntax Rule 2, Format 1, of the Communication Description entry (CD), section 6.

DECLARATIVES

Declaratives are procedures which operate under the control of the input-output system or the DEBUG facility. Declaratives consist of compiler-directing sentences and associated procedures. Declaratives, if used, must be grouped together at the beginning of the PROCEDURE DIVISION. The group of declaratives must be preceded by the key word DECLARATIVES, and must be followed by the words END DECLARATIVES. Each DECLARATIVE consists of a single section and must conform to the rules for procedure formation. The next source statement following the END DECLARATIVES statement must be a section-name or paragraph-name.

USE Declarative

A USE declarative is used to supplement the standard procedures provided by the input-output system. The USE sentence immediately following the section-name identifies the condition calling for the execution of the USE procedures. Only the PERFORM statements may reference all or part of a USE section. The USE sentence alone is never executed. Within a USE procedure, there must be no reference to the main body of the PROCEDURE DIVISION. The construct for the USE declarative is as follows:

```
section-name SECTION.  USE . . . . . .

paragraph-name.  First procedure-statement . . . .
```

Complete rules for writing the formats for USE are stated under the USE statement in this section.

USE FOR DEBUGGING Declarative

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section. The construct of the USE FOR DEBUGGING statement is:

```
section-name SECTION.  USE FOR DEBUGGING . . .
```

Complete rules for writing the format for USE FOR DEBUGGING are stated in section 10 (DEBUG) of this manual.

ARITHMETIC EXPRESSIONS

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operators, and parentheses are given in table 7-1.

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

Arithmetic Operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a separator and followed by a separator.

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

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by the numeric literal +1
–	The effect of multiplication by numeric literal -1.

Formation and Evaluation Rules

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

   1st – Unary plus (+) and minus (-)
   2nd – Exponentiation
   3rd – Multiplication and division
   4th – Addition and subtraction

ARITHMETIC EXPRESSIONS

2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in expressions where it is necessary for deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right. The following expressions are ordinarily considered to be ambiguous.

   A / B * C A / B / C A ** B ** C

   These expressions are permitted in COBOL74 and are interpreted as if written, respectively:

   (A / B) * C (A / B) /C (A ** B) ** C

   Without parenthesizing, the following example

   A + B / C + D ** E * F - G

   is interpreted as

   A + (B / C) + ((D ** E) * F) - G .

   The sequence of operations working from the innermost parentheses to the outermost. That is, first exponentiation, then multiplication and division, and finally addition and subtraction are performed.

3. The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in table 7-1, which illustrates that

   a. The letter 'P' indicates a permissible pair of symbols.

   b. The hyphen character '-' indicates an invalid pair.

   c. The word Variable indicates an identifier or literal.

Table 7-1. Combination of Symbols in Arithmetic Expressions

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

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

5. The following rules apply to evaluation of exponentiation in an arithmetic expression:

   a. If the value of an expression to be raised to a power is zero, the exponent must have a value not greater than zero. Otherwise, the size error condition exists. Refer to the SIZE ERROR phrase in this section.

   b. If the evaluation yields both a positive and a negative real number, the value returned as the result is the positive number.

   c. If no real number exists as the result of the evaluation, the size error condition exists.

6. When no resultant-identifier is associated with an expression, an intermediate data item is used to store the value of the arithmetic expression.

Intermediate Data Item

An intermediate data item is a signed numeric data item containing the values developed in the course of evaluating an arithmetic expression prior to the final value being moved to the resultant-identifier, if any. The length of this data item is determined by the compiler throughout the calculation.

CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, PERFORM, and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case the category is not changed.

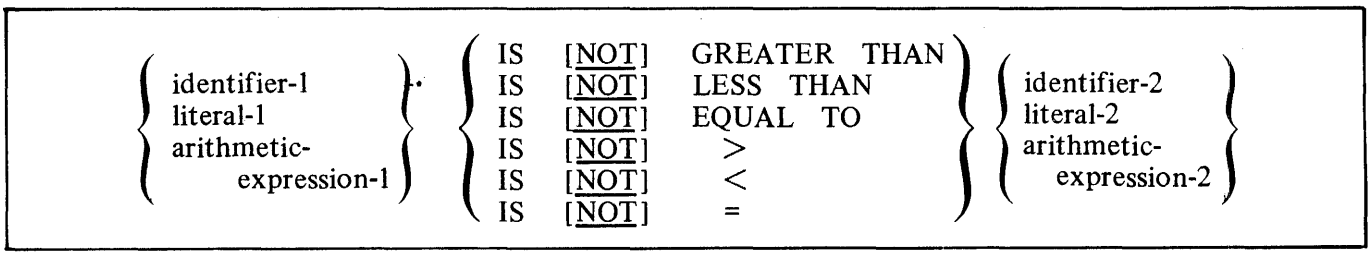
Simple Conditions

The simple conditions are the relation, class, condition-name, switch status, and sign conditions. A simple condition has a truth value of TRUE or FALSE. The inclusion in parentheses of simple conditions does not change the simple truth value. Simple conditions cannot contain more than one relational operator.

Relation Condition

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic expression. A relation condition has a truth value of TRUE if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

General Format:



NOTE
The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '>' (greater than or equal to).

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is the subject of the condition; the second operand (identifier-2, literal-2, or arithmetic-expression-2) is the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space, comma, semicolon, right parenthesis, closing quotation mark, or closing commercial at sign (@) must precede the first reserved word comprising the relational operator. A space, comma, semicolon, left parenthesis, opening quotation mark, or opening commercial at sign (@) must follow the last reserved word comprising the relational operator. If the relational operator consists of more than one reserved word, then a space, comma or semicolon must be used to separate each pair of consecutive reserved

words comprising the relational operator. When used, NOT and the next key word or relational character are one relational operator that defines the comparison to be executed for truth value; for example, NOT EQUAL is a truth test for an unequal comparison; NOT GREATER is a truth test for an equal or less comparison. The meaning of the relational operators is as follows:

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

NOTE

The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '>' (greater than or equal to).

Comparison of Numeric Operands

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

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

Comparison of Nonnumeric Operands

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. Refer to OBJECT-COMPUTER in Section 5 for additional information. If one of the operands is specified as numeric, it must be an integer data item or an integer literal. The following conditions apply:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this alphanumeric data item are compared to the nonnumeric operand. Refer to the MOVE statement and the PICTURE character 'P' in this section.

2. If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item are compared to the nonnumeric operand. Refer to the MOVE statement, and the PICTURE character 'P' for additional information.

3. A noninteger numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands may be compared only when usage is the same.

CONDITIONAL EXPRESSIONS

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

1. Operands of equal size. If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

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

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

Comparisons Involving Index-Names and/or Index Data Items

Relation tests may be made between:

1. Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

2. An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

3. An index data item and an index-name or another index data item. The actual values are compared without conversion.

   The comparison of an index data item with a literal or with any data item not specified above, is not allowed.

Class Condition

The class condition determines whether the operand is numeric, consisting entirely of the characters '0', '1', '2', '3', ..., '9', with or without the operational sign, or alphabetic, consisting entirely of the characters 'A', 'B', 'C', ..., 'Z', and space.

General Format:

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

The usage of the operand used with the ALPHABETIC test must be DISPLAY. The usage of the operand used with the NUMERIC test must be DISPLAY or COMPUTATIONAL. When used, NOT and the next key word specify one class condition that defines the class test to be executed for truth value; for example, NOT NUMERIC is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. The position and representation of valid operational signs is discussed in the PICTURE clause, General Rule 8, the S symbol, and the SIGN clause in section 6.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space character.

Condition-Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not the value is equal to one of the values associated with a condition-name.

General Format:

condition-name

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not the value is within this range, including the end values.

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

The result of the test is TRUE if one of the values corresponding to the condition-name equals the value of the associated conditional variable.

Switch-Status Condition

A switch-status condition determines the ON or OFF status of a switch. The switch-name and the ON or OFF value associated with the condition must be named in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

General Format:

condition-name

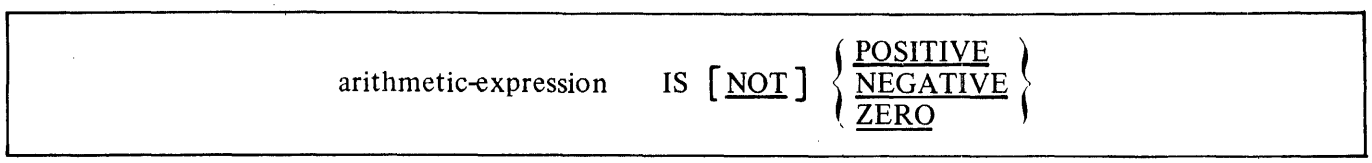
The result of the test is TRUE if the switch is set to the specified position corresponding to the condition-name.

CONDITIONAL EXPRESSIONS

Sign Condition

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

General Format:



When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; for example, NOT ZERO is a truth test for a nonzero (positive or negative) value. An operand is positive if the value is greater than zero, negative if the value is less than zero, and zero if the value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, and/or complex conditions with logical connectors (logical operators AND and OR) or negating these conditions with logical negation (the logical operator NOT). The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated.

The logical operators with meanings follow.

Logical Operator	Meaning
AND	Logical conjunction; the truth value is TRUE if both of the conjoined conditions are TRUE; FALSE if one or both of the conjoined conditions is FALSE.
OR	Logical inclusive OR; the truth value is TRUE if one or both of the included conditions is TRUE; FALSE if both included conditions are FALSE.
NOT	Logical negation or reversal of truth value; the truth value is TRUE if the condition is FALSE; FALSE if the condition is TRUE.

The logical operators must be preceded by a space and followed by a space.

Examples:

The following are illustrations of complex conditions:

AGE IS LESS THAN MAX-AGE AND AGE IS GREATER THAN 20.

AGE IS GREATER THAN 24 OR MARRIED.

STOCK-ON-HAND IS LESS THAN DEMAND OR STOCK-SUPPLY IS GREATER THAN DEMAND + INVENTORY.

A IS EQUAL TO B, AND C IS NOT EQUAL TO D, OR E IS NOT EQUAL TO F, AND G IS POSITIVE, OR H IS LESS THAN I * J.

STOCK-ACCT IS GREATER THAN 72 AND (STK-NUMBER IS LESS THAN 100 OR STK-NUMBER EQUAL TO 62879).

It is not necessary to use the same logical connective throughout the complex expression.

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT. The negated simple condition effects the opposite truth value for a simple condition. Thus, the truth value of a negated simple condition is TRUE only if the truth value of the simple condition is FALSE; the truth value of a negated simple condition is FALSE only if the truth value of the simple condition is TRUE. The inclusion in parentheses of a negated simple condition does not change the truth value.

General Format:

```
NOT simple-condition
```

Combined and Negated Combined Conditions

A combined condition results from connecting conditions with one of the logical operators AND or OR.

General Format:

```
condition { { AND } condition } ...
          { { OR  }           }
```

CONDITIONAL EXPRESSIONS

Where condition may be:

1. A simple condition.
2. A negated simple condition.
3. A combined condition.
4. A negated combined condition. The NOT logical operator followed by a combined condition enclosed within parentheses.
5. Combinations of the above, specified according to the rules summarized in table 7-2.

Although parentheses need never be used when either AND or OR (but not both) is used exclusively in a combined condition, parentheses may be used to effect a final truth value when a mixture of AND, OR, and NOT is used. Refer to table 7-2 and Condition Evaluation Rules in this section for additional information.

Table 7-2 indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of the corresponding right parenthesis.

Table 7-2. Combinations of Conditions, Logical Operators, and Parentheses

Given the following element	Location in Conditional Expression		In a left-to-right sequence of elements:	
	First	Last	Element, when not first, may be immediately preceded by only:	Element, when not last, may be immediately followed by only:
Simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition, (
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

The element pair OR NOT is permissible but the pair NOT OR is not permissible. NOT (is permissible but NOT NOT is not permissible.

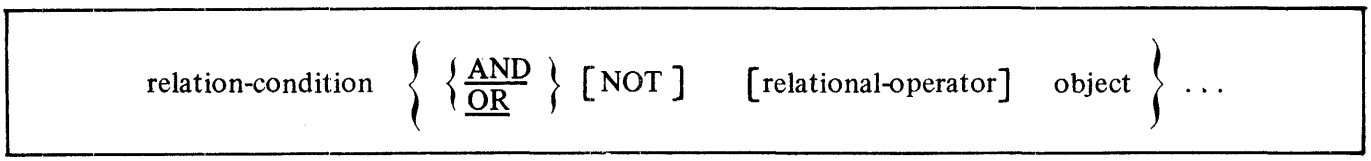
Abbreviated Combined Relation Conditions

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition.

2. The omission of the subject and relational operator of the relation condition.

General Format:



Within a sequence of relation conditions, both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules of table 7-2. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

In abbreviated relation conditions, an arithmetic expression beginning with a left parenthesis may not be the object of an abbreviation where both the subject and the relational operator are implied. An arithmetic expression may be the object if it does not begin with a left parenthesis or if the relational operator is stated.

The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:

1. If the word immediately following NOT is GREATER, '>', LESS, '<', EQUAL, or '=', then the NOT participates as part of the relational operator.

2. The NOT is interpreted as a logical operator and the implied insertion of subject or relational operator results in a negated relation condition.

Examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

CONDITIONAL EXPRESSIONS

Examples:

Abbreviated Combined Relation Condition	Expanded Equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d))))
IF A = B OR C	IF A = B OR A = C
IF A < B OR = C OR D	IF A < B OR A = C OR A = D

Condition Evaluation Rules

Parentheses may be used to specify the order in which individual conditions of complex conditions are to be evaluated when necessary to depart from the implied evaluation precedence. Conditions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used, or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evaluation is implied until the final truth value is determined.

1. Values are established for arithmetic expressions. Refer to Formation and Evaluation Rules in this section.

2. Truth values for simple conditions are established in the following order:

   relation (following the expansion of any abbreviated relation condition)
   class
   condition-name
   switch-status
   sign

3. Truth values for negated simple conditions are established.

4. Truth values for combined conditions are established: .IN +5 P AND logical operators, followed by

OR logical operators.

5. Truth values for negated combined conditions are established.

6. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation of consecutive operations of the same hierarchical level is from left to right.

Example:

To evaluate
C1 AND (C2 OR NOT (C3 OR C4)

reduce as follows:
let C5 equal "C3 OR C4", resulting in C1 AND (C2 OR NOT C5)

let C6 equal "C2 OR NOT C5", resulting in C1 AND C6.

COMMON PHRASES

In the statement descriptions that follow, several phrases appear frequently: the ROUNDED phrase, the SIZE ERROR phrase, and the CORRESPONDING phrase.

In the following discussion, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

ROUNDED Phrase

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by adding a one into the low-order digit whenever the absolute value of the next least significant digit of the intermediate data item is greater than or equal to five.

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

SIZE ERROR Phrase

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

1. If the SIZE ERROR phrase is not specified and a size error condition occurs, the resultant value is stored in each of the receiving fields left truncated where required. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

   If division by zero is the cause of the size error condition, the execution of the program is abnormally terminated.

2. If the SIZE ERROR phrase is specified and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

   For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative statement in the SIZE ERROR phrase is not executed until all of the individual additions and subtractions are completed.

CORRESPONDING Phrase

For the purpose of this discussion, d1 and d2 must each be identifiers that refer to group items. A pair of data items, one from d1 and one from d2 correspond if the following conditions exist:

1. A data item in d1 and a data item in d2 are not designated by the key word FILLER and have the same data-name and the same qualifiers up to, but not including, d1 and d2.

2. At least one of the data items is an elementary data item in the case of a MOVE statement with the CORRESPONDING phrase. Both of the data items are elementary numeric data items in the case of the ADD statement with the CORRESPONDING phrase or the SUBTRACT statement with the CORRESPONDING phrase.

3. The description of d1 and d2 must not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.

4. A data item that is subordinate to d1 or d2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, d1 and d2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses. Refer to the OCCURS clause in section 6.

Refer to the ADD statement in this section for an example of CORRESPONDING.

GENERAL RULES FOR STATEMENT FORMATS

The following paragraphs describe general rules for statement formats.

Arithmetic Statements

The arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT and have several common features:

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is 18 decimal digits.

3. Each arithmetic operation is evaluated using an intermediate data item for the result of the operation. The contents of the intermediate data item are moved to the resultant-identifier according to the rules for the MOVE statement. Rounding is performed and the size error condition is determined only during this MOVE operation. Refer to Intermediate Data Item, Rounded Phrase, Size Error Phrase, and the MOVE Statement in this section for additional information.

Overlapping Operands

When a sending and a receiving item in an arithmetic statement or an INSPECT, MOVE, SET, STRING, or UNSTRING statement share a part, but not all, assigned storage areas, the result of the execution of such a statement is undefined.

Multiple Results in Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following ways:

1. A statement which performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence in which the multiple results are listed.

   The result of the statement

   ```
   ADD A, B, C, TO C, D(C), E
   ```

   is equivalent to:

   ```
   ADD A, B, C GIVING TEMP
   ADD TEMP TO C
   ADD TEMP TO D(C)
   ADD TEMP TO E
   ```

   where TEMP is an intermediate result item provided by the compiler.

Incompatible Data

Except for the class condition (refer to Class Condition in this section), when the contents of a data item are referenced in the PROCEDURE DIVISION and the contents of that data item are not compatible with the class specified for that data item by the PICTURE clause, then the result of such a reference is undefined.

Numeric Functions

A numeric function can be specified as a sending operand in an arithmetic statement, a MOVE statement, or an arithmetic expression.

OFFSET Function

The OFFSET function represents the number of characters preceding a data item in a logical record.

General Format:

OFFSET (data-name)

Syntax Rules:

1. Data-name must be described in the DATA DIVISION.
2. Data-name can be qualified.

General Rules:

1. OFFSET (data-name) represents the number of characters preceding the data item referenced by data-name in the logical record in which data-name is defined. If data-name references a packed numeric data item not aligned on a character boundary, OFFSET (data-name) represents the number of characters preceding the character in which data-name begins. If data-name is a record-name or a 77-level item, the value of OFFSET (data-name) is zero.
2. The internal representation of OFFSET is the same as that of an index data item; that is, it occupies the same space and has the same alignment as an item declared PICTURE S9(7) USAGE IS COMPUTATIONAL.

STATEMENT FORMATS

Categories of Verbs

The verbs available for use with the COBOL74 compiler are categorized below. Although the word IF is not a verb in the English language, it is utilized as such in the COBOL74 language. Its occurrence is a vital feature in the PROCEDURE DIVISION.

Category	Verb	Option
Arithmetic	ADD	
	COMPUTE	
	DIVIDE	
	INSPECT	TALLYING
	MULTIPLY	
	SUBTRACT	
Compiler Directing	COPY	
	USE	
Conditional	ADD	SIZE ERROR
	CALL	OVERFLOW
	COMPUTE	SIZE ERROR
	DELETE	INVALID KEY
	DIVIDE	SIZE ERROR
	IF	
	MULTIPLY	SIZE ERROR
	READ	END or INVALID KEY
	RECEIVE	NO DATA
	RETURN	END
	REWRITE	INVALID KEY
	SEARCH	
	START	INVALID KEY
	STRING	OVERFLOW
	SUBTRACT	SIZE ERROR
	UNSTRING	OVERFLOW
	WRITE	INVALID KEY or END-OF-PAGE
Data Movement	ACCEPT	DATE, DAY, or TIME
	INSPECT	REPLACING
	MOVE	
	STRING	
	UNSTRING	
Ending	STOP	

Category	Verb	Option
Input-Output	ACCEPT	identifier
	CLOSE	
	DELETE	
	DISABLE	
	DISPLAY	
	ENABLE	
	OPEN	
	READ	
	RECEIVE	
	REWRITE	
	SEND	
	START	
	STOP	literal
	WRITE	
Inter-Program Communication	CALL	
	CANCEL	
Ordering	MERGE	
	RELEASE	
	RETURN	
	SORT	
Procedure Branching	ALTER	
	CALL	
	EXIT	
	GO TO	
	PERFORM	
Table Handling	SEARCH	
	SET	

SPECIFIC VERB FORMATS

The specific verb formats, together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages in alphabetic sequence.

ACCEPT

The ACCEPT statement causes low volume data to be made available to the specified data item.

General Format:

Format 1:

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

Format 2:

```
ACCEPT identifier FROM { DATE
                       { DAY
                       { TIME
                       { TIMER
                       { TODAYS-DATE
                       { TODAYS-NAME
```

Syntax Rules:

1. The mnemonic-name in Format 1 must also be specified in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION and must be associated with the hardware-name ODT.

2. An item declared in the LINKAGE SECTION cannot be used with the ACCEPT statement.

General Rules:

Format 1:

1. The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the contents of the data item named by the identifier.

2. The maximum number of characters that can be transferred is unlimited.

3. If the FROM phrase is not given, the device that is used is the ODT.

4. When the operator enters the AX message in response to the ACCEPT, continuation of the object program begins with the next executable statement in sequence.

Format 2:

5. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement.

DATE, DAY, TODAYS-DATE, TODAYS-NAME, TIMER, and TIME, are conceptual data items and are not described in the COBOL74 program. Also, they cannot be the operand in a MOVE or DISPLAY statement.

6. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes are from high order to low order (left to right), year of century, month of year, and day of month (YYMMDD). July 1, 1968 would be expressed as 680701. DATE, when accessed by a COBOL74 program, behaves as that described in the COBOL74 program as an unsigned elementary numeric integer data item, six digits in length.

7. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes are from high order to low order (left to right) year of century, day of year (YYDDD). July 1, 1968 is expressed as 68183. DAY, when accessed by a COBOL74 program, behaves as if described in a COBOL74 program as an unsigned elementary numeric integer data item, five digits in length.

8. TIME is composed of the data elements hours, minutes, seconds, and hundredths of a second, (HHMMSSTH). TIME is based on elapsed time after midnight on a 24-hour clock basis; 2:41 p.m. is expressed as 14410000. TIME, when accessed by a COBOL74 program, behaves as if described in a COBOL74 program as an unsigned elementary numeric integer data item, eight digits in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595990.

9. TODAYS-DATE is composed of the data elements month of year, day of month, and year of century (MMDDYY). July 1, 1968 is expressed as 070168. TODAYS-DATE, when accessed by a COBOL74 program, behaves as if described in COBOL74 as an unsigned elementary numeric integer data item, six digits in length.

10. TODAYS-NAME is composed of the name of the current day of the week. TODAYS-NAME, when accessed by a COBOL74 program, behaves as if described in COBOL74 as an elementary alphanumeric data item nine characters in length with the name of the current day left-justified with space fill.

11. TIMER is composed of the current value of the object computer interval timer in tenths of seconds. TIMER, when accessed by a COBOL74 program, behaves as if described in COBOL74 as an unsigned elementary numeric integer data item, ten digits in length.

ACCEPT MESSAGE COUNT

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

General Format:

ACCEPT cd-name MESSAGE COUNT

Syntax Rules:

1. Cd-name must reference an input CD.

General Rules:

1. The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue.

2. Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by the communication description entry must contain the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-11 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated. Refer to the Communication Description (CD) in section 6.

ADD

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

General Format:

Format 1:

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

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

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

Format 2:

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

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

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

Format 3:

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

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

VERB FORMAT: ADD

Syntax Rules:

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item. In Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. Additional rules and explanations relative to this statement are given in the appropriate paragraphs. Refer to Intermediate Data Item, CORRESPONDING Phrase, ROUNDED Phrase, SIZE ERROR Phrase, Arithmetic Statements, Overlapping Operands, and Multiple Results in Arithmetic Statements in this section.

2. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-n storing the result immediately into identifier-n. This process is repeated, respectively, for each operand following the word TO.

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

4. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

5. The compiler ensures that enough places are carried so that significant digits are not lost during execution.

Examples:

Assume as initial values: X = 2, Y = 10, Z = 15, TOT = 50, and SUB = 30.

Format 1:

```
ADD X TO TOT.                 results TOT = 52
ADD X, Y, Z, TO TOT, SUB.     results TOT = 77, SUB = 57
```

Format 2:

```
ADD X, Y GIVING TOT.              results TOT = 12
ADD X, Y, Z GIVING TOT, SUB.      results TOT = 27, SUB = 27
```

Format 3:

For Format 3, assume the following structures have the initial value in parentheses.

```
01  NOW                    01 LATER
    02  AL                     05  AL
      03  FO  (8)                06  FO  (10)
      03  RD  (10)               06  XY  (10)
    02  ME  (20)               05  AB  (10)
    02  HE  (5)                05  HE  (10)
```

Therefore the statement

ADD CORRESPONDING NOW TO LATER.

results in:

```
02  LATER
    05  AL
        06  FO  (18)
        06  XY  ((10)
    05  AB  (10)
    05  HE  (15)
```

The only data items whose values changed are FO and HE.

ALTER

The ALTER statement modifies a predetermined sequence of operations.

General Format:

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

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

Syntax Rules:

1. Each procedure-name-1, procedure-name-3, ..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

2. Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the PROCEDURE DIVISION.

General Rules:

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ..., so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2, procedure-name-4, ..., respectively. Modified GO TO statements in independent segments may, under some circumstances, be returned to initial states. Refer to Independent Segments in this section.

   All other uses of the ALTER statement are valid and are performed even if procedure-name-1, procedure-name-3 is in an overlayable fixed segment. Refer to Segmentation in this section.

CALL

The CALL statement causes control to be transferred from one object program to another, within the run unit.

General Format:

Format 1:

```
CALL { identifier-1 } [ USING  data-name-1 [ , data-name-2 ] ... ]
     { literal-1    }

[ ; ON OVERFLOW imperative-statement ]
```

Format 2:

```
CALL SYSTEM DUMP
```

Format 3:

```
CALL SYSTEM ZIPSB USING { identifier-2 }
                        { literal-2    }
```

Format 4:

```
CALL SYSTEM WFL USING { identifier-3 }
                      { literal-3    }
```

VERB FORMAT: CALL

Syntax Rules:

Format 1 Only:

1. Literal-1 must be a nonnumeric literal in the form "B" for a single file name, "B/C" for a multi-file-id and file-id, "B ON A" for a single file on a specific disk, or "B/C ON A" for a multi-file-id and file-id on a specific disk. Refer to the B 1000 Systems Software Operation Guide, Volume 1, for the formation of file names.

2. Identifier-1 must be defined as an alphanumeric data item.

3. 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.

4. Each of the operands in the USING phrase must have been defined as a data item in the FILE SECTION, WORKING-STORAGE SECTION, COMMUNICATION SECTION, or LINKAGE SECTION, and must have a level-number of 01 or 77. Data-name-1, data-name-2, ..., may be qualified when refering to data items defined in the FILE SECTION or the COMMUNICATION SECTION.

Format 3 Only:

5. Identifier-2 must be defined as an alphanumeric data item.

6. Literal-2 must be a non-numeric literal.

Format 4 only:

7. Literal-3 must be a non-numeric literal.

8. Identifier-3 must be an 01 level data item whose usage is display.

General Rules:

Format 1 Only:

1. 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.

2. The execution of a CALL statement causes control to pass to the called program.

3. A called program is in initial state the first time it is called within a run unit and the first time it is called after a CANCEL operation to the called program.

   On all other entries into the called program, the state of the program remains unchanged from the state when last exited by an EXIT PROGRAM statement. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

4. If, during the execution of a CALL statement, it is determined that the available portion of object time memory is incapable of accommodating the program specified in the CALL statement, and the ON OVERFLOW phrase is specified, no action is taken and the imperative-statement is executed.

   If the above condition exists and the ON OVERFLOW phrase is not specified, the run unit is suspended until such time as the necessary portion of object time memory to accommodate the program specified in the CALL statement is available.

5. Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program. Program A can CALL Program B, and Program B can CALL Program C, but Program C cannot CALL A or B, and Program B cannot CALL A.

6. The data-names, 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 data-names in the USING phrase of the CALL statement and the USING phrase in the PROCEDURE DIVISION header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling programs. The correspondence is identified by position, not by name. In the case of index-names, no such correspondence is established. Index-names in the called and calling programs always refer to separate indices.

7. The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in the last used state when the EXIT PROGRAM statement returns control to the calling program.

Example of Format 1:

Assume the called program name is PROGONE, and the PROCEDURE DIVISION header is:

```
PROCEDURE DIVISION USING A, B, C.
```

The calling program contains:

```
CALL "PROGONE" USING X, Y, A.
ADD X, Y, A GIVING TOTAL.
```

When the CALL is executed a correspondence is set up between the two sets of data-names such that (A and X), (B and Y), and (C and A) refer to the same data items regardless of the fact that both sets contain an A as a data-name. If PROGONE changes the values of A, B, and C to new values, these data items are available to the calling program in X, Y, and A, respectively.

When PROGONE is exited by a CANCEL or EXIT PROGRAM, control returns to:

```
ADD X, Y, A GIVING TOTAL.
```

VERB FORMAT: CALL

Formats 2, 3 and 4 Only.

8. The execution of a CALL SYSTEM statement causes control to pass to the specified routine within the operating environment.

9. DUMP specifies the point in the PROCEDURE DIVISION at which the operating system takes a "snapshot" of the program memory area.

10. ZIPSB specifies the point in the PROCEDURE DIVISION at which a job control instruction contained within the running object program is to be interpreted.

    The literal, or the data item referenced by identifier-2, must contain a value equivalent to the information contained in an operating system control instruction. Executing the program under a usercode has a direct influence on the value to be contained in the literal or data item. The syntax of the instruction is not checked by the COBOL74 compiler; any errors detected by the operating system have no direct effect on the calling program. The time at which the operating system performs the control instruction depends on availability of system resources.

    Example of Format 3:

    The following literal causes the operating system to list date information for the file (MSI)/RESERVES when the program is executed under the usercode (MSI) but gets an error message or information about the wrong file when not executed under the usercode:

    ```
    CALL SYSTEM ZIPSB USING "*P RD DF RESERVES"
    ```

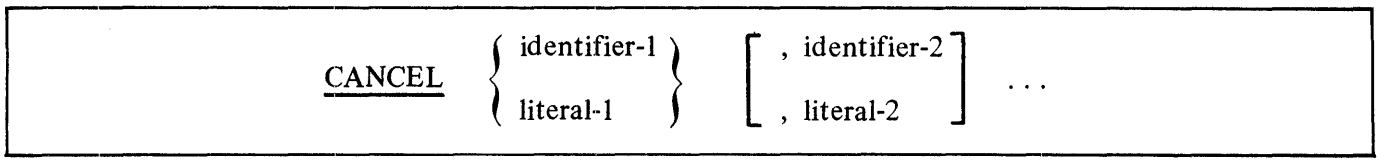
11. WFL (Work Flow Language) causes an independent job task to be initiated by submitting a WFL job to the WFL Compiler. After initiating the job task, the executing program does not wait for the iniated task to be completed but immediately proceeds to execute the next statement.

    The contents of identifier-3 or literal-3 must be a complete WFL job (see CSG Work Flow Language Standard 1955 2843) or any single WFL statement allowed from the ODT (see CSG ODT Command Standard 1955 2835). The syntax of the WFL job deck is not checked by the COBOL compiler; any errors detected by the WFL Compiler will have no direct effect on the calling program.

CANCEL

The CANCEL statement releases the memory areas occupied by the referred to program.

General Format:



Syntax Rules:

1. Literal-1, literal-2, ..., must be a nonnumeric literal in the form "B" for a single file name, "B/C" for a multi-file-id and file-id, "B ON A" for a single file on a specific disk, or "B/C ON A" for a multi-file-id and file-id on a specific disk. Refer to the B 1000 Systems Software Operation Guide, Volume 1, for the formation of file names.

2. Identifier-1, identifier-2, ..., must each be defined as an alphanumeric data item such that values can be a program name.

General Rules:

1. Subsequent to the execution of a CANCEL statement, the program that has been cancelled ceases to have any logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program results in that program being initiated in the initial state.

2. A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.

3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent CALL statement.

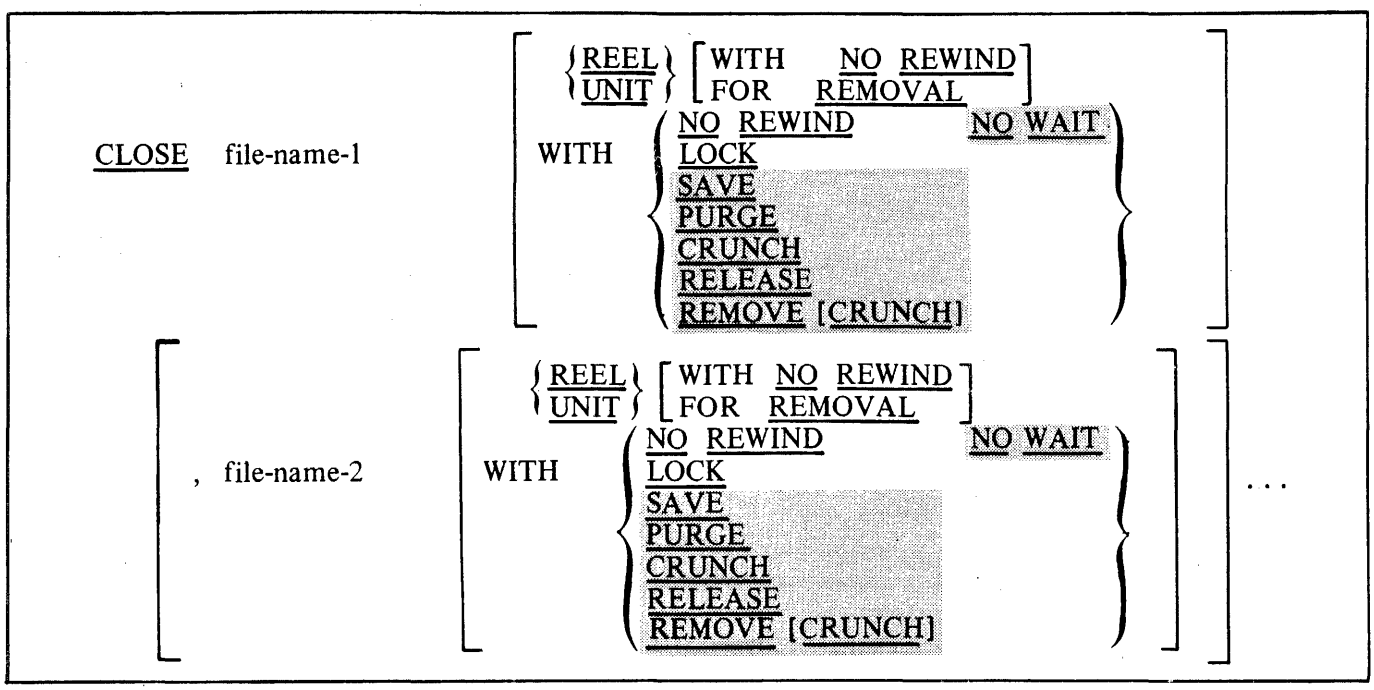
4. A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

5. No action is taken when a CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is, at present, cancelled. Control passes to the next statement.

CLOSE

The CLOSE statement terminates the processing of a file, or a reel/unit of a file, and also may specify the disposition of the file and the device to which it is assigned.

General Format:



Syntax Rules:

1. The REEL or UNIT phrase must only be used for sequential files.
2. The files referenced in the CLOSE statement need not all have the same organization or access.
3. The WITH NO WAIT phrase can be specified only for port files.

General Rules:

Except where otherwise stated in the following general rules, the terms reel and unit are synonymous and completely interchangeable in the CLOSE statement.

1. A CLOSE statement may only be executed for a file in an open mode.
2. For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:
   a. Sequential non-reel/unit. A sequential file whose input or output medium is such that the concepts of rewind and reel/unit have no meaning. This category includes mass storage files. A CLOSE statement executed for a non-reel/unit file may affect the disposition of

the device to which it is assigned. However, the assignment and control of the physical device on which certain non-reel/unit files (including mass storage files) reside, is regarded to be the exclusive right of the operating system. Consequently, CLOSE statements executed for these files affect only the disposition and association of the physical file with the logical file, not the disposition of the physical device.

b. Sequential single-reel/unit. A sequential file that is entirely contained on one reel/unit.

c. Sequential multi-reel/unit. A sequential file that is contained on more than one reel/unit.

d. Indexed and Relative Files. Indexed and Relative files are labeled mass storage files.

3. The results of executing each type of CLOSE for each category of file are summarized in table 7-3.

Table 7-3. Relationship of Categories of Files and Formats of the CLOSE Statement

CLOSE Statement Format	File Category			
	Sequential Non-Reel/Unit	Sequential Single-Reel/Unit	Sequential Multi-Reel/Unit	Indexed & Relative Files
CLOSE	C,H	C,G,H	C,G,A,H	C1,H
CLOSE WITH LOCK	C,E,I,J	C,G,E,I,J	C,G,E,A,I,J	C1,I,E
CLOSE WITH NO REWIND	X	C,B,H	C,B,A,H	
CLOSE REEL/UNIT	X	X	F,G	
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G,J	
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B	
CLOSE WITH RELEASE	C,I,J	C,I,J,G	C,G,A,I,J	C1,J
CLOSE WITH REMOVE	C,J,N	X	X	C1,I,N
CLOSE WITH SAVE	C,I,L	X	X	C1,I,L
CLOSE WITH CRUNCH	C,I,M	X	X	
CLOSE WITH PURGE	C,K	C,G,K	C,G,K,A	C1,K
CLOSE WITH REMOVE and CRUNCH	C,I,J,M,N	X	X	

VERB FORMAT: CLOSE

The definitions of the symbols in the table are given in the following pages. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

'A' Previous Reels/Units Unaffected

Input Files

All reels/units in the file prior to the current reel/unit are processed according to the standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last in the file, the reels/units in the file following the current one are not processed.

Output Files:

All reels/units in the file prior to the current reel/unit are processed according to the standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

'B' No Rewind of Current Reel

The current reel/unit is left in its current position.

'C' Close File

Input Files and Input-Output Files:

If the file is positioned at the end and label records are specified for the file, the labels are processed according to the standard label convention.

If label records are specified, the label records must be present. Label records that are not specified are ignored.

Closing operations are executed. If the file is positioned at the end and label records are not specified for the file, label processing does not take place, but other closing operations are executed. If the file is positioned other than at the end, the closing operations are executed, but there is no ending label processing.

Output Files

If label records are specified for the file, the labels are processed according to the standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations are executed. If label records are not specified for the file, label processing does not take place, but other closing operations are executed.

'C1' CLOSE FILE

The CLOSE operation marks the logical file as closed.

'D' Reel/Unit Removal

The current reel/unit is rewound and released to the system. However, the reel or unit may be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

'E' File Lock

The logical file is marked so it cannot be reopened during execution of the program. If the file is a mass storage file, it is made a permanent file before being made unavailable. If the file is assigned to a tape device, the physical unit is made not ready.

'F' Close Reel/Unit

Input Files:

The following operations take place:

1. A reel/unit swap.

2. The standard beginning reel/unit label procedure is executed.

The next executed READ statement for that file makes available the next data record on the new reel/unit.

Output Files

The following operations take place:

1. The standard ending reel/unit label procedure is executed.

2. A reel/unit swap.

3. The standard beginning reel/unit label procedure is executed.

The next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

'G' Rewind

The current reel is positioned at its physical beginning.

'H' File Retention

The association between the logical file and the physical file is retained. Subsequent reopening of the file may not require the operating system to search for the physical file.

VERB FORMAT: CLOSE

'I' File Release

The association between the logical file and the physical file is severed. The areas of memory allocated for buffers may be released to the system. A new disk file is not made permanent. A new tape file is made permanent.

'J' Device Release

If the device to which the file was assigned is allowed to be controlled by the object program, it is released as available to the system.

'K' File Purge

This disposition is only valid for files assigned to tape or mass storage devices. If the file is assigned at object time to a tape device, the reel is rewound. If it has a write ring, a scratch label is written on it, and the device is released as available to the system. If the file is a permanent mass storage file, the file name is removed from the system directory, and the mass storage area occupied by the file is released as available to the system.

'L' File Save

This disposition is only valid for mass storage files. The file is made a permanent file.

'M' File Crunch

This disposition is only valid for mass storage files. The file is made a permanent file, and any unused portions of mass storage areas allocated for the file are released as available to the system. The file cannot subsequently be extended by opening the file OPEN EXTEND.

'N' File Remove

This disposition is only valid for mass storage files. The file is made a permanent file. If a file with an identical name exists in the system directory, it is removed.

'X' Illegal

This is an illegal combination of a CLOSE option and a file category. If the CLOSE statement specifies the REEL/UNIT phrase, the CLOSE statement has no effect, and the file is not closed. If the CLOSE statement does not specify the REEL/UNIT phrase, any optional disposition is ignored, but the file is closed.

4. If a file is in the open mode when a STOP RUN statement is executed, when a CANCEL statement is executed for the program containing that file, or when an abnormal termination occurs, the action taken is to close the file as if a simple CLOSE statement had been executed.

5. If the OPTIONAL phrase has been specified for the file in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION and the file is not present, the standard end-of-file processing is not performed for that file.

6. If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT or MERGE statements with the USING or GIVING phrases) that references that file can be executed, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

7. The WITH NO REWIND, FOR REMOVAL, SAVE, PURGE, CRUNCH, RELEASE, and REMOVE phrases have no effect at object time if they do not apply to the storage media on which the file resides.

8. The execution of a CLOSE statement has no effect upon the contents or the availability of the file record area.

9. For a port file having an ACTUAL KEY clause, the value of the ACTUAL KEY determines which subfile of the file is to be closed. If the ACTUAL KEY value is non-zero, only the specified subfile is closed. If the ACTUAL KEY value is zero or if the ACTUAL KEY clause is not specified, all opened subfiles are closed.

10. A CLOSE statement with no phrase specified causes the program to wait until the file is closed before resuming execution. The possibility of this suspension is prevented for port files by specifying the WITH NO WAIT phrase. In this case, control is returned to the next statement without waiting for the close operation to be completed.

COMPUTE

The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

General Format:

```
COMPUTE identifier-1 [ROUNDED] [ , identifier-2 [ROUNDED]] ...
    = arithmetic-expression [; ON SIZE ERROR imperative-statement ]
```

Syntax Rules:

1. Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

General Rules:

1. Additional rules and explanations relative to this statement are given in the appropriate paragraphs. Refer to Intermediate Data Item, ROUNDED Phrase, SIZE ERROR Phrase, Arithmetic Statements, Overlapping Operands, and Multiple Results in Arithmetic Statements in this section.

2. An arithmetic expression consisting of a single identifier or literal provides a method of setting the values of identifier-1, identifier-2, and so forth, equal to the value of the single identifier or literal. (Refer to Arithmetic Expressions in this section.)

3. If more than one identifier is specified for the result of the operation (those identifiers preceding the equal sign), the value of the arithmetic expression is computed, and the resultant value is then stored as the new value of each successive identifier, such as identifier-1, identifier-2, and so on.

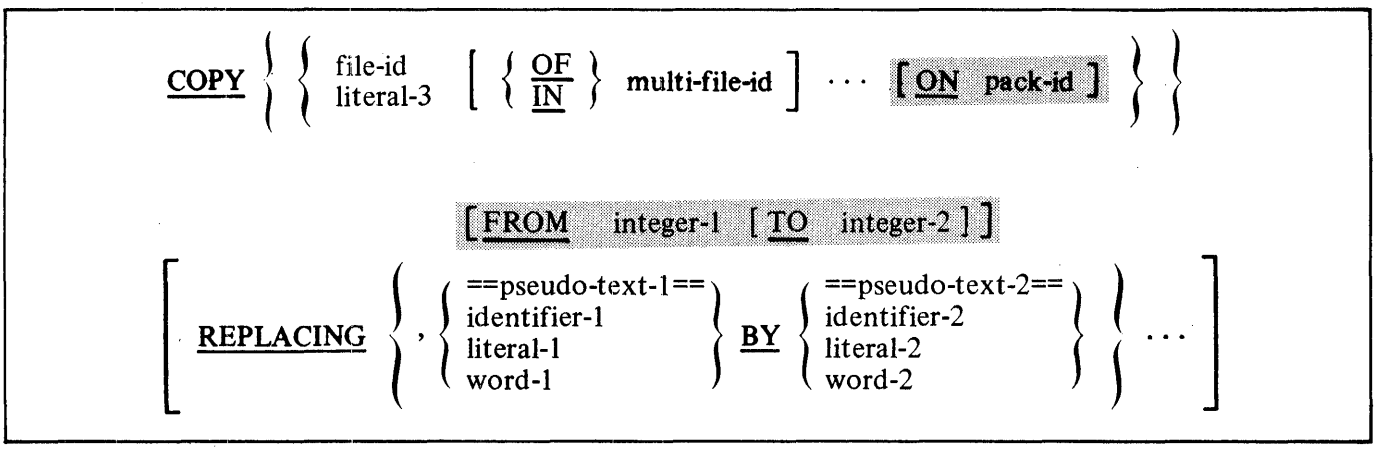
COPY

The COPY statement incorporates text from a library into a COBOL74 source program.

COBOL74 libraries contain library texts that are available to the compiler for copying at compile time. The effect of the interpretation of the COPY statement is to insert text into the source program, where it is treated by the compiler as part of the source program. Other than the use of dollar options, (NEW, DELETE, MERGE, and so forth; refer to section 11, COBOL74 Compiler Control), libraries must be created by means other than the COBOL74 compiler.

Additionally, the COPY statement can replace all occurrences of a given literal, identifier, word or group of words in the library text, with alternate text, during the copying process.

General Format:



Syntax Rules:

1. If more than one COBOL74 library is available during compilation, file-id must be qualified by the multi-file-id identifying the COBOL74 library in which the text associated with file-id resides.

   Within one COBOL74 library, each file-id must be unique.

   File-id specifies the external identification of a file in the COBOL74 library.

   Multi-file-id specifies the external identification of a volume-id, or directory-id, which is the name of the COBOL74 library.

   Pack-id specifies the name of the disk on which the library file resides.

2. The COPY statement must be preceded by a space and terminated by the separator period. If the COPY statement appears in a "debug line," the word COPY cannot start in column 8. See section 10 in this manual.

VERB FORMAT: COPY

3. Pseudo-text-1 must not be null, and may not consist solely of the character space(s) or of comment lines. Psuedo-text-1 represents one or more COBOL74 words. It may not consist of a portion of a COBOL74 word.

4. Pseudo-text-2 may be null.

5. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. However, both characters of the pseudo-text delimiter (–) must be on the same line. For more details refer to Indicator Area for continuation of lines, Section 3.

6. Word-1 or word-2 may be any single COBOL74 word.

7. A COPY statement may occur in the source program anywhere a character-string or a separator may occur, except that a COPY statement must not occur within a COPY statement nor in the library text.

8. Literal-3 must be a nonnumeric literal in the form "B" for a single file name, "B/C" for a multi-file-id and file-id, "B ON A" for a single file on a specific disk, or "B/C ON A" for a multi-file-id and file-id on a specific disk. Refer to the B 1000 Systems Software Operation Guide, Volume 1, for the formation of file names.

9. Integer-1 may be zero.

General Rules:

1. The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

2. The effect of processing a COPY statement is that the library text associated with file-id is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

3. If the REPLACING phrase is not specified, the library text is copied unchanged.

   If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of pseudo-text-1, identifier-1, word-1, and literal-1 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, or literal-2.

4. For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.

5. The comparison operation that determines text replacement is explained in the following paragraphs.

   Any separator comma, semicolon and/or space(s) preceding the leftmost library text-word is copied into the source program. Starting with the leftmost library text-word and the first pseudo-text-1, identifier-1, word-1, or literal-1 that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text-words.

Pseudo-text-1, identifier-1, word-1, or literal-1 match the library text only if, the ordered sequence of text-words that forms pseudo-text-1, identifier-1, word-1, or literal-1 is equal, character for character, to the ordered sequence of library text-words. For purposes of matching, each occurrence of a separator comma or semicolon in pseudo-text-1 or in the library text is considered to be a single space, except when pseudo-text-1 consists solely of either a separator comma or semicolon, in which case it participates in the match as a text-word. Each sequence of one or more space separators is considered to be a single space.

If no match occurs, the comparison is repeated with each next successive pseudo-text-1, identifier-1, word-1, or literal-1, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text-word is copied into the source program. The next successive library text-word is then considered as the leftmost library text-word, and the comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

Whenever a match occurs between pseudo-text-1, identifier-1, word-1, or literal-1 and the library text, the corresponding pseudo-text-2, identifier-2, word-2, or literal-2 is placed into the source program. The library text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost library text-word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

The comparison operation continues until the rightmost text-word in the library text has either participated in a match or been considered as a leftmost library text-word and participated in a complete comparison cycle.

6. A comment line occurring in the library text and pseudo-text-1 is interpreted, for purposes of matching, as a single space. Comment lines appearing in pseudo-text-2 and library text are copied into the source program unchanged.

7. Debugging lines are permitted within library text and pseudo-text-2. Debugging lines are not permitted within pseudo-text-1; text-words within a debugging line participate in the matching rules as if the "D" did not appear in the indicator area. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement appears as though it were specified on debugging lines with the following exception. Comment lines in library text appear as comment lines in the resultant source program.

8. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

9. The syntactic accuracy of the library text cannot be independently determined. The syntactic accuracy of the entire COBOL74 source program cannot be determined until all COPY statements have been completely processed.
10. Library text must conform to the rules for COBOL74 reference format.

11. For purposes of compilation, text-words, after replacement, are placed in the source program according to the rules for reference format. Refer to Field Definitions for reference format, section 3.

VERB FORMAT: COPY

12. The optional FROM clause provides a method of copying only a portion of a source file. For example, a programmer can copy certain portions of a master source file into many COBOL74 programs.

13. If literal-3 is specified, it is the external identification of the library text.

14. If file-id is specified and the optional multi-file-id and pack-id clauses are not present, the file-id is the internal name of the library text and the default external identification.

Examples: The following items illustrate uses of the COPY statement.

```
FD INV-FILE COPY "MASTER/INVENTORY ON MYPACK".

FD PAYROLL COPY "LASTYR/PAYROLL" REPLACING "1979" BY "1980".

COPY "INPUTFILE".

COPY "SOFTWARE/FILE" REPLACING -MARK 10.0- BY -RELEASE 11.0-.

COPY "THISFILE ON MYPACK" REPLACING AAA BY INPUT-DATA.

COPY FILEFDS OF LIB ON USER1 FROM 3500 to 4800.
```

DELETE

The DELETE statement logically removes a record from a mass storage file.

General Format:

```
DELETE file-name RECORD [ ; INVALID KEY imperative-statement ]
```

Syntax Rules:

1. The INVALID KEY phrase must not be specified for a DELETE statement which references a file in sequential access mode.

2. The INVALID KEY phrase must be specified for a DELETE statement which references a file not in sequential access mode and for which an applicable USE procedure is not specified.

3. A DELETE statement cannot be used with a SORT or MERGE file or with a Sequential File.

General Rules:

1. The associated file must be open in the I-O mode at the time of the execution of the DELETE statement. Refer to the OPEN Statement in this section.

2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The record that was accessed by that READ statement is logically removed from the file.

3. For a file in random or dynamic access mode, if the file is Relative, the record identified by the Relative Key data item is deleted; if the file is Indexed, the record identified by the Prime Record Key is deleted. If the file does not contain the record specified by the key, an INVALID KEY condition exists. Refer to the Invalid Key condition in section 5.

4. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed. The block control information, BCI, included at the start of each physical record block, is updated to show that the record slot occupied by the deleted record is now available to be reused when records are added to the file.

5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

6. The current record pointer is not affected by the execution of a DELETE statement.

7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated. Refer to I-O Status in section 5.

DISABLE

The DISABLE statement inhibits data transfer between the specified output queue and destinations for output or between specified sources and input queue for input.

General Format:

```
DISABLE { INPUT [TERMINAL] } cd-name WITH KEY { identifier-1 }
        { OUTPUT           }                  { literal-1    }
```

Syntax Rules:

1. Cd-name must reference an input CD when the INPUT phrase is specified.
2. Cd-name must reference an output CD when the OUTPUT phrase is specified.
3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric, containing at least 1 but not more then 10 characters.

General Rules:

1. The DISABLE INPUT <cd-name> statement provides a logical disconnection between the Data Communication Subsystem and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the DISABLE statement is not required in this program. The logical path for the transfer of data between the COBOL74 programs and the Data Communication Subsystem is not affected by the DISABLE statement.

   When the logical disconnection specified by the DISABLE statement is already in existence, is to be handled by some means external to this program, or is denied, the status key data item in the area referenced by CD-name is updated. Refer to Communication Description Structure (CD) in section 6.

2. When the INPUT phrase with the optional word TERMINAL is specified, input to a specific terminal is disabled. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queue and subqueues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are deactivated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the data referenced by cd-name are deactivated.

5. The KEY phrase provides a password facility.

   An additional area of interaction between the DISABLE and the Data Communication Subsystem is the password. Within the message is a 10-byte field which contains a character string. It is the option of an MCS (Message Control System), to compare the password specified by literal-1 or identifier-1 to a master password. The data referenced by literal-1 or identifier-1 is transferred to the MCS according to the rules of the MOVE statement. The password is received in an alphanumeric data item 1 to 10 characters in length.

DISPLAY

The DISPLAY statement causes low volume data to be transferred to an appropriate hardware device.

General Format:

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

[ UPON  mnemonic-name ]
```

Syntax Rules:

1. The mnemonic-name is associated with a hardware device in the SPECIAL-NAMES paragraph in the ENVIRONMENT DIVISION and must be associated with the hardware-name ODT.
2. Each literal may be any figurative constant, except ALL.
3. If the literal is numeric, then it must be an unsigned integer.
4. DISPLAY cannot use an item declared in the LINKAGE SECTION.
5. DISPLAY cannot use the conceptual data items DATE, DAY, TODAYS-DATE, TODAYS-NAME, TIMER AND TIME.
6. DISPLAY cannot use an index data item.

General Rules:

1. The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed.
2. The maximum number of characters that can be transmitted is unlimited.
3. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.
4. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered. If the data transferred does not fit on one line, carriage returns and line feeds are supplied so that the data is extended to other lines of print.
5. If the UPON phrase is not used, the device used is the ODT.

DIVIDE

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

General Formats:

Format 1:

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

Format 2:

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

Format 3:

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

VERB FORMAT: DIVIDE

Syntax Rules:

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules:

1. Additional rules and explanations relative to this statement are given in the appropriate paragraphs. Refer to Intermediate Data Item, Arithmetic Statements, Overlapping Operands, and Multiple Results in Arithmetic Statements in section 7. Also, refer to General Rules 4 through 6 below for a discussion of the ROUNDED phrase and the SIZE ERROR phrase as they pertain to Format 3.

2. When Format 1 is used, the value of identifier-2 is divided by either the value of identifier-1 or literal-1. The value of the dividend (identifier-2) is replaced by this quotient; and similarly for identifier-1 or literal-1 and identifier-3, and so forth.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2, and the result is stored in identifier-3, identifier-4, and so on.

4. Format 3 is used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL74 is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded.

5. In Format 3, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described above. Appropriate decimal alignment and truncation (not rounding) is performed for the content of the data item referenced by identifier-4, as needed.

6. When the ON SIZE ERROR phrase is used in Format 3, the following rules pertain:

   a. If the size error occurs on the quotient, no remainder calculation is meaningful. The contents of the data items referenced by both identifier-3 and identifier-4 remain unchanged.

   b. If the size error occurs on the remainder, the contents of the data items referenced by identifier-4 remain unchanged. However, as with other instances of multiple results of arithmetic statements, the user is responsible for performing the necessary analysis to determine which situation has actually occurred.

ENABLE

The ENABLE statement allows data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

General Format:

```
ENABLE { INPUT [TERMINAL] } cd-name WITH KEY { identifier-1 }
       { OUTPUT           }                  { literal-1    }
```

Syntax Rules:

1. Cd-name must reference an input CD when the INPUT phrase is specified.
2. Cd-name must reference an output CD when the OUTPUT phrase is specified.
3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric containing 1 to 10 characters, inclusive.

General Rules:

1. The ENABLE INPUT <cd-name> statement provides a logical connection between the Data Communication Subsystem and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the ENABLE statement is not required. The logical path for the transfer of data between the COBOL74 programs and the Data Communication Subsystem is not affected by the ENABLE statement.

   When the logical connection specified by the ENABLE statement is already in existence, (only if a RECEIVE is done instead), or is to be denied, the status key data item in the area referenced by CD-name is updated. Refer to the Communication Description Structure (CD) in section 6.

2. The INPUT phrase with the optional word TERMINAL is used to ENABLE a previously disabled (DISABLE) logical path between the source and all associated queues and subqueues. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful.

3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queue and subqueues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE3) of the area referenced by cd-name are activated.

4. When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are activated.

VERB FORMAT: ENABLE

5. The KEY phrase provides a password facility.

   An additional area of interaction between the ENABLE and the Data Communication Subsystem is the password. Within the message is a 10-byte field which contains a character string. It is the option of the MCS (Message Control System), to compare the password specified by literal-1 or identifier-1 to a master password. The data referenced by literal-1 or identifier-1 is transferred to the MCS according to the rules of the MOVE statement. The password is received in an alphanumeric data item of 1 to 10 characters in length.

EXIT

The EXIT statement provides a means of documenting the logical end point for a series of sections or paragraphs that may be executed under the control of a PERFORM statement.

General Format:

```
EXIT.
```

Syntax Rules:

1. The EXIT statement must appear in a sentence alone.
2. The EXIT sentence must be the only sentence in the paragraph.

General Rules:

1. An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

EXIT PROGRAM

The EXIT PROGRAM statement marks the logical end of a called program.

General Format:

```
EXIT PROGRAM.
```

Syntax Rules:

1. The EXIT PROGRAM statement must appear in a sentence alone.
2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rules:

1. An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. If the program is not called, control passes through the EXIT PROGRAM statement to the first sentence of the next paragraph.

VERB FORMAT: GO TO

GO TO

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

General Format:

Format 1:

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

Format 2:

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

Syntax Rules:

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

2. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.

3. A Format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.

4. If a GO TO statement, represented by Format 1, appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules:

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

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

3. When a Format 2 GO TO statement is executed, control is transferred to the procedure-name whose ordinal position in the list following the GO TO corresponds to the value of the identifier being 1, 2, ..., n. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

IF

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is TRUE or FALSE.

General Format:

```
                                  ( statement-1      )   ( ; ELSE statement-2      )
IF condition ;                    (                  )   (                         )
                                  ( NEXT SENTENCE    )   ( ; ELSE NEXT SENTENCE    )
```

Syntax Rules:

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either may be followed by a conditional statement.

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

General Rules:

1. When an IF statement is executed, the following transfers of control occur:

   a. If the condition is TRUE, statement-1 is executed, if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. Refer to Categories of Statements in this section. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   b. If the condition is TRUE and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   c. If the condition is FALSE, statement-1 or NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. Refer to Categories of Statements in this section. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

   d. If the condition is FALSE, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

2. Statement-1 and/or statement-2 may contain an IF statement. In this case, the IF statement is nested. The Mark 11.0 B 1000 COBOL74 allows 45 nested IF statements.

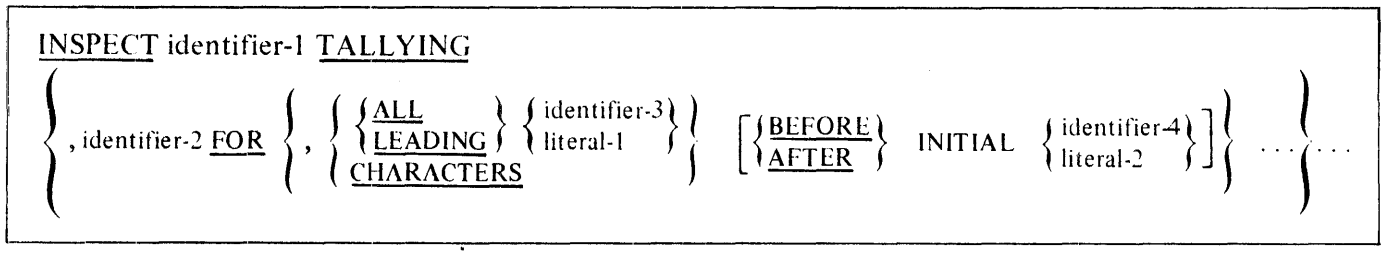
   IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

INSPECT

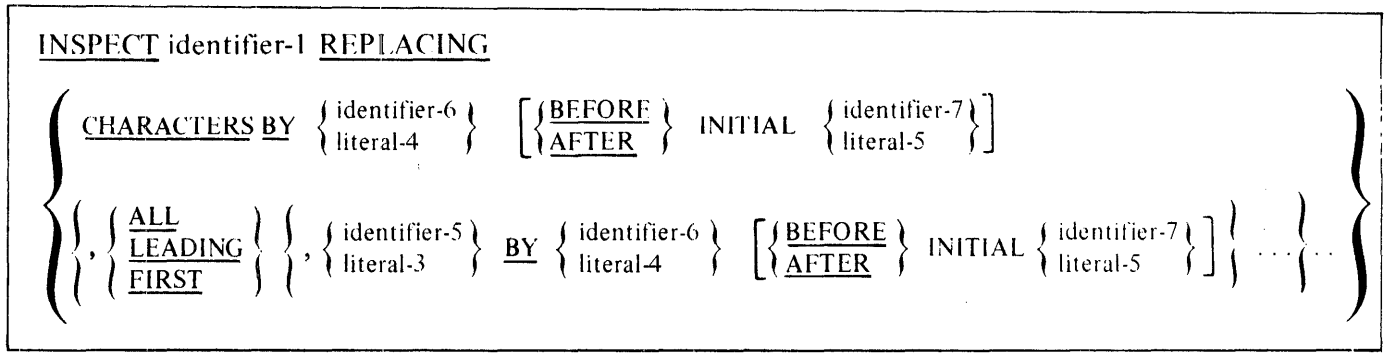
The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), or tally and replace (Format 3) occurrences of single characters or groups of characters in a data item.

General Formats:

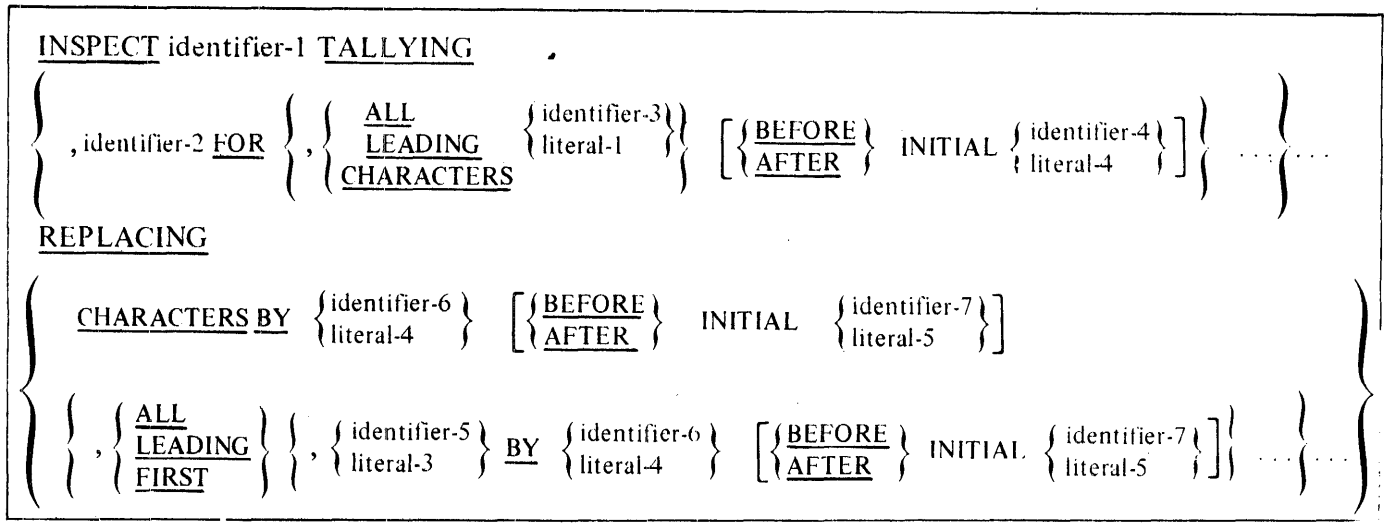
Format 1:



Format 2:



Format 3:



VERB FORMAT: INSPECT

Syntax Rules:

All Formats

1. Identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as usage is DISPLAY.

2. Identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric, or numeric item described (either implicitly or explicitly) as usage is DISPLAY.

3. Each literal must be nonnumeric and may be any figurative constant, except ALL.

Formats 1 and 3 only

4. Identifier-2 must reference an elementary numeric data item.

5. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one character data item.

Formats 2 and 3 only

6. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

7. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

8. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

General Rules:

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of class, and proceeds from left to right to the rightmost character position as described in General Rules 4 through 6 which follow. If identifier-1 is signed, the sign character is not inspected.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are treated as follows:

   a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

   b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though redefined as alphanumeric (refer to General Rule 2a) and the INSPECT statement had been written to reference the redefined data item.

c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 is described as signed numeric, the sign character of the data item is not inspected.

3. In General Rules 4 through 11 all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3).

5. The comparison operation, to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced, occurs as follows:

   a. The operands of the TALLYING and REPLACING phrases are considered in the order specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match only if equal, character for character.

   b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

   c. Whenever a match occurs, tallying and/or replacing takes place as described in General Rules 8 through 10. The character position in the data item referenced by identifier-1, immediately to the right of the rightmost character position that participated in the match, is now considered to be the leftmost character position of the data item referenced by identifier-1. The comparison cycle starts again with the first literal-1, literal-3.

   d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

   e. If the CHARACTERS phrase is specified, an implied one character operand participates in the cycle described in steps 5a through 5d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is always considered to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in General Rule 5 is affected by the BEFORE and AFTER phrases as follows:

   a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in General Rule 5.

VERB FORMAT: INSPECT

b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

Example:

Assume that identifier-1 contains ABCDEFGH.

Phrase	Range of Inspection
none	A through H
BEFORE "G"	A through F
BEFORE "DEF"	A through C
BEFORE "FGE"	A through H
BEFORE "A"	Null

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5, within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Example:

Assume that identifier-1 contains ABCDEFGH.

Phrase	Range of Inspection
none	A through H
AFTER "C"	D through H
AFTER "BCD"	E through H
AFTER "DCE"	Null
AFTER "H"	Null

VERB FORMAT: INSPECT

Format 1:

7. The contents of the data item referenced by identifier-2 are not initialized by the execution of the INSPECT statement.

8. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 are incremented by 1 for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

      INSPECT word TALLYING count FOR ALL "BA".

      word = BARBARA count = 2
      word = HBAABCT count = 1

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 are incremented by 1 for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

      INSPECT word TALLYING count-1 FOR LEADING "N" BEFORE INITIAL "O", count-2 FOR LEADING "E" BEFORE INITIAL "R".

      word = PATTERSON count-1 = 0 count-2 = 1
      word = MCKINNON count-1 = 2 count-2 = 0

   c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 are incremented by 1 for each character matched (refer to General Rule 5e) within the contents of the data item referenced by identifier-1.

      INSPECT word TALLYING count FOR CHARACTERS.

      word = KOMP count = 4
      word = HARRINGTON count = 10

VERB FORMAT: INSPECT

Format 2:

9. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.
10. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched (refer to General Rule 5e) in the contents of the data item referenced by identifier-1 is replaced by literal-4.

    INSPECT word REPLACING CHARACTERS BY "M".

    word = MMMGOOD count = MMMMMMM
    word = KOUNT210 count = MMMMMMMM

    b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

    INSPECT word REPLACING ALL "R" BY "Z" BEFORE INITIAL "B".

    word = ROBERTS after = ZOBERTS
    word = RCHRBR after = ZCHZBR

    c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

    INSPECT word REPLACING LEADING "S" BY "X" AFTER INITIAL "O".

    word = SPORTS after = SPORTX
    word = CROSS after = CROXX

    d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

    INSPECT word REPLACING FIRST "A" BY "S".

    word = ABCA after = SBCA
    word = TVANAR after = TVSNAR

VERB FORMAT: INSPECT

Format 3:

11. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written: one statement is a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement. The other statement is a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement; the general rules given for matching and replacing apply to the Format 2 statement.

Examples:

INSPECT word TALLYING count-1 FOR LEADING "L" BEFORE INITIAL "A", count-2 FOR LEADING "A" BEFORE INITIAL "L".

```
word = LARGE      count-1 = 1    count-2 = 0
word = ANALYST    count-1 = 0    count-2 = 1
```

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

```
word = CALLAR     count = 2    after = CALLAR
word = SALAMI     count = 1    after = SALEMI
word = LATTER     count = 1    after = LETTER
```

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

```
word = ARXAX      after = GRXAX
word = HANDAX     after = HGNDGX
```

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

```
word = ADJECTIVE    count = 6    after = BDJCTIVE
word = JACK         count = 3    after = JBCK
word = JUJMAB       count = 5    after = JUJMBB
```

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

```
word = RXXBQWY      after = RYYZQQY
word = YZACDWBR     after = YZACDWZR
word = RAWRXEB      after = RAQRYEZ
```

VERB FORMAT: INSPECT

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

before: 1 2 X Z A B C D

after: B B B B B A B C D

INSPECT word REPLACING ALL "S" BY "H", "B" BY "A", "P" BY "L", "C" BY "E".

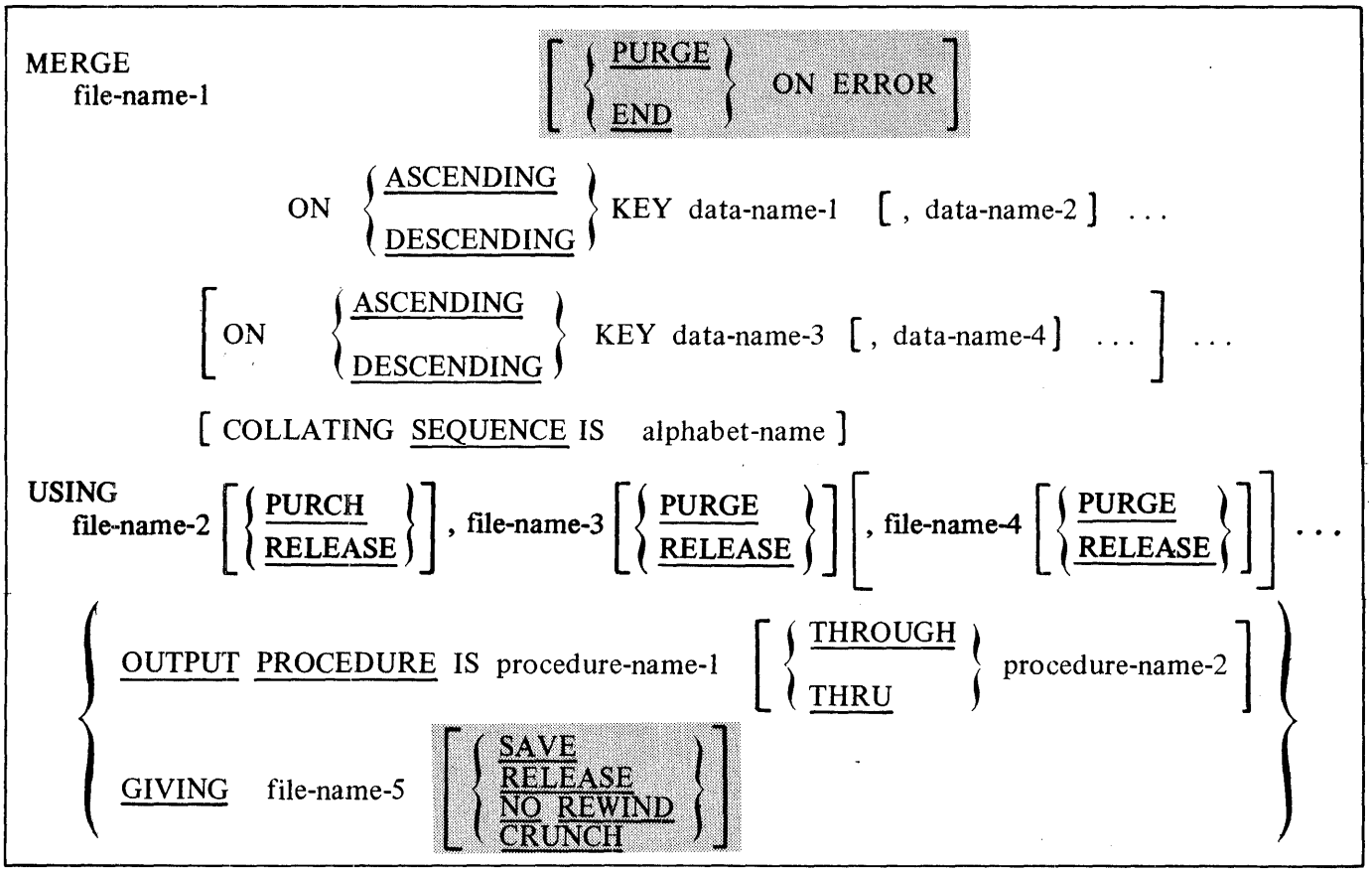
word = SBPC after = HALE
word = HOPPOWPOG after = HOLLOWLOG

VERB FORMAT: MERGE

MERGE

The MERGE statement combines two or more files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or an output file.

General Format:



Syntax Rules:

1. File-name-1 must be described in a sort-merge file description (SD) entry in the DATA DIVISION.

2. Procedure-name-1 represents the name of an output procedure.

3. File-name-2, file-name-3, file-name-4, and file-name-5 must be described in a file description entry (FD), not in a sort-merge file description entry (SD), in the DATA DIVISION. The actual size of the logical record(s) described for file-name-2, file-name-3, file-name-4, and file-name-5 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause an equal number of character positions to be allocated for the corresponding records.

VERB FORMAT: MERGE

4. The words THRU and THROUGH are equivalent.

5. Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names may all be described within one of the record descriptions or in any combination of record descriptions. It is not necessary to redescribe the KEY data-names in each record description.

   e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

6. No more than one file-name from a multiple file reel can appear in the MERGE statement.

7. File-names must not be repeated within the MERGE statement.

8. MERGE statements may appear anywhere except in the Declarative portion of the PROCEDURE DIVISION or in an input or output procedure associated with a SORT or MERGE statement.

9. As many as eight file-names may be specified in the USING clause.

General Rules:

1. The MERGE statement merges all records contained on file-name-2, file-name-3, and file-name-4. The files referenced in the MERGE statement must not be open at the time the MERGE statement is executed. These files are automatically opened and closed by the MERGE operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file.

2. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to division into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, and so forth.

   a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

   b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

   b. Second, the collating sequence established as the program collating sequence.

4. The output procedure must consist of one or more paragraphs or sections that appear contiguously in a source program and do not form a part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time in merged order, from file-name-1.

The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any transfers of control to points outside the output procedure. ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL74 statements that cause an implied transfer of control to declaratives are allowed.

   b. The output procedures must not contain any SORT or MERGE statements.

   c. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the output procedures. ALTER, GO TO, and PERFORM statements in the remainder of the PROCEDURE DIVISION are not permitted to refer to procedure-names within the output procedures.

5. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last paragraph or section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

6. If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

7. When, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one data record are equal to the contents of the corresponding key data items of one or more other data records, the order of return of these records:

   a. Follows the order of the associated input files as specified in the MERGE statement.

   b. Is such that all records associated with one input file are returned prior to the return of records from another input file.

VERB FORMAT: MERGE

8. When the records in the files referenced by file-name-2, file-name-3, ... are not ordered as described in the ASCENDING or DESCENDING KEY clauses, MERGE takes place as previously described but with all improperly ordered data records being placed on the output file or returned to the output procedure immediately after being read from respective input files. As a result, when such a condition exists, the output from the MERGE statement is not in a strict ASCENDING or DESCENDING KEY order.

9. The ON ERROR option is provided to allow programmers limited control over irrecoverable parity errors when an OUTPUT PROCEDURE is not present in a program. PURGE causes all records in a block containing an irrecoverable parity error to be dropped, and processing is continued after an ODT message giving the relative position in the file of the bad block is displayed. END causes a program termination and is assumed if no option is specified.

10. The PURGE and RELEASE options specify the type of file close for the USING files, file-name-2, file-name-3, file-name-4, and so forth. The SAVE, RELEASE, NO REWIND, and CRUNCH options specify the type of file close for the GIVING file, file-name-5. Refer to the CLOSE statement in this section.

MOVE

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

General Format:

Format 1:

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

Format 2:

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

Syntax Rules:

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

2. CORR is an abbreviation for CORRESPONDING.

3. When the CORRESPONDING phrase is used, both identifiers must be group items.

4. An index data item cannot appear as an operand of a MOVE statement. Refer to the USAGE clause in section 6.

General Rules:

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in the CORRESPONDING phrase. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

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

VERB FORMAT: MOVE

Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands. The result of the statement:

```
MOVE a (b) TO b, c (b)
```

is equivalent to:

```
MOVE a (b) TO temp
MOVE temp TO b
MOVE temp TO c (b)
```

Temp is an intermediate result item provided by the compiler.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause in section 6. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.
   The following rules apply to an elementary move between the categories:

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

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

   c. A noninteger numeric literal or a noninteger numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to the rules given in General Rule 4.

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

   a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined under Standard Alignment Rules in section 2. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign is not moved. If the operational sign occupied a separate character position (refer to the SIGN clause in section 6), that character is not moved and the size of the sending item is considered to be one less than the actual size (in terms of standard data format characters).

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules, except where zeroes are replaced because of editing requirements.

   1) When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. Refer to the SIGN clause in section 6. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

   2) When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

   3) When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area is filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in General Rule 4 of the OCCURS clause in section 6.

6. The validity of the various types of MOVE statements is summarized in table 7-4. The general rule reference (for example, 4c) indicates the rule that prohibits the move or the behavior of a valid move.

Table 7-4. A Valid MOVE Statement

Category of Sending Data Item		Category of Receiving Data Item		
		Alphabetic	**Alphanumeric Edited** **Alphanumeric**	**Numeric Integer** **Numeric Noninteger** **Numeric Edited**
ALPHABETIC		Yes/4c	Yes/4a	No/3a
ALPHANUMERIC		Yes/4c	Yes/4a	Yes/4b
ALPHANUMERIC EDITED		Yes/4c	Yes/4a	No/3a
NUMERIC	INTEGER	No/3b	Yes/4a	Yes/4b
	NONINTEGER	No/3b	No/3c	Yes/4b
NUMERIC EDITED		No/3b	Yes/4a	No/3a

VERB FORMAT: MOVE

Valid Move Combinations

Figure 7-1 shows the valid combinations of sending and receiving fields permitted in COBOL74.

SENDING \ RECEIVING		ALPHABETIC	AN		AE	DISPLAY NUMERIC		CMP NUMERIC		NE
			GROUP	ELEM		INTEGER	REAL	INTEGER	REAL	
ALPHABETIC		①	①	①	⑥	*	*	*	*	*
AN	GROUP	①	①	①	①	①	①	③	③	①
	ELEM (a)	①	①	①	⑥	②	②	②	②	⑦
AE		①	①	①	⑥	*	*	*	*	*
DISPLAY NUMERIC (DN OR LIT)	INTEGER	*	①	①	⑥	②	②	②	②	⑦
	REAL	*	①	*	*	②	②	②	②	⑦
CMP NUMERIC	INTEGER (b)	*	⑤	⑤	④	②	②	②	②	⑦
	REAL	*	⑤	*	*	②	②	②	②	⑦
NE		*	①	①	⑥	*	*	*	*	*

a – also non-numeric literal

b – also undigit literal

	NON-NUMERIC MOVE	NUMERIC MOVE	LEFT JUST.	JUST. BY DECIMAL	SPACE FILL	ZERO FILL ON RIGHT	ZERO FILL ON LEFT	ANY NECESSARY TRANSLATION	SENDING ZONES STRIPPED	PROPER ZONES STRIPPED OR SUPPLIED BY INTERPRETER	EDITING PERFORMED
①	√		√		√			√			
②		√		√			√			√	
③	√		√			√			√		
④	√		√		√			√		√	√
⑤	√		√		√			√		√	
⑥	√		√		√			√			√
⑦		√		√			√				√

* ILLEGAL

G12337

Figure 7-1. Valid MOVE Statement Combinations

MULTIPLY

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

General Format:

Format 1:

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

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

Format 2:

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

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

Syntax Rules:

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The composite of operands, determined by superimposing all receiving data items of a given statement, aligned on decimal points, must not exceed 18 digits.

General Rules:

1. Refer to ROUNDED Phrase, SIZE ERROR Phrase, Arithmetic Statements, Overlapping Operands, and Multiple Results in Arithmetic Statements in this section for additional rules and information.

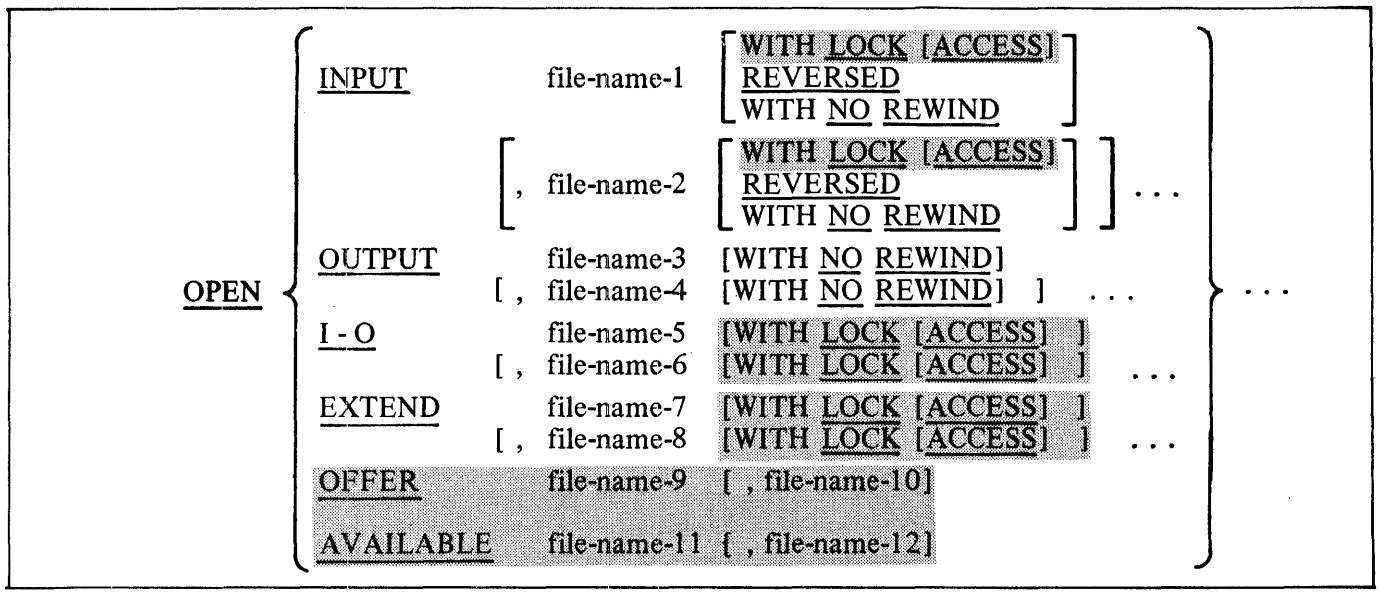
2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; similarly for identifier-1 or literal-1 and identifier-3, and so forth.

3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, and so on.

OPEN

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

General Format:



Syntax Rules:

1. The REVERSED and the NO REWIND phrases can only be used with sequential single reel/unit files. Refer to CLOSE in this section for additional information.
2. The I-O phrase can be used only for mass storage files.
3. The EXTEND phrase can be used only for sequential files.
4. The EXTEND phrase must not be specified for multiple file reels. Refer to the I-O-CONTROL Paragraph in section 5.
5. The files referenced in the OPEN statement need not all have the same organization or access.
6. File-name-9, file-name-10, file-name-11, and file-name-12 must be port files. File-name-5 and file-name-6 can be port files. File-name-1, file-name-2, file-name-3, file-name-4, file-name-7, and file-name-8 must not be port files.

VERB FORMAT: OPEN

General Rules:

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The execution of an OPEN statement does not affect either the contents or availability of the file's record area.

3. When a given file is not in an open mode, no statement (except for a SORT or MERGE statement with the USING or GIVING phrases) that references that file, either explicitly or implicitly, can be executed.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In table 7-5, I, R, or S at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the corresponding file organization and open mode given at the top of the column.

Table 7-5. Permissible Statements

File Access Mode	Statement	Open Mode			
		Input	Output	Input-Output	Extend
Sequential	READ	I R S		I R S	
	WRITE		I R S		S
	REWRITE			I R S	
	START	I R		I R	
	DELETE			I R	
Random	READ	I R S		I R S	
	WRITE		I R S	I R S	
	REWRITE			I R S	
	START				
	DELETE			I R	
Dynamic	READ	I R		I R	
	WRITE		I R	I R	
	REWRITE			I R	
	START	I R		I R	
	DELETE			I R	

The letters I, R, and S have the following meanings:

I = Indexed
R = Relative
S = Sequential

5. A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the REEL, UNIT, or LOCK phrase, for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. If label records are specified for the file, the beginning labels are processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with conventions for input label checking.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with conventions for output label writing.

      If label records are not specified, it is possible by operator intervention to equate the file to one with labels, in which case the label records are ignored.

8. The file description entry for file-name-1, file-name-2, file-name-5, file-name-6, file-name-7, or file-name-8 must be equivalent to that used when this file was created.

9. If an input file is designated with the OPTIONAL phrase in the SELECT clause, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the AT END condition to be executed only when the operator has responded with an "OF" system message. Refer to the READ Statement in this section.

10. The REVERSED and NO REWIND phrases can only be used with sequential single reel/unit files. Refer to CLOSE in this section.

11. The REVERSED and WITH NO REWIND phrases are ignored if they do not apply to the storage media on which the file resides.

12. If the storage media for the file permits rewinding, the following rules apply:

    a. When neither the REVERSED, the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at the beginning.

    b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at the beginning prior to execution of the OPEN statement.

    c. When the REVERSED phrase is specified, the file is positioned at the end by execution of the OPEN statement.

13. When the REVERSED phrase is specified, the subsequent READ statements for the file make the data records of the file available in reversed order starting with the last record.

VERB FORMAT: OPEN

14. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the prime record key is established as the key of reference and is used to determine the first record to be accessed. If no records exist in the file, the current record pointer is set such that the next executed Format 1 READ statement for the file results in an AT END condition.

15. When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file had been opened with the OUTPUT phrase.

16. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The beginning file labels are processed only in the case of a single reel unit file.

    b. The beginning reel/unit labels on the last existing reel/unit are processed as though the file was being opened with the INPUT phrase.

    c. The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.

    d. Processing then proceeds as though the file had been opened with the OUTPUT phrase.

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

18. When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The labels are checked in accordance with conventions for input-output label checking.

    b. The new labels are written in accordance with conventions for input-output label writing.

19. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

    A relative or sequential file opened with the OUTPUT phrase specified is entered into the disk directory as a result of an appropriate CLOSE statement.

20. OPEN OUTPUT of an ISAM File enters the file in the disk directory during MCP processing of the OPEN statement. Any existing ISAM file structures with the same external names as the newly created ISAM file will be removed at this time. For further information on ISAM file naming, refer to appendix G in this manual.

21. The LOCK and LOCK ACCESS options only have meaning for mass-storage files and are ignored for all other files.

22. When an OPEN with the LOCK option (without ACCESS) is executed, the following occurs:

    a. If the specified file is already in an open mode, the program is suspended awaiting exclusive availability of the file.

    b. If the file is not currently in an open mode by any program, the file is opened.

23. When an OPEN WITH LOCK ACCESS is executed, the program is suspended if any of the following conditions applies:

    a. The file is currently OPEN WITH LOCK.

    b. The file is currently OPEN I-O or EXTEND.

    c. The file is currently OPEN INPUT LOCK ACCESS and this OPEN request is I-O LOCK ACCESS or EXTEND LOCK ACCESS.

    The program remains suspended until none of these conditions is in effect or until the program is terminated. In any other case, the OPEN WITH LOCK ACCESS causes the file to be opened.

24. When file-name-1 is a queue file, the TITLE of the file is compared with the TITLE of each of the queue files currently available to this program. If such a queue file exists, this queue file is connected to it. If such a queue file does not exist, a new queue file with any associated subfiles is created.

    The characteristics of a queue file are established at the time it is created; that is, the file attributes defined by the first program to open the queue file apply to all other programs that subsequently open the queue file.

25. The logical communication path between two port files is established by the operating system, provided that their connection descriptions match.

    If an ACTUAL KEY clause is specified, its value determines which subfile of the file is to be opened. If the ACTUAL KEY value is zero or if the ACTUAL KEY clause is not specified, the entire port file is opened. If the ACTUAL KEY value is non-zero, only the specified subfile is opened.

26. The OFFER phrase specifies that the subfile can be offered for matching to another process and that control is returned immediately to the next statement without waiting for a match to occur.

27. The AVAILABLE phrase stipulates that a specific subfile is opened if it matches a port file that has already been offered. If a match does not occur, the subfile is not opened and it is no longer considered for subsequent matching. Control is returned immediately to the next statement.

    If the FILE STATUS clause is declared for the subfile, the status key is updated to a value of 00 if control was returned with the file opened, or updated to a value of 81 if control was returned with the subfile unopened.

VERB FORMAT: OPEN

28. The AVAILABLE phrase can be used to test for the presence or absence of a file on disk if the organization of the file is sequential or relative. A subsequent conditional query of the OPEN attribute indicates the presence or absence of the file on disk.

Example:

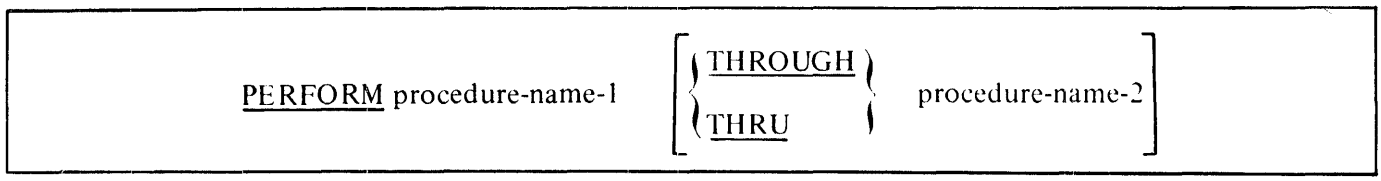
```
OPEN AVAILABLE <file-name>.
IF ATTRIBUTE OPEN OF <file-name> = FALSE
    DISPLAY <file-name> " FILE NOT THERE, PROGRAM GOING TO EOJ"
    GO TO END-OF-JOB
ELSE
    NEXT SENTENCE.
```

PERFORM

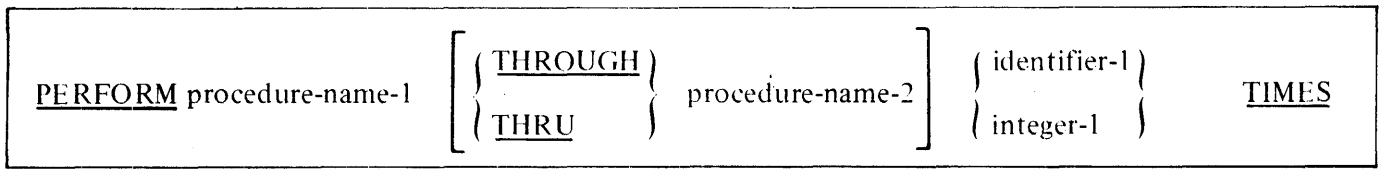
The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

General Format:

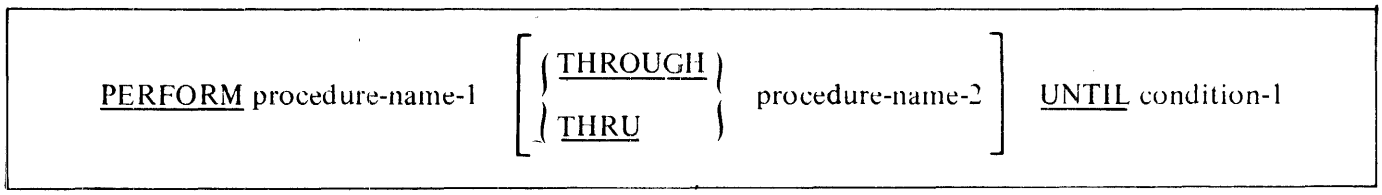
Format 1:



Format 2:

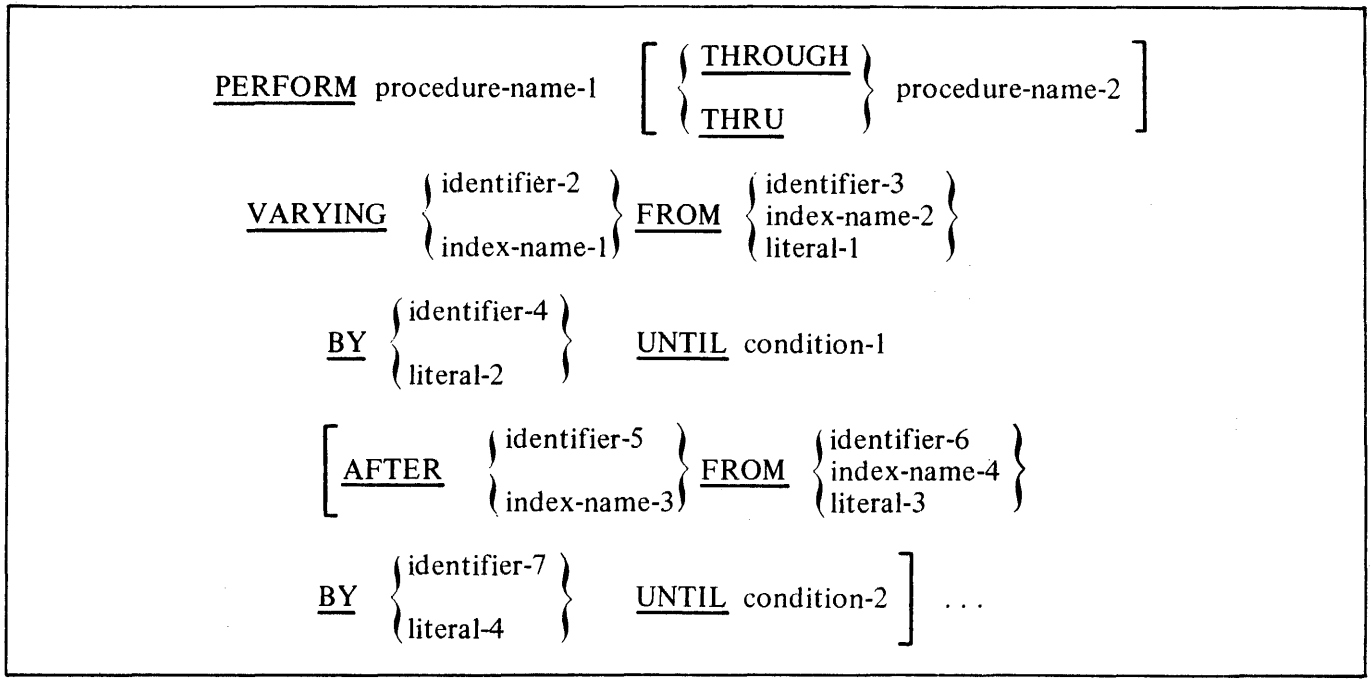


Format 3:



VERB FORMAT: PERFORM

Format 4:



Syntax Rules:

1. Each identifier represents a numeric elementary item described in the DATA DIVISION. In Format 2, identifier-1 must be described as a numeric integer.
2. Each literal represents a numeric literal.
3. The words THRU and THROUGH are equivalent.
4. If an index-name is specified in the VARYING or AFTER phrase, then:
   a. The identifier in the associated FROM and BY phrases must be an integer data item.
   b. The literal in the associated FROM phrase must be a positive integer.
   c. The literal in the associated BY phrase must be a nonzero integer.
5. If an index-name is specifed in the FROM phrase, then:
   a. The identifier in the associated VARYING or AFTER phrase must be an integer data item.
   b. The identifier in the associated BY phrase must be an integer data item.
   c. The literal in the associated BY phrase must be an integer.

6. Literal in the BY phrase must not be zero.

7. Condition-1, condition-2 may be any conditional expression as described in Conditional Expressions in this section.

8. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declarative section of the program, then both must be procedure-names in the same declarative section.

General Rules:

1. The data items referenced by identifier-4 and identifier-7 must not have a zero value.

2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

3. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1, except as indicated in General Rules 6b, 6c, and 6d which follow. This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

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

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

   c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. No particular sequential relationship is required to exist between procedure-name-1 and procedure-name-2. There may be more than one logical path of program control through the performed range of procedures. A common method, though not a required one, of documenting the terminal paragraph of a performed range of procedures is through the use of the EXIT statement.

5. An implicit return mechanism is established at the end of a performed range of procedures and is activated by the execution of a PERFORM statement. Program control reaching an active return mechanism always returns to the activating PERFORM statement. A return mechanism permanently deactivates by transferring program control back to a PERFORM statement. An active return mechanism is temporarily deactivated by the execution of a PERFORM statement. Program control always passes through a nonactive return mechanism to the next executable statement following the PERFORM range.

VERB FORMAT: PERFORM

6. The PERFORM statements operate as follows with General Rule 5 applying to all formats:

   a. Format 1 is the format of the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once and then control passes to the next executable statement following the PERFORM statement.

   b. Format 2 is the format of the PERFORM...TIMES statement. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

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

   c. Format 3 is the format of the PERFORM...UNTIL statement. The specified procedures are performed until the condition specified by the UNTIL phrase is TRUE. When the condition is TRUE, control is transferred to the next executable statement after the PERFORM statement. If the condition is TRUE when the PERFORM statement is executed, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.

   d. Format 4 is the format of the PERFORM...VARYING statement. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names. When index-name appears in a VARYING and/or AFTER phrase, it is initialized and subsequently augmented according to the rules of the SET statement. When index-name appears in the FROM phrase, identifier, when it appears in an associated VARYING or AFTER phrase, is initialized according to the rules of the SET statement. Subsequent augmentation is described in the following paragraphs.

      Identifier-2 and all occurrences, if any, of identifier-5 are set, in the order of occurrence in the PERFORM statement, to the value of the literal or the current value of the identifier in the associated FROM phrase. Condition-1 is then evaluated.

      If any condition, other than the last, is FALSE on any occasion when it is evaluated, the next condition is evaluated immediately.

      If the last condition is FALSE when evaluated, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The last BY value is then added to the associated identifier-2 or identifier-5 and the condition evaluated again.

If condition-1 is TRUE when evaluated, control is transferred to the next executable statement after the PERFORM statement. If condition-1 is TRUE the first time it is evaluated, the remaining conditions are not evaluated and the sequence of procedures, procedure-name-1 through procedure-name-2, is not executed.

If any condition-2 is TRUE when evaluated, these actions take place: first, the associated identifier-5 is set to the value of literal-3 or the current value of identifier-6 in the associated FROM phrase. Second, the identifier of the immediately preceding AFTER or VARYING phrase is incremented by the associated BY value, and the condition specified in that preceding phrase is then evaluated.

Figure 7-2 is a flowchart for the VARYING phrase of a PERFORM statement having one condition.

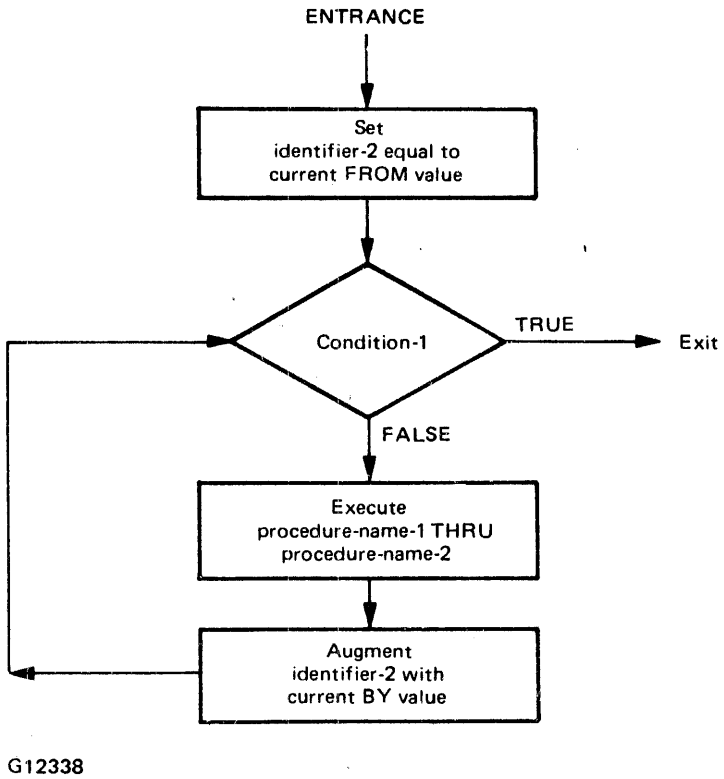


Figure 7-2. PERFORM VARYING with One Condition

VERB FORMAT: PERFORM

Figure 7-3 is a flowchart for the VARYING phrase of a PERFORM statement having two conditions.

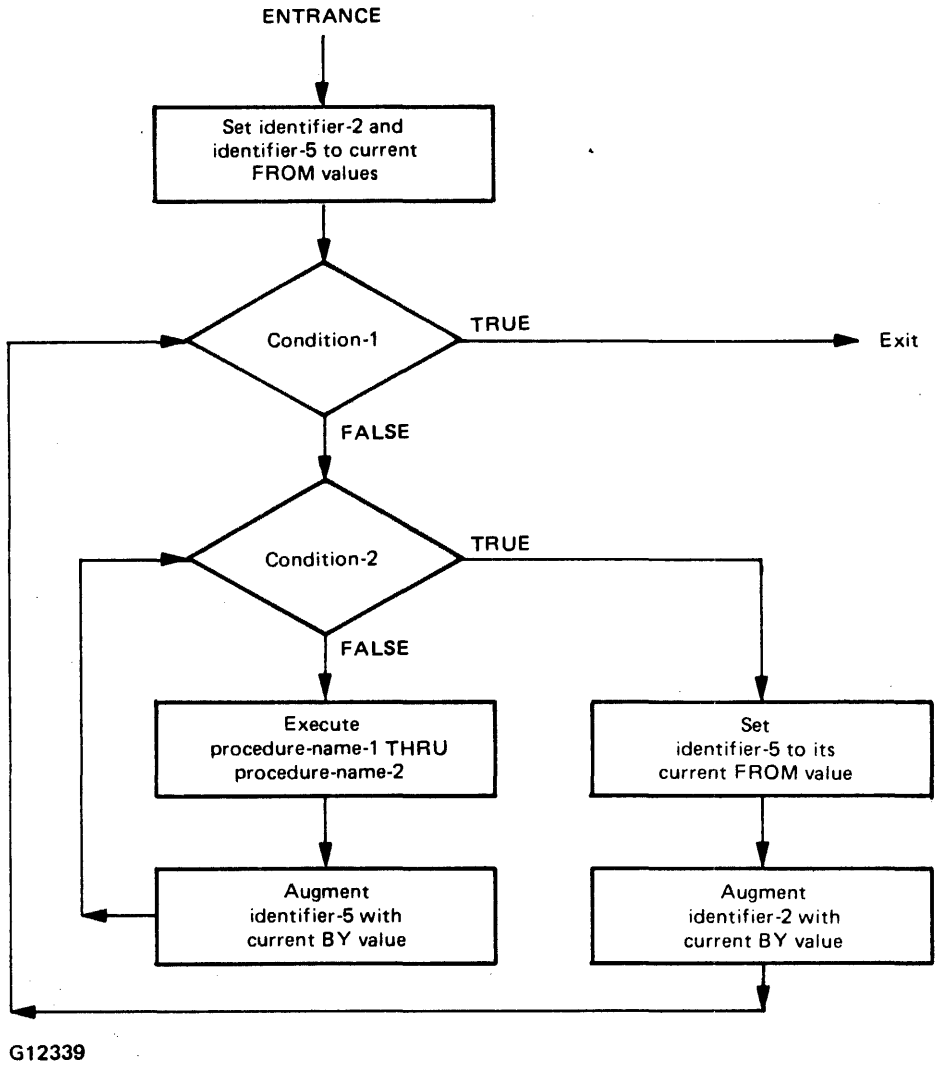


Figure 7-3. PERFORM VARYING with Two Conditions

7. A procedure executed under the control of a PERFORM statement may execute PERFORM statements. There is no requirement that the range of procedures executed under the control of the nested PERFORM statement be declared totally within, or disjoint from, the range of procedures executed by the first PERFORM statement. The permanent deactivation of an active return mechanism causes the last return mechanism temporarily deactivated to again become active, allowing overlapping PERFORM ranges, or two or more PERFORM ranges that have a common exit point, to logically execute the same as disjoint PERFORM ranges.

   Transferring program control, by means of a GO TO statement, from a range of procedures being executed under control of a PERFORM statement does not cause the return mechanism to be deactivated. Subsequently transferring program control back into the PERFORM range causes control to return to the PERFORM statement, provided that the return mechanism is still active. Repeatedly branching from a PERFORM range without allowing control to ever reach an active return mechanism may cause the program to terminate abnormally by exhausting the resources allocated to account for return mechanisms.

READ

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

General Format:

Format 1:

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

Format 2:

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

Format 3:

```
READ file-name RECORD [WITH NO WAIT] [INTO identifier]
    [; AT END imperative-statement]
```

Format 4:

```
READ file-name RECORD [WITH NO WAIT] [INTO identifier]
    [; INVALID KEY imperative-statement]
```

VERB FORMAT: READ

Syntax Rules:

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

2. For an Indexed File, data-name must be the name of a data item specified as a record key associated with file-name.

3. Data-name may be qualified.

4. Format 1 must be used for all files in sequential access mode.

5. The NEXT phrase must be specified for Relative and Indexed Files in dynamic access mode when records are to be retrieved sequentially.

6. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

7. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for the file-name.

8. Format 3 and Format 4 may be used for port and queue files only.

General Rules:

1. The associated file must be open in the INPUT or I-O mode and must not be a SORT or MERGE file. Refer to the OPEN statement in this section.

2. The record to be made available by a Format 1 READ statement for other than port or queue files is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer is positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer. If the record is no longer accessible, which may be caused by the deletion of the record or a change in an alternate record key, the current record pointer is updated to point to the next existing record within the established key of reference and that record is then made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference, and that record is then made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. Refer to I-O Status in section 5.

4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement for other than port or queue files is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.

VERB FORMAT: READ

5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The sending area is considered to be a group item whose size is equal to the maximum record size for this file.

   The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a Format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. Refer to I-O Status in section 5.
10. When the AT END condition is recognized, the following actions are taken in the specified order:

    a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. Refer to I-O Status in section 5.

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.

    c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly for this file, and that procedure is executed.

       When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

11. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For Indexed Files, the key of reference is also undefined.

12. When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

    a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for the file.

c. A successful Format 2 READ statement for that file.

d. A SEEK statement.

13. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in the preceding General Rule 2.

14. For an Indexed File being sequentially accessed, records having the same duplicate value in an alternate record key, which is the key of reference, are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

15. For an Indexed File, if the KEY phrase is specified in a Format 2 READ statement, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements until a different key of reference is established for the file.

16. For an Indexed File, if the KEY phrase is not specified in a Format 2 READ statement, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements until a different key of reference is established for the file.

17. Execution of a Format 2 READ statement on an Indexed File causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. The current record pointer is positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Refer to the INVALID KEY Condition in section 5.

18. If the end of a reel or unit is recognized during execution of a READ statement, and the logical end of the file has not been reached, the following operations are executed:

    a. The standard ending reel/unit label procedure.

    b. A reel/unit swap.

    c. The standard beginning reel/unit label procedure.

    d. The first data record of the new reel/unit is made available.

19. If a Sequential File described with the OPTIONAL phrase is not present at the time the file is opened, then at the time of execution of the first READ statement, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end-of-file procedures are not performed. Refer to the FILE-CONTROL Paragraph, and I-O Status in section 5; OPEN Statement, and USE Statement in section 7. Execution of the program then proceeds as specified in general rule 10.

VERB FORMAT: READ

20. In a Sequential File, if the ACTUAL KEY phrase is specified for a file whose access mode is sequential, the execution of a Format 1 READ statement updates the contents of the ACTUAL KEY data item to the ordinal number of the logical record accessed.

21. The execution of a Format 2 READ statement accesses the record specified by the contents of the ACTUAL KEY data item.

22. If, upon the execution of a Format 2 READ statement, the contents of the ACTUAL KEY data item are less than 1 or greater than the ordinal number of the last record written to the file, the READ statement is unsuccessful and the INVALID KEY condition exists.

23. In a Relative File, if the RELATIVE KEY phrase is specified, the execution of a Format 1 READ statement updates the content of the RELATIVE KEY data item so that it contains the relative record number of the record made available.

24. For a Relative File, the execution of a Format 2 READ statement sets the current record pointer and makes available the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. Refer to the INVALID KEY condition under Invalid Key in section 5.

25. For a queue file, the message made available by a READ statement is determined as follows:

    a. If the file is not connected to a queue file family, the oldest message in the queue file is made available.

    b. If the file is connected to a queue file family, the contents of the ACTUAL KEY data item associated with the file specify the index of the subfile that is to be read. The oldest message in that subfile is made available.

       If no ACTUAL KEY data item is specified or if the contents of the ACTUAL KEY data item are zero, a message from one of the subfiles is made available. The members of the queue file family are examined beginning with queue number one, and the first queue member found not empty is read.

26. When the WITH NO WAIT option for queue files is used, the program senses a Q-Empty condition (no messages in the queue to read), and continues with the next executable instruction. If the WITH NO WAIT option is not specified and a Q-Empty condition occurs, the program is suspended until the read operation is completed.

27. Use of the AT END clause for a queue file notifies the reading program that the writing program has terminated. It could serve as a termination condition, possibly for an abnormal end, for the reading program.

28. For port files, a READ statement causes the program to wait until a logical record is available. If the WITH NO WAIT phrase is specified, the program does not wait. A status key value of 94 indicates that no logical record was available for the read operation.

29. If an ACTUAL KEY clause is declared for a port file, the user is responsible for updating the ACTUAL KEY data item with an appropriate subfile index. If the ACTUAL KEY value is non-zero, a read operation from the specified subfile is performed. If the ACTUAL KEY value is zero, a "non-selective" read operation is performed and the ACTUAL KEY data item is updated with the subfile index of the subfile that was read.

    For a non-selective read operation, the first logical record to arrive at a subfile in the port file is returned as the data for the READ statement. The subfile to be read is determined by the operating system and no specific selection algorithm is guaranteed. However, a selection method which is "fair" to all subfiles containing a logical record is provided. That is, one subfile is not read continuously at the expense of the other subfiles.

    If no ACTUAL KEY clause is declared for the port file, the file must contain a single subfile, and that subfile is read.

RECEIVE

The RECEIVE statement makes available to the COBOL74 program a message, message segment, or a portion of a message or segment, and other pertinent information about that data from a queue maintained by the Data Communication Subsystem. The RECEIVE statement allows for a specific imperative statement when no data is available.

Message segmentation (SEGMENT option) requires a participating Data Communication Subsystem that supports this feature. Refer to Communication Section in section 6 of this manual.

General Format:

```
RECEIVE  cd-name { MESSAGE } INTO  identifier-1
                 { SEGMENT }
                 [ ; NO DATA imperative-statement ]
```

Syntax Rules:

1. Cd-name must reference an input CD.

General Rules:

1. The contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE3) of the area referenced by cd-name designate the queue structure containing the message. Refer to the Communication Description Structure (CD) in section 6.

2. The message, message segment, or portion of a message or segment is transferred to the receiving character positions of the area referenced by identifier-1 aligned to the left without space fill.

3. When, during the execution of a RECEIVE statement, the data is made available in the data item referenced by identifier-1, control is transferred to the next executable statement, whether or not the NO DATA phrase is specified.

4. When, during the execution of a RECEIVE statement, the data is not made available in the data item referenced by identifier-1:

   a. If the NO DATA phrase is specified, the RECEIVE operation is terminated with the indication that action is complete (refer to General Rule 5), and the imperative statement in the NO DATA phrase is executed.

   b. If the NO DATA phrase is not specified, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

   c. If the queue is unknown or access is denied to the queue, control passes to the next executable statement, whether or not the NO DATA phrase is specified. Refer to table 6-3, Communication Status Key Condition, in section 6.

5. The data items identified by the input CD are appropriately updated at each execution of a RECEIVE statement. Refer to the Communication Description Structure (CD) in section 6.

6. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the Data Communication Subsystem does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.

7. When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:

   a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

   b. If a message size is less than the size of the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

   c. If a message size is greater than the size of the area referenced by identifier-1, the message fills the area referenced by identifier-1 left to right, starting with the leftmost character of the message. The remainder of the message is transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, sub-queue, etc. Note that the remainder of the message is treated as a new message and the rules 7a, 7b, and 7c are applied again.

RELEASE

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

General Format:

```
RELEASE record-name [FROM identifier]
```

Syntax Rules:

1. A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose Sort-Merge file description entry contains record-name. Refer to SORT in this section.

2. Record-name must be the name of a logical record in the associated Sort-Merge file description entry and may be qualified.

3. Record-name and identifier must not refer to the same storage area.

General Rules:

1. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.

2. If the FROM phrase is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase.

3. The execution of a RELEASE statement has no effect upon either the contents or accessibility of the record area. If the sort-merge file is named in a SAME RECORD AREA clause, the logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated sort-merge file, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of RELEASE statements.

VERB FORMAT: RETURN

RETURN

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

General Format:

```
RETURN file-name RECORD [INTO identifier]  ; AT END imperative-statement
```

Syntax Rules:

1. File-name must be described by a Sort-Merge file description entry in the DATA DIVISION.

2. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name.

3. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

General Rules:

1. When the logical records of a file are described with more than one record description, these records automatically share the same storage area, which is an implicit redefinition of the area.

2. The execution of the RETURN statement causes the next record, in the order specified by the KEY clause in the SORT or MERGE statement, to be transferred to the record area.

It is the responsibility of the COBOL74 programmer to account, if necessary, for the size of the logical record returned.

3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase, with the sending area considered to be a group item having a fixed size equal to the maximum record size. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

4. When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

5. If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs.

   When the AT END condition occurs, no transfer of data to the record area takes place and the contents of the record area are undisturbed. After the AT END condition occurs, the contents of the record area are still accessible. After the execution of the imperative-statement in the AT END phrase, no other RETURN statement may be executed as part of the current output procedure.

REWRITE

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format:

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

Syntax Rules:

1. Record-name and identifier must not refer to the same storage area.
2. Record-name is the name of a logical record in the FILE SECTION of the DATA DIVISION and may be qualified.
3. The INVALID KEY phrase must not be specified for a REWRITE statement which references a file in sequential access mode.
4. The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified.

General Rules:

1. The file associated with record-name must be a mass storage file and must be open in the I-O mode at the time of execution of this statement. It cannot be a SORT or MERGE file. Refer to the OPEN statement in this section.
2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must be a successfully executed READ statement. The record that is accessed by the READ statement is logically replaced.
3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.
4. The execution of the REWRITE statement has no effect upon the contents or accessibility of the record area. If the associated file is named in a SAME RECORD AREA clause, the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area before execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. Refer to I-O STATUS in section 5.

8. For a Relative File accessed in either random or dynamic access mode, the record specified by the contents of the RELATIVE KEY data item associated with the file is the one rewritten. If the file does not contain the record specified by the key, the INVALID KEY condition exists. Refer to the Invalid Key in section 5. The updating operation does not take place and the data in the record area is unaffected.

9. In a Sequential File, contents of the ACTUAL KEY data item, if specified, are ignored during the execution of the REWRITE statement and are not updated by execution of that statement.
10. For an Indexed File in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed, the value contained in the prime record key data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file.

11. For an Indexed File in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

12. In an Indexed File, the contents of alternate record key data items of the record being rewritten may differ from those in the record being replaced. The contents of the record key data items are utilized during the execution of the REWRITE statement in such a way that subsequent access of the record may be made based upon any of those specified record keys.

13. In an Indexed File, the INVALID KEY condition exists when:

    a. The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file.

    b. The value contained in the prime record key data item does not equal that of any record stored in the file, or

    c. The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified is equal to that of a record already stored in the file.

    The updating operation does not take place and the data in the record area unaffected. Refer to the Invalid Key in section 5.

SEARCH

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

General Formats:

Format 1:

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

Format 2:

```
SEARCH ALL identifier-1 [; AT END imperative-statement-1]
    ; WHEN { data-name-1      { IS EQUAL TO } { identifier-3      } }
           { condition-name-1  { IS =        } { literal-1         }
                               {             } { arithmetic-expression-1 } }
    [ AND { date-name-2       { IS EQUAL TO } { identifier-4      } } ]   ...
          { condition-name-2  { IS =        } { literal-2         }
                              {             } { arithmetic-expression-2 } }
    { imperative-statement-2 }
    { NEXT SENTENCE          }
```

NOTE
To avoid confusion with other symbols, the required relational character ' = ' is not underlined.

VERB FORMAT: SEARCH

Syntax Rules:

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. Identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, condition-1, condition-2, and so on, may be any condition as described in Conditional Expressions in this section.

4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indices or literals as required, and must be referenced in the KEY clause of identifier-1. Identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 must not be referenced in the KEY clause of identifier-1 or be indexed by the first index-name associated with identifier-1.

   In Format 2, when a data-name in the KEY clause of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of identifier-1 is referenced, all preceding data-names in the KEY clause of identifier-1 or their associated condition-names must also be referenced.

General Rules:

1. If Format 1 of the SEARCH statement is used, a serial type of search operation takes place, starting with the current index setting.

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the operation is terminated immediately. The number of occurrences of identifier-1, the last of which is the highest permitted, is discussed in the OCCURS clause. Refer to the OCCURS clause in section 6. Then, if the AT END phrase is specified, imperative-statement-1 is executed, and the AT END phrase is not specified, control passes to the next executable sentence.

   b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1, the SEARCH statement operates by evaluating the conditions in the order in which written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are 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 outside the permissible range of occurrence values, in which case the search terminates as indicated in General Rule 1a. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

VERB FORMAT: SEARCH

2. In a Format 2 SEARCH statement, the results of the SEARCH ALL operation are predictable only when the data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of identifier-1.

3. If Format 2 of the SEARCH statement is used, a binary search type of operation takes place; the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation according to accepted binary techniques, with the restriction that at no time is it set to a value exceeding the value which corresponds to the last element of the table, or which is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause. Refer to the OCCURS clause in section 6. If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence when not specified. In either case, the final setting of the index is the value which corresponds to an occurrence number which is one greater than the last element of the table.

   If all the conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2.

   When the contents of the key(s) referenced in the WHEN clause are not sufficient to identify a unique table element, the index indicates the occurrence closest to the beginning of the table which satisfies the condition.

4. After execution of imperative-statement-1, imperative-statement-2, or imperative-statement-3, that does not terminate with a GO TO statement, control passes to the next executable sentence.

5. In Format 2, the index-name that is used for the search operation is the first or only index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

6. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first or only index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

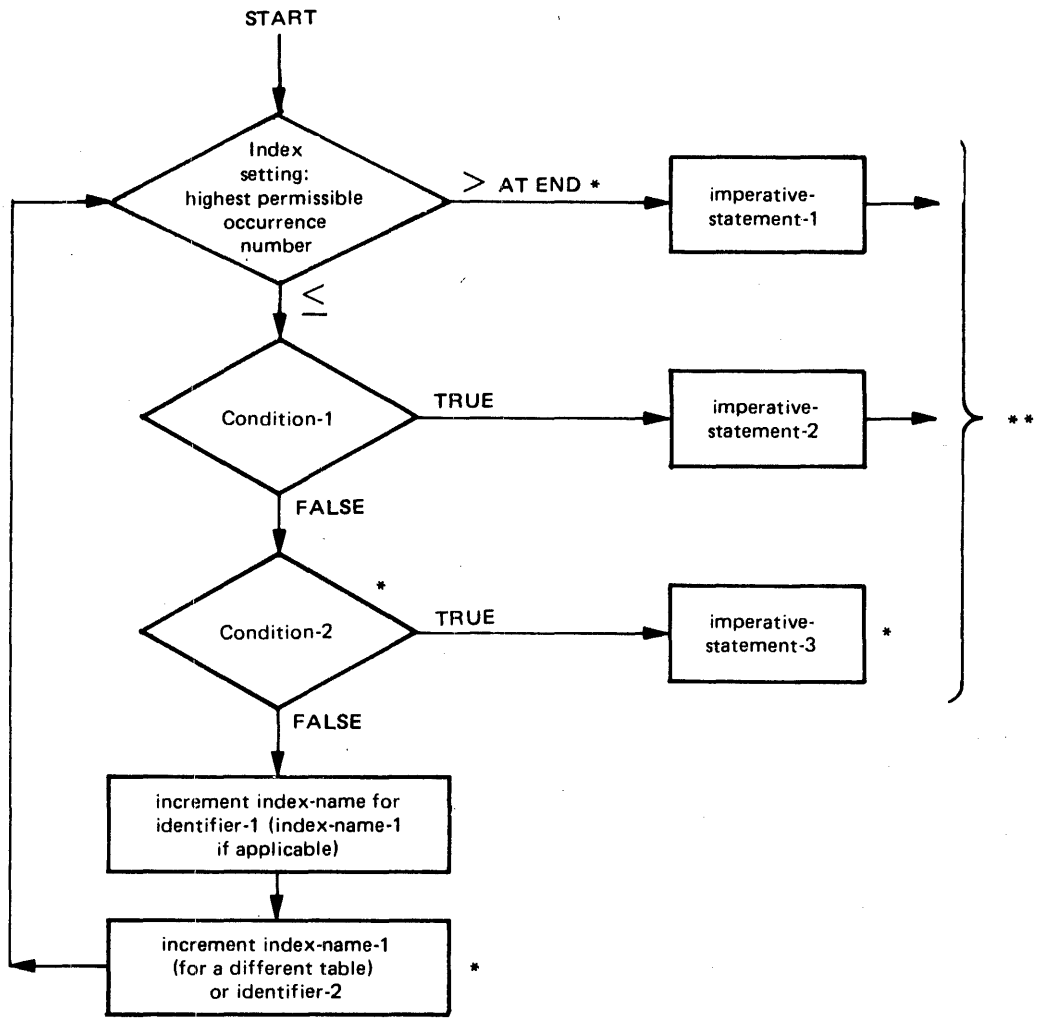
7. In Format 1, if the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of identifier-1, that index-name is used for this search. If this is not the case, or if the VARYING identifier-2 phrase is specified, the first or only index-name given in the INDEXED BY phrase of identifier-1 is used for the search. In addition, the following operations occur:

   a. If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY phrase 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 is incremented.

   b. If the VARYING identifier-2 phrase is specified, and identifier-2 is an index data item, then the data item referenced by identifier-2 is incremented by the same amount and at the same time as the index associated with identifier-1 is incremented. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value 1 at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

8. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause providing for a multidimensional table, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire multidimensional table it is necessary to execute a SEARCH statement several times. Before each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

Figure 7-4 contains a flowchart of the Format 1 SEARCH operation containing two WHEN phrases.



* THESE OPERATIONS ARE OPTIONS INCLUDED ONLY WHEN SPECIFIED IN THE SEARCH STATEMENT.

** EACH OF THESE CONTROL TRANSFERS IS TO THE NEXT EXECUTABLE SENTENCE UNLESS THE IMPERATIVE-STATEMENT ENDS WITH A GO TO STATEMENT.

G12340

Figure 7-4. SEARCH with Two WHEN Phrases

SEEK

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

General Format:

```
SEEK file-name RECORD
```

Syntax Rules:

1. File-name must be the name of a mass storage file.
2. The ACTUAL KEY clause must be specified in the FILE-CONTROL entry for file-name.

General Rules:

1. The SEEK statement may serve to prelocate the address of a record of a mass storage file so that overlap of I/O and Processor operations may be facilitated.
2. The value of the ACTUAL KEY specified for file-name is used as the ordinal number of the record sought.
3. For files specified for SEQUENTIAL access, the SEEK statement serves to reposition the file, such that if the next statement which references the file is a READ statement or a WRITE statement, the record accessed will be that specified by the contents of the ACTUAL KEY data item at the time of execution of the SEEK statement.
4. The execution of a SEEK statement does not cause the contents of the Status Key data item to be updated.

SEND

The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more output queues maintained by the Data Communication Subsystem. Use of the partial message feature (ESI and EGI) requires a participating Data Communication Subsystem that supports this construct. Valid syntax for ESI and EGI is ignored by non-participating, non-supporting Data Communication Subsystems. Refer to the Communication Section in section 6 of this manual and the B 1000 Systems Network Definition Language (NDL) Reference Manual for further information.

General Formats:

Format 1:

```
SEND    cd-name    [ FROM  identifier-1 ]
```

Format 2:

```
SEND cd-name   [FROM identifier-1]  { WITH identifier-2 }
                                    { WITH ESI         }
                                    { WITH EMI         }
                                    { WITH EGI         }

[ { BEFORE }                                                                  ]
[ { AFTER  } ADVANCING { { { identifier-3 } [LINE ] } }                       ]
[                      { { { integer      } [LINES] } }                       ]
[                      { { mnemonic-name }                                    ]
[                      { { PAGE          }                                    ]
```

Syntax Rules:

1. Cd-name must reference an output CD.
2. Identifier-2 must reference a 1-character integer without an operational sign.
3. When identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary integer item.
4. When the mnemonic-name phrase is used, it must be associated with a CHANNEL number. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.
5. Integer or the value of the data item referenced by identifier-3 may be zero.

VERB FORMAT: SEND

General Rules:

All formats:

1. When a receiving communication device such as a printer, display screen, or card punch, is oriented to a fixed line size:

   a. Each message or message segment begins at the leftmost character position of the physical line.

   b. A message or message segment that is smaller than the physical line size is released so as to appear space-filled to the right.

   c. Excess characters of a message or message segment are not truncated. Characters are packed to a size equal to that of the physical line and then sent to the device. The process continues on the next line with the excess characters.

2. When a receiving communication device (paper tape punch, or another computer) is oriented to handle variable length messages, each message or message segment begins on the next available character position of the communications device.

3. As part of the execution of a SEND statement, the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name is the user's indication of the number of leftmost character positions of the data item referenced by identifier-1 from which data is to be transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. Refer to table 6-3, Communication Status Key Condition, in section 6.

4. As part of the execution of a SEND statement, the contents of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name are updated. Refer to Communication Description Structure (CD) in section 6.

5. The effect of having special control characters within the contents of the data item referenced by identifier-1 are undefined.

6. A single execution of a SEND statement for Format 1 releases only a single portion of a message or of a message segment to the Data Communication Subsystem.

   A single execution of a SEND statement for Format 2 never releases to the Data Communication Subsystem more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI, EMI, or EGI.

The length of the portion of the message or message segment is defined by rule 3 above.

The Data Communication Subsystem may not transmit any portion of a message to a communication device until the entire message has been placed in the output queue.

7. Disposition of a portion of a message not terminated by an EMI or EGI is defined within the Data Communication Subsystem.

Format 2:

8. The contents of the data item referenced by identifier-2 indicates that the contents of the data item referenced by identifier-1 are to have associated with them an ending indicator. Table 7-6 lists the possible values for the ending indicator and the meaning associated with that value.

   A supporting Data Communication Subsystem recognizes these indicators and establishes whatever is necessary to maintain group, message, and segment control as defined in the following rules:

   a. If the contents of the data item referenced by identifier-2 contains a "1", the contents of the data item referenced by identifier-1 is associated with an end of segment indicator or ESI.

      The ESI indicates to the Data Communication Sybsystem that the message segment is complete.

   b. If the contents of the data item referenced by identifier-2 contains a "2", the contents of the data item referenced by identifier-1 is associated with an end of message indicator or EMI.

      The EMI indicates to the Data Communication Sybsystem that the message is complete.

   c. If the contents of the data item referenced by identifier-2 contains a "3", the contents of the data item referenced by identifier-1 is associated with an end of group indicator or EGI.

      The EGI indicates to the Data Communication Sybsystem that the group of messages is complete.

      The interpretation given to the EGI is dependent upon options as specified in the Data Communication Subsystem.

   d. The hierarchy of ending indicators is EGI, EMI, and ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.
      If the content of the data item referenced by identifier-2 is other than a "1", "2", or "3", and identifier-1 is not specified, then an error is indicated by the value in the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred.

VERB FORMAT: SEND

9. The ADVANCING phrase allows control of the vertical positioning of each message on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the vertical positioning specified or implied is ignored.

10. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing is provided to act as if the user has specified AFTER ADVANCING 1 LINE.

11. If the ADVANCING phrase is implicitly or explicitly specified and vertical positioning is applicable, the following rules apply:

    a. If identifier-3 or integer is specified, characters transmitted to the communication device are repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.

    b. If mnemonic-name is specified, characters transmitted to the communication device are positioned to the line number corresponding to the CHANNEL number.

    c. If the BEFORE phrase is used, the message is represented on the communication device before vertical repositioning according to General Rules 9a and 9b.

    d. If the AFTER phrase is used, the message is represented on the communication device after vertical repositioning according to General Rules 9a and 9b.

    e. If PAGE is specified, characters transmitted to the communication device are represented on the device BEFORE or AFTER (depending upon the phrase used) the device is repositioned to the next page. If PAGE is specified but page has no meaning in conjunction with a specific device, then page advancing is provided to act as if the user has specified BEFORE or AFTER ADVANCING 1 LINE.

Table 7-6. Specifying End Indicators

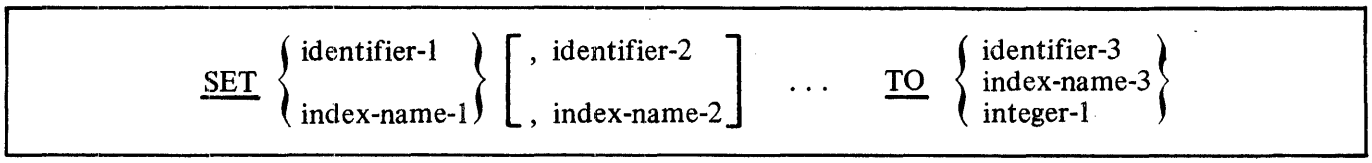
	Means of Specifying	
Type of Indicator	**identifier-1**	**identifier-2**
no indicator	blank	0
end of segment	ESI	1
end of message	EMI	2
end of group	EGI	3

SET

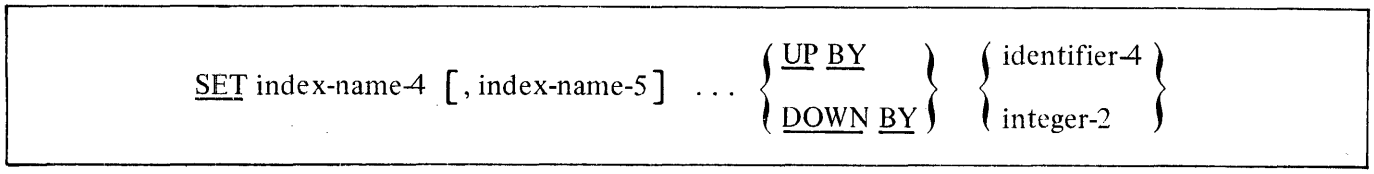
The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

General Format:

Format 1:



Format 2:



Syntax Rules:

1. 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.

2. Identifier-1 and identifier-3 must name index data items, or elementary items described as an integer.

3. Identifier-4 must be described as an elementary numeric integer.

4. Integer-1 and integer-2 may be signed. Integer-1 may be zero.

General Rules:

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement should correspond to an occurrence number of an element in the associated table.

   If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement should correspond to an occurrence number of an element in the associated table. If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement should correspond to an occurrence number of an element in the associated table.

VERB FORMAT: SET

It is permissible to set index-name-1, index-name-2, index-name-4, index-name-5, to any value with the following restrictions:

a. If overflow occurs, the value in the index-name is left-truncated according to the arithmetic rules for a size error condition without a SIZE ERROR phrase. Refer to the SIZE ERROR phrase in this section.

b. When a statement using the index-name to refer to a table element is executed, the value in the index or the value produced by relative indexing must fall within the range specified by the OCCURS clause defining the table. Otherwise, an abnormal termination of the program occurs. Refer to Indexing in section 2.

3. When a Format 1 SET statement is executed, the following actions occur:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. 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.

   b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. No conversion takes place in either case.

   c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

   d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement, any subscripting or indexing associated with identifier-1, and so forth, is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4; thereafter, the process is repeated for index-name-5, and so on. Each time, the value of identifier-4 is used as it was at the beginning of the execution of the statement.

5. Data in table 7-7 represents the validity of various operand combinations in the SET statement. The general rule reference (for example, 3b) indicates the applicable general rule.

Table 7-7. SET Statement Combinations

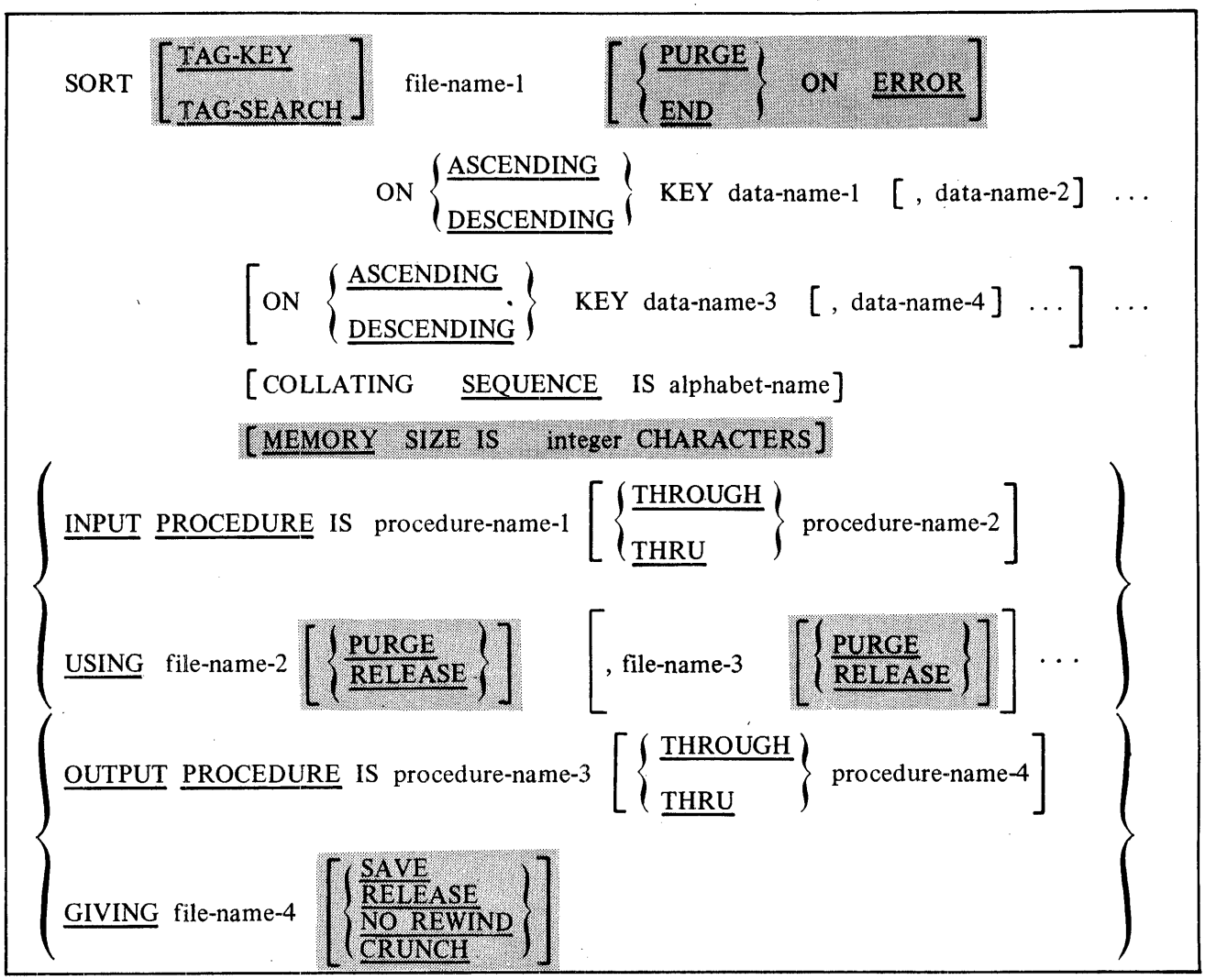
Sending Item	Receiving Item		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	NO/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b*
Index Data Item	No/3c	Valid/3a*	Valid/3b*

*No conversion takes place

SORT

The SORT statement creates a sort file two ways: first, by executing input procedures or, second, by transferring records from another file. The records are then sorted on a set of specified keys, and in the final phase of the sort operation, made available in sorted order, to some output procedures or to an output file.

General Format:



VERB FORMAT: SORT

Syntax Rules:

1. File-name-1 must be described in a sort-merge file description (SD) entry in the DATA DIVISION.

2. Procedure-name-1 represents the name of an input procedure. Procedure-name-3 represents the name of an output procedure.

3. File-name-2, file-name-3, and file-name-4 must be described in a file description entry (FD), not in a Sort-Merge file description entry (SD), in the DATA DIVISION. The actual size of the logical record(s) described for file-name-2 and file-name-4 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records so as to cause an equal number of character positions to be allocated for the corresponding records.

4. Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable length items.

   d. If file-name-1 has more than one record description, then the data items identified by KEY data-names may all be described within one of the record descriptions or in any combination of record descriptions. It is not necessary to redescribe the KEY data-names in each record description.

   e. None of the data items identified by KEY data-names can be described by an entry which contains an OCCURS clause or which is subordinate to an entry which contains an OCCURS clause.

5. The words THRU and THROUGH are equivalent.

6. SORT statements may appear anywhere except in the declaratives portion of the PROCEDURE DIVISION or in an input or output procedure associated with a SORT or MERGE statement.

7. No more than one file-name from a multiple file reel can appear in the SORT statement.

General Rules:

1. The PROCEDURE DIVISION may contain more than one SORT statement appearing anywhere except:

   a. In the declaratives portion.

   b. In the input and output procedures associated with a SORT or MERGE statement.

2. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key.

   a. When the ASCENDING phrase is specified, the sorted sequence is from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

   b. When the DESCENDING phrase is specified, the sorted sequence is from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, by the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

   b. Second, by the collating sequence established as the program collating sequence.

4. The input procedure must consist of one or more paragraphs or sections which appear contiguously in a source program and which do not form a part of any output procedure. In order to transfer records to the file referenced by file-name-1, the input procedure must include the execution of at least one RELEASE statement. Control must not be passed to the input procedure except when a related SORT statement is being executed. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

   a. The input procedure must not contain any SORT or MERGE statements.

   b. The input procedure must not contain any explicit transfers of control to points outside the input procedure. ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL74 statements that cause an implied transfer of control to declaratives are allowed.

   c. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the input procedure. ALTER, GO TO, and PERFORM statements in the remainder of the PROCEDURE DIVISION must not refer to procedure-names within the input procedure.

   Violations of these restrictions on procedural statements concerning input procedures are not syntaxed by the compiler. It is the responsibility of the programmer to ensure that these restrictions are enforced.

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

VERB FORMAT: SORT

6. The output procedure must consist of one or more paragraphs or sections which appear contiguously in a source program and which do not form part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned, one at a time in sorted order, from the sort file. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any SORT or MERGE statements.

   b. The output procedure must not contain any explicit transfers of control to points outside the output procedure. ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL74 statements that cause an implied transfer of control to declaratives are allowed.

   c. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the output procedure. ALTER, GO TO, and PERFORM statements in the remainder of the PROCEDURE DIVISION must not refer to procedure-names within the output procedure.

   Violations of these restrictions on procedural statements concerning output procedures are not syntaxed by the compiler. It is the responsibility of the programmer to ensure that these restrictions are enforced.

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

8. If the USING phrase is specified, all the records in file-name-2 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2. These implicit functions are performed such that any associated USE procedures are executed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 to the file area for file-name-1 and the release of records to the initial phase of the SORT operation.

9. If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically written on file-name-4 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-4 must not be open. The SORT statement automatically initiates the processing, releases logical records to and terminates the processing of file-name-4. These implicit functions are performed so that any associated USE procedures are executed. The terminating function is performed as for a CLOSE statement, without optional phrases, had been executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the SORT operation and the moving of the records from the file area for file-name-1 to the file area for file-name-4.

10. When the TAG-KEY option is used, sorting is performed on keys rather than on the entire record. The record numbers are placed in the sorted order in the GIVING file, which is restricted to an eight-digit record size. The TAG-KEY option prohibits use of INPUT or OUTPUT procedures.

11. When the TAG-SEARCH option is used, sorting is performed on keys rather than on the entire record. The records are then ordered according to the sorted order of the record numbers.

12. The ON ERROR option is provided to allow programmers limited control over irrecoverable parity errors when INPUT/OUTPUT procedures are not present in a program. PURGE causes all records in a block containing an irrecoverable parity error to be dropped, and processing is continued after an ODT message giving the relative position in the file of the bad block is displayed. END causes a program termination and is assumed if no option is specified.

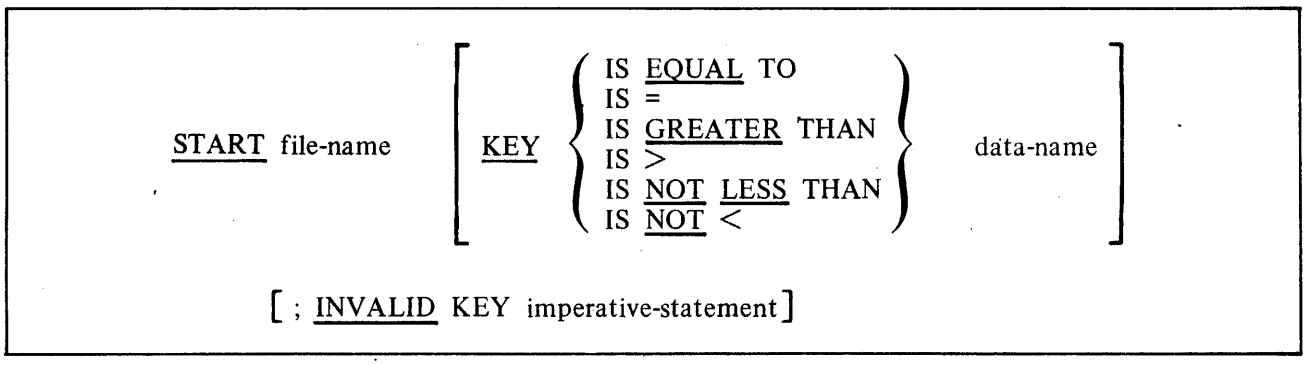
13. MEMORY SIZE is used as a guideline for allocation of the SORT memory area.

14. The SAVE, PURGE, and RELEASE options specify the type of file close for the USING files, file-name-2, and file-name-3. The SAVE, RELEASE, NO REWIND, and CRUNCH options specify the type of file close for the GIVING file, file-name-4. Refer to the CLOSE statement in this section.

START

The START statement provides a basis for logical positioning within an indexed or relative file, for subsequent sequential retrieval of records.

General Format:



NOTE
The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '>', meaning greater than or equal to.

Syntax Rules:

1. File-name must be the name of an Indexed or Relative File.
2. File-name must be the name of a file with sequential or dynamic access.
3. Data-name may be qualified.
4. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.
5. If file-name is the name of an Indexed File, and if the KEY phrase is specified, data-name may reference a data item specified as a record key associated with file-name. Data-name may reference any data item of category alphanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.
6. Data-name, if specified for a Relative File, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

General Rules:

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed. Refer to the OPEN statement in this section.
2. If the KEY phrase is not specified, the relational operator 'IS EQUAL TO' is implied.

3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in General Rule 5. If file-name references an indexed file and the operands are of unequal size, comparison proceeds as though the longer operand were truncated on the right, so that its length is equal to that of the shorter operand. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause has no effect on the comparison. Refer to Comparison of Nonnumeric Operands in this section.

   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. Refer to Invalid Key Condition in section$5.

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. Refer to I-O Status in section$5.

5. For an Indexed File using the KEY phrase, the comparison described in General Rule 3 uses the data item referenced by data-name.

6. For an Indexed File not using the KEY phrase, the comparison described in General Rule 3 uses the data item referenced in the RECORD KEY clause associated with file-name.

7. For a Relative File, the comparison described in General Rule 3 uses the data item referenced by the RELATIVE KEY clause associated with file-name.

8. Upon completion of the successful execution of the START statement, a key of reference is established and used in subsequent Format 1 READ statements as follows:

   a. If the KEY phrase is not specified, the prime record key specified for file-name becomes the key of reference.

   b. If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

   c. If the KEY phrase is specified, and data-name is not specified as a record key for file-name, the record key whose leftmost character position corresponds to the leftmost character position of the data item specified by data-name, becomes the key of reference.

   d. Refer to the READ statement in this section.

9. For an Indexed File, if the execution of the START statement is not successful, the key of reference is undefined.

STOP

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

General Format:

```
STOP { RUN     }
     { literal }
```

Syntax Rules:

1. The literal may be numeric, nonnumeric, or any figurative constant except ALL.
2. If the literal is numeric, then it must be an unsigned integer.
3. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

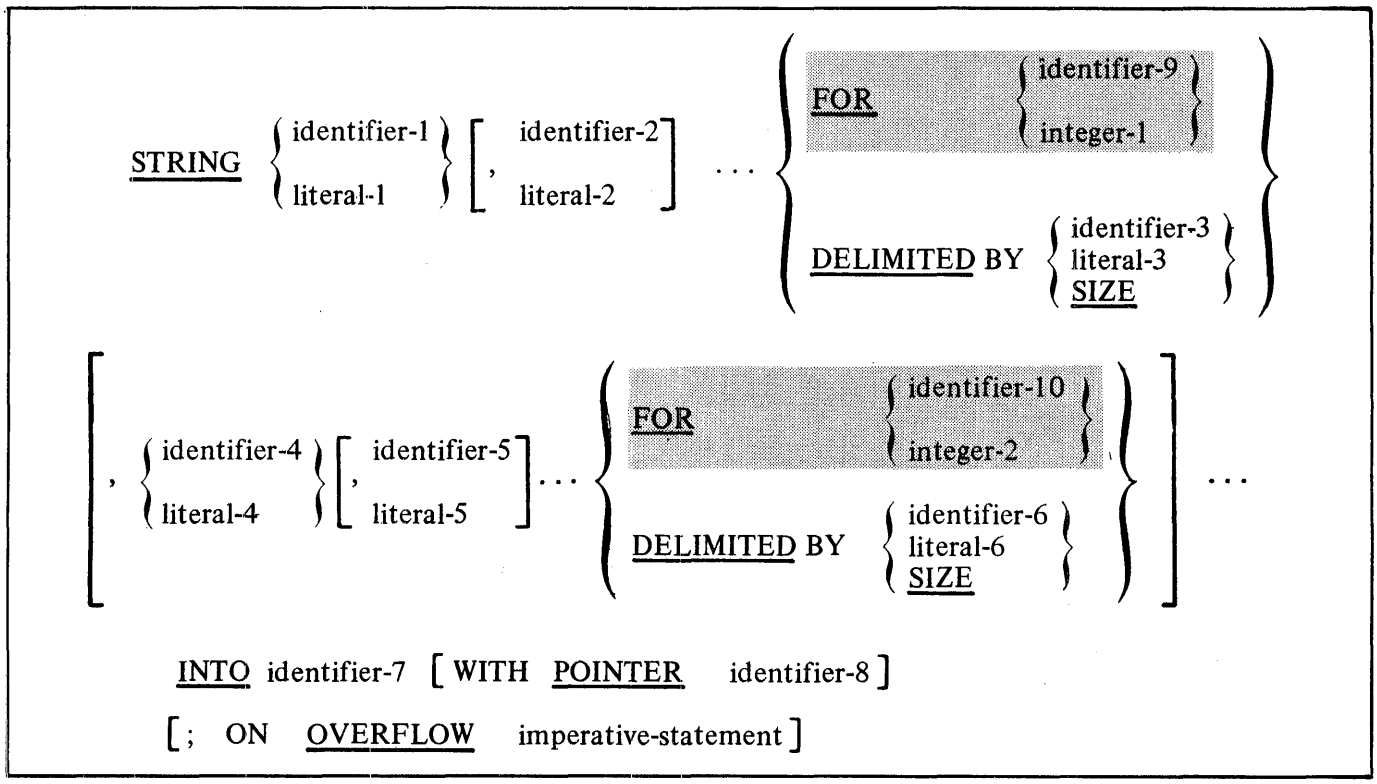
General Rules:

1. If the RUN phrase is used, then the ending procedure established by the installation and/or the compiler is instituted.
2. If STOP literal is specified, the literal is displayed on the Operator Display Terminal (ODT), and the program issues an ACCEPT message. When the operator enters the AX message, continuation of the object program begins with the next executable statement in sequence.

STRING

The STRING statement concatenates partial or complete contents of two or more data items into a single data item.

General Format:



Syntax Rules:

1. Each literal may be any figurative constant without the optional word ALL.
2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-8, identifier-9, and identifier-10, must be described implicitly or explicitly, as usage is DISPLAY.
3. Identifier-7 must represent an alphanumeric data item without editing symbols or the JUSTIFIED clause.
4. Identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size of the area referenced by identifier-7 plus 1. The symbol 'P' may not be used in the PICTURE character-string of identifier-8.
5. Where identifier-1, identifier-2, ..., or identifier-3 is an elementary numeric data item, it must be described as an integer without the symbol 'P' in the PICTURE character-string.
6. Identifier-9 and identifier-10 must represent elementary numeric integer data items.

VERB FORMAT: STRING

General Rules:

1. All references to identifier-1, identifier-2, identifier-3, identifier-9, literal-1, literal-2, literal-3, and integer-1 apply equally to identifier-4, identifier-5, identifier-6, identifier-10, literal-4, literal-5, literal-6 and integer-2, respectively, and all recursions thereof.

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

3. 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 a delimiter, it stands for a single character nonnumeric literal.

4. 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.

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

   a. Those 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 according to the rules for alphanumeric to alphanumeric moves: however, no space-filling is provided. Refer to the MOVE Statement in this section.

   b. 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.

   c. 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 is reached.

   d. If the FOR phrase is specified, 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 number of characters specified by integer-1, or by the contents of identifier-9, are moved.

6. 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.

7. If the POINTER phrase is not specified, the following General Rules 8, 9, 10, 11, and 12 apply as if the user had specified identifier-8 with an initial value of 1.

8. 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. Identifier-8 is increased by one before the move of the next character. The value associated with identifier-8 is changed during execution of the STRING statement only by the behavior specified in this general rule.

9. 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 contain data that was present before this execution of the STRING statement.
10. 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 1 or greater than the number of character positions in the data item referenced by identifier-7, no additional data is transferred to the data item referenced by identifier-7, and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

11. If the ON OVERFLOW phrase is not specified, when the conditions described in General Rule 10 are encountered, control is transferred to the next executable statement.

12. Integer-1, identifier-9, indicate the number of characters to be moved.

Examples:

Assume that:

```
NOW = A, B, C, D ... Z (26 characters)
CHNG = GHI
LATER = LLL
AFTR = A, B, C, D ... Z (26 characters)
THEN  = TTTT
```

1. STRING LATER, "NEW", THEN, SPACE DELIMITED BY SIZE INTO AFTR

   After execution AFTR contains:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 L L L N E W T T T T   L M N O P Q R S T U V W X Y Z
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2. MOVE 1 TO HERE. STRING THEN, "NEXT", SPACE, LATER DELIMITED BY SIZE; NOW DELIMITED BY CHNG INTO AFTR WITH POINTER HERE

   After execution AFTR contains:

```
  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 T T T T N E X T   L L L A B C D E F S T U V W X Y Z
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                    {  not changed  }
```

VERB FORMAT: STRING

3. MOVE 8 TO HERE. STRING THEN, "NEXT", SPACE, LATER DELIMITED BY SIZE; NOW DELIMITED BY CHNG INTO AFTR WITH POINTER HERE

After execution AFTR contains:

```
                  8
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 A B C D E F G T T T T N E X T   L L L A B C D E F Z
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
{ not changed }
```

SUBTRACT

The SUBTRACT statement is used to subtract one or the sum of two or more numeric data items from one or more items, and set the values of one or more items equal to the results.

General Format:

Format 1:

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

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

Format 2:

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

GIVING identifier-n [ROUNDED] [ , identifier-o [ROUNDED] ] ...

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

Format 3:

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ROUNDED]
         { CORR          }

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

VERB FORMAT: SUBTRACT

Syntax Rules:

1. Each identifier must refer to a numeric elementary item except that

   a. In Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or to an elementary numeric edited item.

   b. In Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. Additional rules and explanations related to this statement are given in the appropriate paragraphs. Refer to Intermediate Data Item, ROUNDED Phrase, SIZE ERROR Phrase, CORRESPONDING Phrase, Arithmetic Statements, Overlapping Operands, and Multiple Results in Arithmetic Statements in this section.

2. In Format 1, all literals or identifiers preceding the word FROM are added together, and this total is subtracted from the current value of identifier-m. The result is immediately stored into identifier-m, and this process is repeated respectively for each operand following the word FROM.

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

4. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

Examples:

Assume as initial values : X = 2, Y = 10, Z = 15, TOT = 50, and SUB = 30.

Format 1:

```
SUBTRACT X FROM TOT.                    results TOT=48
SUBTRACT X, Y, Z, FROM TOT, SUB.        results TOT=23, SUB=3
```

Format 2:

```
SUBTRACT X, Y FROM SUB GIVING TOT.      results TOT=18
SUBTRACT X, Y,FROM Z GIVING TOT, SUB.   results TOT=3, SUB=3
```

For Format 3 assume that the following structures have the initial value in parentheses.

```
01  NOW                    01 LATER
    02  AL                     05  AL
      03  FO  (8)                06  FO  (10)
      03  RD  (10)               06  XY  (10)
    02  CR  (20)               05  AB  (10)
    02  TR  (5)                05  TR  (10)
```

The following statement

SUBTRACT CORRESPONDING NOW FROM LATER.

results in

```
02  LATER
    05  AL
        06  FO  (2)
        06  XY  ((10)
    05  AB  (10)
    05  TR  (5)
```

Note, the only data names whose values changed are FO and TR.

UNSTRING

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into one or more receiving fields.

General Format:

Format 1:

```
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]
```

Format 2:

```
UNSTRING identifier-1
    INTO identifier-4 FOR { identifier-12 }
                          { integer-1     }
        [ , identifier-7 FOR { identifier-13 } ] ...
                             { integer-2     }
    [WITH POINTER identifier-10] [TALLYING IN identifier-11]
    [; ON OVERFLOW imperative-statement]
```

VERB FORMAT: UNSTRING

Syntax Rules:

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL.

2. Identifier-1, identifier-2, identifier-3, identifier-5, and identifier-8 must be described, implicitly or explicitly, as alphanumeric data items.

3. Identifier-4 and identifier-7 may be described as either alphabetic, alphanumeric, or numeric and must be described as usage is DISPLAY. However, the alphabetic symbol 'B' and the numeric symbol 'P' may not be used in the PICTURE character-string.

4. Identifier-6, identifier-9, identifier-10, and identifier-11 must be described as elementary numeric integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).

5. No identifier may name a level 88 entry.

6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

7. Identifier-12 and identifier-13 must be described as elementary numeric integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).

General Rules:

1. All references to identifier-2, literal-1, identifier-4, identifier-5, identifier-6, identifier-12, and integer-1, apply equally to identifier-3, literal-2, identifier-7, identifier-8 identifier-13, and integer-2, respectively, and all recursions thereof.

2. Identifier-1 represents the sending area.

3. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

4. Literal-1 or the data item referenced by identifier-2 specifies a delimiter. In Format-2, integer-1 or the data item referenced by identifier-12 specifies a count of the number of characters within identifier-1 which is moved to identifier-4. If the number of characters remaining in the data item referenced by identifier-1 is less than the number of characters specified by integer-1 or the data item referenced by identifier-12, then the short field is transferred according to General Rule 13c.

5. 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).

6. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.

VERB FORMAT: UNSTRING

7. 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.

8. When a figurative constant is used as a delimiter, it stands for a single character nonnumeric literal.

   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 only one occurrence, and this occurrence is moved to the receiving data item according to General Rule 13d.

9. When any examination encounters two contiguous delimiters, the current receiving area is either space or zero filled according to the description of the receiving area.

10. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computer's character set.

11. Each literal-1 or data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given to be recognized as a delimiter.

12. 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 are 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.

13. 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:

    a. 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 contents 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.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until a delimiter specified by the value of literal-1 or the data item referenced by identifier-2 is encountered. Refer to General Rule 11. 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.

    c. The characters examined, excluding any delimiting characters(s), are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement. Refer to the MOVE statement in this section.

d. If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement. Refer to the MOVE statement for additional information. 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.

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

f. 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.

g. 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 behavior described in General Rules 13b through 13f is repeated either until all the characters are exhausted in the data item referenced by identifier-1 or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

15. The contents of the data item referenced by identifier-10 are 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 be a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

16. 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 data receiving items acted upon.

17. Any of the following situations causes an overflow condition:

    a. An UNSTRING statement 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.

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

    c. An UNSTRING is initiated, and the value in the data item referenced by identifier-12 is less than 1 or greater than the size of the data item referenced by identifier-1.

    d. If, during execution of an UNSTRING statement, all data receiving areas have been acted upon, and the number of characters acted upon is less than the value of identifier-12 or integer-1.

VERB FORMAT: UNSTRING

18. An overflow condition occurs if during the execution of an UNSTRING, the value of the pointer item (identifier-10) becomes less than or greater than the number of characters in the sending item, or if all the receiving items have been acted upon and the sending item still contains data 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 statement.

19. The evaluation of subscripting and indexing for the identifiers is as follows:

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

    b. 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.

Examples:

Assume that ALPHA = ABCHXEFGXXZXIJKL, that the size of each of the receiving data items – ITEM-1, ITEM-2, or ITEM-4 is equal to 5, and that ITEM-3 is equal to 4.

1. UNSTRING ALPHA INTO ITEM-1, ITEM-2, ITEM-3, ITEM-4

2. UNSTRING ALPHA DELIMITED BY "X" INTO ITEM-1, ITEM-2, ITEM-3

3. UNSTRING ALPHA DELIMITED BY ALL "X" INTO ITEM-1, ITEM-2, ITEM-3

4. UNSTRING ALPHA DELIMITED BY "X" OR "H" INTO ITEM-1, ITEM-2, ITEM-3

After execution the contents of the receiving fields are as follows.

	ITEM-1	ITEM-2	ITEM-3	ITEM-4
1.	A B C H X	E F G X X	Z X I J	K L
2.	A B C H	E F G		
3.	A B C H	E F G	Z	
4.	A B C		E F G	

USE

The USE statement specifies procedures for input-output error handling that are in addition to the standard procedures provided by the input-output control system.

General Format:

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON
                   { ERROR     }

. { file-name-1 [, file-name-2] ... }
  { INPUT                          }
  { OUTPUT                         }
  { I-O                            }
  { EXTEND                         } .
```

Syntax Rules:

1. A USE statement, when present, must immediately follow a section header in the Declaratives Section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one, or more procedural paragraphs that define the procedures to be used.

2. The USE statement only defines the conditions calling for the execution of the USE procedures, and is never executed.

3. The same file-name can appear in a different specific arrangement of the format. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

5. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

6. EXTEND can only be used for Sequential Files.

General Rules:

1. The designated procedures are executed two ways: first, by the input-output system after completing the standard input-output error routine, or second, upon recognition of the INVALID KEY or AT END conditions when the INVALID KEY phrase or AT END phrase has not been specified in the input-output statement.

2. After execution of a USE procedure, control is returned to the routine that caused the USE procedures to be invoked.

VERB FORMAT: USE

3. 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.

4. Within a USE procedure, there must not be the execution of any statement that causes the execution of a previously invoked USE procedure that has not yet returned control to the invoking routine.

VERB FORMAT: WAIT

WAIT

The WAIT statement suspends the execution of the object program for a specified interval of time or until one or more conditions are true.

General Format:

```
WAIT UNTIL
{ arithmetic-expression
  ODT-INPUT-PRESENT
  { WRITE-OK
    READ-OK
    ATTRIBUTE CHANGEEVENT
    ATTRIBUTE INPUTEVENT
    ATTRIBUTE OUTPUTEVENT } [ IN ] file-name-1 [ (identifier-1) ]
                           [ OF ]              [ (integer-1)  ]
                           [ ON ]
} . . .
[USING identifier-2]
[GIVING identifier-3]
```

Syntax Rules:

1. No more than one arithmetic-expression can be specified, and if specified, it must be the first item in the list. The arithmetic-expression must be a data-name or literal.
2. The data items referenced by identifier-1, identifier-2, and identifier-3 must be described elementary numeric integer data items without the symbol "P" in its PICTURE character-string.
3. If identifier-1 or integer-1 is specified, file-name-1 must reference a queue file.
4. If ATTRIBUTE CHANGEEVENT, ATTRIBUTE INPUTEVENT, or ATTRIBUTE OUTPUTEVENT is specified, file-name-1 must reference a port file.
5. The maximum number of events which can be specified in a WAIT statement is 15.

General Rules:

1. When arithmetic-expression is specified, the execution of the object program is suspended for the number of seconds determined by the value of the expression. If the value of the expression is negative, the number of seconds determined by the value is zero.
2. When the ODT-INPUT-PRESENT option is specified, the execution of the object program is suspended until a message is entered from the Operator Display Terminal (ODT) for the program.
3. When identifier-1 or integer-1 is specified, the value specified must be a valid subfile number of the file. If the file is a queue file family and no subscript is specified, or if the value of identifier-1 or integer-1 is zero, the event is always true.

VERB FORMAT: WAIT

4. When the WRITE-OK option is specified, the execution of the object program is suspended until there is space in file-name-1 for at least one more record. For serial I-O devices such as punch or tape, the WRITE-OK event is true when there is space in file-name-1 for at least one record. For disk and backup files, this event is always true. For files that are not open, this event is always false. This event is not valid for port files.

5. When the READ-OK option is specified, the execution of the object program is suspended until there is at least one record in file-name-1 to be read. For serial I-O devices such as card readers and tape, the READ-OK event is true when a record is present in the buffer. For disk and pseudo-readers, this event is always true. For files that are not open, this event is always false. This event is not valid for port files.

6. The event ATTRIBUTE CHANGEEVENT becomes true when the FILESTATE attribute of the port file referenced by file-name-1 changes. (Refer to the B 1000 Systems Burroughs Network Architecture (BNA) Operation and Installation Manual for examples of how to use this and the following two events.)

7. The event ATTRIBUTE INPUTEVENT becomes true when there is a message to be read from the port file referenced by file-name-1.

8. The event ATTRIBUTE OUTPUTEVENT becomes true when output buffers become available to write to the port file referenced by file-name-1.

9. If any event in the list of conditions is true, the wait operation is terminated and control is passed to the next executable statement.

   If, in the initial scan of the events, no event is found to be true, program execution is suspended until any event in the list becomes true. At that time the wait operation is terminated and control is passed to the next executable statement.

   When the USING clause is not specified, each event in the list is tested for a true condition beginning with the first event in the list and proceeding to the last event in the list.

10. When the USING clause is specified, the value referenced by identifier-2 determines the starting position in the event list where the wait operation begins testing for a true condition. For example, a value of 2 in identifier-2 causes the wait operation to start with the second event in the list, proceed through the events to the last event in the list, and then test the first event in the list. If the value specified by identifier-2 is zero or greater than the number of events in the list, the program is terminated.

11. When the GIVING clause is specified, the data item referenced by identifier-3 is set to the position in the list of events which terminated the wait operation. For example, if the second event in the list of events is found to be true, the data item referenced by identifier-3 is set to the value 2.

    If the event that is found to be true is a queue family, there is no notification as to which subfile in the queue family caused the true condition.

WRITE

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines within a logical page.

General Format:

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

Format 2:

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

Format 3:

```
WRITE record-name [WITH NO WAIT] [FROM identifier-1]
                  [; INVALID KEY imperative-statement]
```

Syntax Rules:

1. Record-name and identifier-1 must not reference the same storage area.
2. When mnemonic-name is specified, it must be associated with a CHANNEL number. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.
3. The record-name is the name of a logical record in the FILE SECTION of the DATA DIVISION and may be qualified.
4. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

VERB FORMAT: WRITE

5. Integer-1 or the value of the data item referenced by identifier-2 may be zero.

6. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry (FD) for the associated file.

7. The words END-OF-PAGE and EOP are equivalent.

8. The ADVANCING mnemonic-name phrase cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.

9. Format 1 must be used if the ACTUAL KEY phrase is not specified in the file-control entry for the file associated with record-name. Format 2 must be used if the ACTUAL KEY phrase is specified in the file-control entry for the file associated with record-name.

10. Format 2 is used for Relative and Indexed Files and also Sequential Files in the Random access mode.

11. In Format 2, the INVALID KEY phrase must be specified if an applicable use procedure is not specified for the associated file.

12. Format 3 must be used for port or queue files only.

General Rules:

These rules follow under the following headings:

Mass and Non-Mass Storage Files

Non-Mass Storage Files

Mass Storage Files
1) Sequential Files – (Queue files)
2) Indexed Files
3) Relative Files

Mass and Non-Mass Storage Files

1. The associated file must be open in the OUTPUT, INPUT-OUTPUT, or EXTEND mode at the time of the execution of this statement, and must not be a SORT or MERGE File. Refer to the OPEN statement in this section.

2. The execution of a WRITE statement has no effect upon either the contents or accessibility of the record area. If the associated file is named in the SAME RECORD AREA clause, the logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause; the associated output file and the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of

   a. The statement:

      MOVE identifier-1 TO record-name

      according to the rules specified for the MOVE statement, followed by

   b. The same WRITE statement without the FROM phrase.

   The contents of the record area before the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. Refer to I-O Status in section 5.

6. The maximum record size for a file is established when the file is created and must not subsequently be changed.

7. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

8. The execution of the WRITE statement releases a logical record to the operating system.

Non-Mass Storage Files

9. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing is provided to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

   a. If identifier-2 is specified, the page is advanced the number of lines equal to the current value of identifier-2.

   b. If integer-1 is specified, the page is advanced the number of lines equal to the value of integer-1.

   c. If mnemonic-name is specified, the page is advanced to the line number corresponding to the CHANNEL number.

   d. If the BEFORE phrase is used, the line is written before the page is advanced according to rules 9a, 9b, and 9c.

   e. If the AFTER phrase is used, the line is written after the page is advanced according to rules 9a, 9b, and 9c.

VERB FORMAT: WRITE

f. If PAGE is specified, the record is written on the logical page BEFORE or AFTER (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file containing a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause. If the record to be written is associated with a file which does not contain a LINAGE clause, the repositioning is to CHANNEL 1 or line 1 of the next logical page when appropriate for the hardware device.

   If PAGE has no meaning in conjunction with a specific device, then advancing is provided by the compiler to act as if the user had specified BEFORE or AFTER ADVANCING 1 LINE. In either case, page positioning depends on the phrase used.

10. If the logical end of the page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative-statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name.

11. An end-of-page condition is reached whenever the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then the imperative statement in the END-OF-PAGE phrase is executed.

    An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

    Overflow occurs when a WRITE statement, if executed, causes the LINAGE-COUNTER to exceed the value specified by integer-1 or to exceed the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is written (depending on the phrase used) on the logical page before or after the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

    If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

    If integer-2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both integer-2 (or the data item referenced by data-name-2) and integer-1 (or the data item referenced by data-name-1), then the operation proceeds as if integer-2 or data-name-2 had not been specified.

12. After the recognition of an end-of-reel or an end-of-unit of an output file that is contained on more than one physical reel/unit, the WRITE statement performs the following operations:

    a. The standard ending reel/unit label procedure.

    b. A reel/unit swap.

c. The standard beginning reel/unit label procedure.

Mass Storage Files

13. When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful. The contents of the record area are unaffected and the FILE STATUS data item, if any, associated with the file-name of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated for the INVALID KEY condition. Refer to I-O Status in section 5.

Sequential Files:

14. When an attempt is made to write beyond the externally defined boundaries of a Sequential File, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

    a. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation. Refer to I-O Status in section 5.

    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure is then executed.

    c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

15. If the ACTUAL KEY phrase is specified for a mass storage file whose access mode is sequential, the successful execution of a Format 2 WRITE statement updates the contents of the ACTUAL KEY data item to the ordinal number of the logical record written.

16. For a mass storage file whose access mode is sequential, the execution of a Format 2 WRITE statement releases the record area to the next logical record in the file.

17. For a mass storage file whose access mode is sequential, the INVALID KEY condition exists when a maximum logical size is specified for the file and no more logical records may be written.

18. For a mass storage file whose access mode is random, the execution of a Format 2 WRITE statement releases the record area to the logical record of the file specified by the contents of the ACTUAL KEY data item.

19. For a mass storage file whose access mode is random, an INVALID KEY condition exists when the value of the ACTUAL KEY data item is less than 1 or greater than the ordinal number of the last logical record allowed for the file, provided that the maximum logical size of the file is specified.

20. For a queue file, execution of a WRITE statement releases a logical message to the operating system as follows:

    a. If the queue file is not connected to a queue file family, the message is released to the queue file. If an ACTUAL KEY data item is specified for the file, its contents are ignored.

VERB FORMAT: WRITE

b. If the queue file is connected to a queue file family, the contents of the ACTUAL KEY data item associated with the queue file specifies the index of the subfile to which the message is to be released. An INVALID KEY condition exists if the value of the ACTUAL KEY data item is less than one or greater than the maximum number of subfiles in the family.

21. When the WITH NO WAIT option for queue files is specified, the program senses a Q-Full condition (the queue contains the maximum number of messages it can store), and continues with the next executable instruction. If the WITH NO WAIT option is not specified and a Q-Full condition occurs, the program is suspended until the write operation is completed.

22. A WRITE statement causes the program to wait until a buffer is available to store the record. Suspension of the program can be prevented for a port file by using the WITH NO WAIT phrase. A status key value of 95 is returned when no buffer is available for the logical record.

23. If an ACTUAL KEY clause is specified for a port file, the user is responsible for updating the value of the ACTUAL KEY data item with an appropriate value. A WRITE statement passes the value of the ACTUAL KEY data item to the I-O system which forwards the message to the desired subfile. If the ACTUAL KEY value is zero, a "broadcast" write operation is performed. For a broadcast write operation, the message is sent to all opened subfiles of the port file.

    If no ACTUAL KEY clause is specified for the port file, the port file must have only one subfile. All messages are written to that subfile.

Indexed Files:

24. For an Indexed File, execution of the WRITE statement causes the contents of the record area to be released. The contents of the record keys are utilized such that subsequent access of the record key may be made based upon any of those specified record keys.

25. The value of the prime record key must be unique within the records in the Indexed File.

26. In an Indexed File, the data item specified as the prime record key must be set by the program to the desired value before the execution of the WRITE statement.

27. If the sequential access mode is specified for an Indexed File, records must be released in ascending order of prime record key values.

28. If random or dynamic access mode is specified for an Indexed File, records may be released in any program-specified order.

29. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an Indexed File, the value of the alternate record key may be nonunique only if the DUPLICATES phrase is specified for that data item. In this case, the storage of records is such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released.

30. The INVALID KEY condition exists under any of the following circumstances:

    a. When sequential access mode is specified for a file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record.

    b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file.

    c. When the file is opened in the output or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file.

    d. When an attempt is made to write beyond the externally defined boundaries of the file.

Relative Files:

31. When a Relative File is opened in the output mode, records may be placed into the file in one of the following ways:

    a. If the access mode is sequential, the WRITE statement causes a record to be released. The first record has a relative record number of 1 and subsequent records released have relative record numbers of 2, 3, 4, and so on. If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released is placed into the RELATIVE KEY data item during execution of the WRITE statement.

    b. If the access mode is random or dynamic before the execution of the WRITE statement, the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released by execution of the WRITE statement.

32. When a Relative File is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released.

33. The INVALID KEY condition exists under any of the following circumstances:

    a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists in the file.

    b. When an attempt is made to write beyond the externally defined boundaries of the file.

SECTION 8
FILE ATTRIBUTES

GENERAL

File attributes provide the capability for defining, monitoring, or changing properties or attributes of a file.

FILE ATTRIBUTE IDENTIFIER

A file attribute-identifier is a word or sequence of words sufficient to make unique reference to an attribute of a specific file.

General Format:

```
ATTRIBUTE  attribute-name { OF } file-name [(subfile-specifier)]
                          { IN }
```

Syntax Rules:

1. Attribute-name is a system name. Refer to appendix D for additional information.

2. A subfile-specifier is required for some attributes associated with subfiles of a port file. Otherwise, the subfile-specifier has no meaning. The subfile-specifier must be a numeric literal or data-name.

3. An attribute identifier can be changed through the use of the CHANGE statement in the PROCEDURE DIVISION. Refer to the CHANGE statement paragraph in this section.

General Rules:

1. A file attribute may be classified as belonging to one of three categories, depending upon the attribute-name specified in the file attribute-identifier.

   a. Alphanumeric attribute-names.

      An alphanumeric attribute-name is equivalent to an elementary alphanumeric DISPLAY data item having a size equal to the maximum allowed for the specified attribute. Alphanumeric attribute-names are allowed as operands of relational conditions and as sending operands of Format 1 MOVE statements. All alphanumeric attribute-names currently available are:

      HOSTNAME
      INTNAME
      MYHOSTNAME
      MYNAME
      SERIALNO
      TITLE
      YOURNAME
      YOURUSERCODE

FILE ATTRIBUTE

b. Numeric attribute-names.

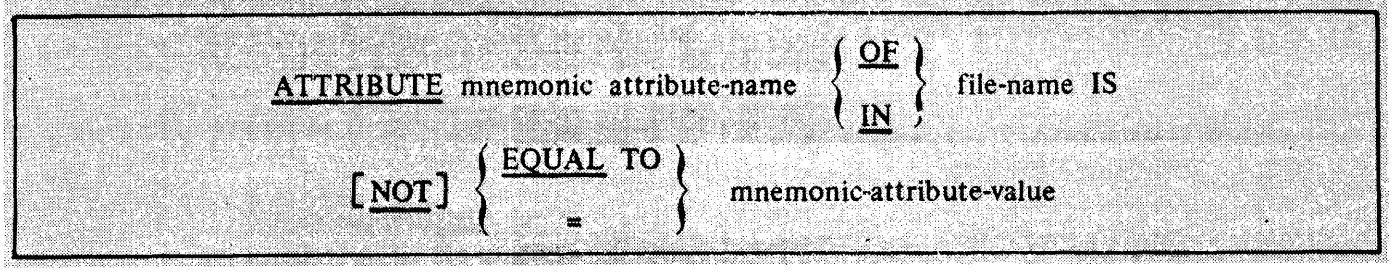
A numeric attribute-name is equivalent to an elementary data item that represents a signed integer with eight decimal digits. Numeric attribute-names are allowed as operands of relational conditions and as sending operands of Format 1 MOVE statements. All numeric attribute-names currently available are:

AREALENGTH
AREAS
BLOCK
BLOCKSIZE
BUFFERS
CENSUS
CHANGEDSUBFILE
CREATIONDATE
CURRENTBLOCK
FILESECTION
FRAMESIZE
LASTRECORD
LASTSTATION
LASTSUBFILE
MAXCENSUS
MAXRECSIZE
MAXSTATIONS
MAXSUBFILES
MINRECSIZE
NEXTRECORD
RECORD
SAVEFACTOR
USEDATE
VOLUMEINDEX

c. Mnemonic attribute-names.

Certain file attributes are associated with values which are best expressed as mnemonic names, since the magnitude of the actual value is unrelated to the meaning.

Mnemonic attribute-names are allowed as operands in limited relational condition expressions as follows:



FILE ATTRIBUTE

Mnemonic attribute relation conditions may not be abbreviated. The names for the mnemonic attribute values are system names (refer to appendix D, Glossary), and are not necessarily reserved words.

Mnemonic attribute-names have only mnemonics for values. All mnemonic attribute-names are allowed as operands of relational conditions. All mnemonic attributes and associated mnemonic values currently available are:

Mnemonic Attribute	Mnemonic Value
ATTERR	TRUE, FALSE
AUDITED	TRUE, FALSE
BACKUPKIND	DISK, TAPE, DONTCARE
BACKUPPERMITTED	DONTBACKUP, DONTCARE, MUSTBACKUP
BLOCKSTRUCTURE	FIXED, VARIABLE
CHANGEEVENT	TRUE, FALSE
COMPRESSION	TRUE, FALSE
DENSITY	BPI200, BPI556, BPI800, BPI1600, BPI6250
DEPENDENTSPECS	TRUE, FALSE
DIRECTION	FORWARD, REVERSE
EXTMODE	EBCDIC, ASCII, BINARY
FILEKIND	DATA, CODEFILE, INTRINSICFILE, CONTROLDECK
FILESTATE	AWAITINGHOST, BLOCKED, CLOSED, CLOSEPENDING, DEACTIVATED, DEACTIVATIONPENDING, OFFERED, OPENED, SHUTTINGDOWN
FLEXIBLE	TRUE, FALSE
INPUTEVENT	TRUE, FALSE
KIND	DISK, PAPERPUNCH, PAPERREADER, PRINTER, PUNCH, PUNCH80, PUNCH96, READER, READER80, READER96, ODT, TAPE, TAPECASSETTE, TAPEPE, TAPE9, TAPE7
LABEL	EBCDICLABEL, ASCIILABEL, OMITTED
MYUSE	IN, OUT, IO
NEWFILE	TRUE, FALSE
OPEN	TRUE, FALSE
OPTIONAL	TRUE, FALSE
OTHERUSE	SECURED, IN, OUT, IO
OUTPUTEVENT	TRUE, FALSE
PARITY	ODD, EVEN
PRINTDISPOSITION	DONTPRINT, CLOSE, EOJ
SECURITYTYPE	PUBLIC, PRIVATE
SUBFILERROR	DATALOST, DISCONNECTED, NOBUFFER, NOERROR, NOFILEFOUND, UNREACHABLEHOST
TRANSLATE	FORCESOFT, NOSOFT
TRANSLATING	TRUE, FALSE
UPDATEFILE	TRUE, FALSE
USERBACKUPNAME	TRUE, FALSE

FILE ATTRIBUTE

2. The subfile-specifier option can be used only with a port file. The value of the data-name or literal specifies which subfile of the port file is affected.

   If the subfile-specifier option is not specified, the port file attribute is accessed.

   If the value of the data-name or literal used as a subfile-specifier is non-zero, the value specifies a subfile index and that subfile attribute is accessed.

   If the value of the data-name or literal is zero, all subfile attributes are accessed.

CHANGE

The CHANGE statement allows the procedural modification or redefinition of the attributes of a file.

General Format:

Format 1:

```
                                  { numeric attribute-name      }   { OF }
CHANGE ATTRIBUTE                  { alphanumeric attribute-name }   { IN }   file-name

                                  TO { identifier }
                                     { literal    }
```

Format 2:

```
                                                                  { OF }
CHANGE ATTRIBUTE mnemonic attribute-name                          { IN }   file-name

                                  TO mnemonic-attribute-value
```

Syntax Rules:

1. In Format 1, if an alphanumeric attribute-name is specified, the literal must be a nonnumeric literal and the identifier must be a nonnumeric DISPLAY data item. If a numeric attribute-name is specified, the literal must be a numeric literal and the identifier must be a numeric data item that represents an integer. The B 1000 COBOL74 compiler does not verify the values represented by identifier, or literal.

2. In Format 2, the mnemonic-attribute-value must be associated with the attribute specified. The B 1000 COBOL74 compiler does not verify the value represented by mnemonic-attribute-value.

3. Certain attributes are not allowed to be changed at any time. These attributes may not be specified in a CHANGE statement.

General Rules:

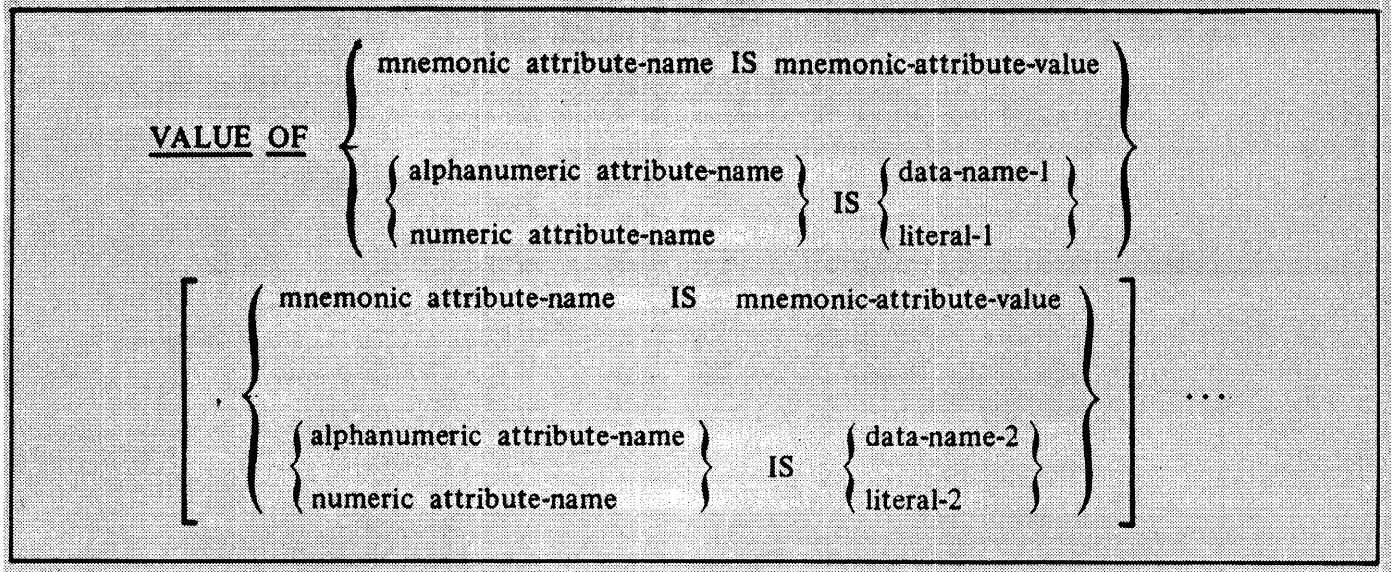
1. Certain attributes may not be changed while the file is in an open mode. Attempts to CHANGE these attributes while the file is open are ignored.

2. In Format 1, attempts to CHANGE attributes to values which are invalid are ignored.

VALUE OF

The VALUE OF clause of the File Description entry (FD) may be used to define the initial values for the attributes of a file.

General Format:



Syntax Rules:

1. Data-name-1, data-name-2, ..., may be qualified but cannot be subscripted or indexed.
2. The mnemonic-attribute-value must be associated with the attribute specified.
3. If an alphanumeric attribute-name is specified, the literal-1, literal-2, ... must be a nonnumeric literal and the identifier must be a nonnumeric DISPLAY data item. If a numeric attribute-name is specified, the literal must be a numeric literal and the identifier must be a numeric data item that represents an integer.
4. Use of data-name-1, data-name-2, ..., is only valid for the following attributes:

   AREALENGTH
   AREAS
   BLOCKSIZE
   BUFFERS
   INTNAME
   MAXRECSIZE
   MINRECSIZE
   SAVEFACTOR
   SERIALNO
   TITLE
   VOLUMEINDEX

VALUE OF

General Rules:

1. The descriptive clauses and phrases of the INPUT-OUTPUT SECTION and the file and record descriptions, other than the VALUE OF clause, implicitly determine the initial values for appropriate attributes of a file. These attribute values, however, may be overridden or other attributes specified by the VALUE OF clause.

2. If an attribute-name is equated to a literal value, this value becomes part of the file description given by the file when first referenced at run time. Any specification in this file description may be overridden by label equation.

3. If an attribute-name is equated to a data-name value, the attribute is implicitly changed to this value just prior to the execution of any explicit OPEN, SORT, or MERGE statement that references the file.

ATTRIBUTE NAME DESCRIPTIONS

FILE ATTRIBUTE-NAME DESCRIPTIONS

The following is a description of each of the attribute-names.

AREALENGTH
Specifies the number of bytes in an area of a disk file. It represents the value of BLOCKSIZE multiplied by the number of blocks per area. For relative and ISAM files, BLOCKSIZE must include the Block Control Information. The maximum value allowed for AREALENGTH is 2097151 bytes. This attribute may be read anytime or changed when the file is closed. Changing the value of AREALENGTH for an existing file has no meaning and is ignored by the operating system.

AREAS
Specifies the number of areas that can be allocated for a disk file. This attribute may be read anytime. It may be increased when the file is closed if the FLEXIBLE file attribute is true. The maximum number of areas allowed for a file is increased to the B 1000 system maximum of 105 for any file opened or created when the FLEXIBLE attribute is true.

ATTERR
Indicates whether the last file attribute action was in error. Mnemonic values are TRUE or FALSE. This attribute may be read anytime.

AUDITED
This attribute causes disk file updates to be recorded immediately in the file records rather than through a list of memory buffers. Mnemonic values are TRUE or FALSE. This attribute can be read anytime or changed when the file is closed. When an ISAM file has more than one user and one user specifies AUDITED, the ISAM file remains in the AUDITED state for all users, until the last user requesting that the file be audited goes to end-of-job.

BACKUPKIND
Specifies the device type to be used for printer backup files. Mnemonic values are DISK, TAPE, or DONTCARE. For the mnemonic DONTCARE, the KIND of the backup file is determined in the same manner as if BACKUPKIND were never set. This attribute may be read anytime or changed when the file is closed.

BACKUPPERMITTED
Specifies whether a printer backup file may be assigned. Mnemonic values are DONTBACKUP when no backup is allowed, DONTCARE when backup is allowed, and MUSTBACKUP when backup is required. This attribute may be read anytime or changed when the file is closed.

BLOCK
Indicates the number of the logical block referenced in the last I-O statement. This attribute may be read anytime.

BLOCKSIZE
Specifies the value of the length of a block in FRAMESIZE units. This attribute may be read anytime or changed when the file is closed.

BLOCKSTRUCTURE
Specifies the format of records of a file. Mnemonic values are FIXED or VARIABLE. This attribute may be read anytime or changed when the file is closed.

ATTRIBUTE NAME DESCRIPTIONS

BUFFERS
Specifies the number of buffers assigned to a file. This attribute may be read anytime or changed when the file is closed.

CENSUS
Indicates the number of messages in a subfile of a port file or queue file. This subfile attribute can be read when the subfile is open.

CHANGEDSUBFILE
Indicates the number of a subfile with a true CHANGEEVENT subfile attribute. The CHANGEDSUBFILE port file attribute can be read when one or more subfiles are open.

CHANGEEVENT
The CHANGEEVENT subfile attribute is true whenever the value of the FILESTATE attribute changes. The CHANGEEVENT subfile attribute is reset when the FILESTATE subfile attribute is interrogated. The CHANGEEVENT port file attribute can be read when one or more subfiles are open.

CREATIONDATE
Indicates the creation date of a disk file with a string of five characters, (YYDDD). This attribute may be read anytime.

COMPRESSION
Specifies that data is compressed for transmission by means of a subfile of a port file. Mnemonic values are TRUE or FALSE. This subfile attribute can be read anytime or changed when the subfile is closed.

CURRENTBLOCK
Indicates the size, in FRAMESIZE units, of the block currently in use. This attribute may be read anytime.

DENSITY
Specifies the recording density of a magnetic tape file. Mnemonic values are BPI200, BPI556, BPI800, BPI1600, or BPI6250. This attribute may be read anytime or changed when the file is closed.

DEPENDENTSPECS
Specifies whether the attributes BLOCKSTRUCTURE, MINRECSIZE, MAXRECSIZE, BLOCKSIZE, and FRAMESIZE of a logical file are to be changed to agree with corresponding values of an associated permanent file. Mnemonic values are TRUE or FALSE. This attribute may be read anytime or changed when the file is closed.

DIRECTION
Specifies the direction in which records of a file are to be accessed. Mnemonic values are FORWARD or REVERSE. This attribute may be read anytime or changed when the file is closed.

EXTMODE
Specifies the recording mode of records within a file. Mnemonic values are EBCDIC, ASCII, or BINARY. This attribute may be read anytime or changed when the file is closed.

FAMILYNAME
Specifies the identifiers of a disk where the file will reside. This attribute may be read anytime or changed when the file is closed.

ATTRIBUTE NAME DESCRIPTIONS

FILEKIND
Specifies the purpose of a disk file. Mnemonic values are DATA, CONTROLDECK for a pseudo reader file, CODEFILE for an object code file, or INTRINSICFILE for a file containing one or more intrinsics. This attribute may be read anytime or changed anytime.

FILESECTION
Specifies the reel number of a tape file. This attribute may be read anytime or changed when the file is closed.

FILESTATE
Indicates the current disposition of a subfile of a port file. Mnemonic values and their descriptions follow.

ATTRIBUTE NAME DESCRIPTIONS

Mnemonic	Description
AWAITINGHOST	The host specified by the HOSTNAME subfile attribute is unreachable. The subfile remains in this state until the host becomes reachable. The FILESTATE can then change to OFFERED, OPENED, or CLOSED. I/O operations are not valid when the file is in the AWAITINGHOST state.
BLOCKED	The remote host has become temporarily unreachable. The port file remains open and all I/O operations are valid.
CLOSED	Initial state of a subfile. The subfile returns to this state when it is closed by the user.
CLOSEPENDING	The user has closed the subfile, but the remote subfile has not yet acknowledged the closure. When acknowledgment of the closure is received, the FILESTATE subfile attribute is changed to CLOSED.
DEACTIVATED	The remote subfile has been closed and this subfile does not have data queued for input. Close is the only valid operation for a subfile in this state.
DEACTIVATIONPENDING	The remote subfile has been closed and this subfile has data queued for input.
OFFERED	A file open has occurred and the host specified by the HOSTNAME subfile attribute is reachable, but no matching subfile has been found. I/O operations are not valid when the file is in this state.
OPENED	The subfile is open and can be used to send or receive data.
SHUTTINGDOWN	The system operator has requested that communications with the host involved in the subfile dialog be terminated. This notification gives the program the opportunity to terminate in an orderly manner. The port file remains open and all I/O operations are valid.

ATTRIBUTE NAME DESCRIPTIONS

The FILESTATE subfile attribute can be read when the subfile is open.

FLEXIBLE
Specifies whether a disk file may be allocated more than the number of areas originally specified. The mnemonic value may be TRUE or FALSE. This attribute may be read anytime or changed when the file is closed.

NOTE
Use of the FLEXIBLE attribute to increase the maximum number of areas for an existing ISAM file can cause one or more of the subfiles to exceed its alloted space prior to the data file exhausting its maximum number of areas.

FRAMESIZE
Specifies the number of bits to be transferred as a unit of data. For B 1000 Systems, FRAMESIZE is 8. This is the same as the number of bits required to represent an EBCDIC character. This attribute may be read anytime or changed when the file is closed.

HOSTNAME
Specifies the name of the host system on which the file resides. This subfile attribute can be read anytime or changed when the subfile is closed.

INTNAME
Specifies the value of the internal file name. This attribute may be read anytime or changed when the file is closed.

KIND
Specifies the device to be associated with the logical file. Mnemonic values are DISK, PRINTER, PUNCH, PAPERPUNCH, PAPERREADER, READER, READER80, READER96, ODT, TAPE, TAPECASSETTE, TAPEPE, TAPE7, or TAPE9. This attribute may be read anytime or changed when the file is closed.

LABEL
Specifies whether or not the file has label records. Mnemonic values are EBCDICLABEL, ASCII-LABEL, STANDARD or OMITTED. This attribute may be read anytime or changed when the file is closed.

LASTRECORD
Indicates the record number of the last record in the physical file. This attribute may be read only when the file is opened.

ATTRIBUTE NAME DESCRIPTIONS

LASTSTATION
Specifies a one relative number that indicates the last logical station from which a remote file was READ or to which a remote file will be written. This attribute may be read anytime or changed prior to a write.

LASTSUBFILE
Indicates the subfile number of the last subfile for which an input or output operation has occurred. This attribute may be read only when the port or queue file is open.

MAXCENSUS
Indicates the maximum number of messages that can exist in a subfile of a port or queue file. This subfile attribute can be read when the file is open.

MAXRECSIZE
Specifies the maximum size of records in FRAMESIZE units. This attribute may be read anytime or changed when the file is closed.

MAXSTATIONS
Specifies the maximum number of stations that can be attached to a remote file. This file attribute can be read anytime or changed when the remote file is closed. If this file attribute is not specified, the default value is 1.

MAXSUBFILES
Specifies the maximum number of subfiles of a port or queue file. This attribute can be read anytime or changed when the port file is closed. If this port file attribute is not specified, the default value is 1.

MINRECSIZE
Specifies the minimum size of records in FRAMESIZE units. This attribute may be read anytime or changed when the file is closed.

MYHOSTNAME
Indicates the name of the host system from which a port file is transmitting. This port file attribute can be read anytime.

MYNAME
Indicates the name of a port file. This port file attribute can be read anytime.

MYUSE
Specifies whether the file is used for input, output, or both. Mnemonic values are IN, OUT, or IO. This file attribute can be read anytime or changed when the file is closed.

NEWFILE
Specifies whether the file is a new file. Mnemonic values are TRUE or FALSE. This attribute may be read anytime or changed when the file is closed.

NEXTRECORD
Indicates the relative record number of the next record to be processed in an I-O statement. This attribute may be read anytime.

ATTRIBUTE NAME DESCRIPTIONS

OPEN
Indicates whether the file is open or not. Mnemonic values are TRUE or FALSE. This attribute may be read anytime.

OPTIONAL
Specifies whether or not the assignment of a permanent file is optional. Mnemonic values are TRUE or FALSE. This attribute may be read anytime or changed when the file is closed.

OTHERUSE
Specifies how the files can be used by other programs during the time a program has the file opened. Mnemonic values are SECURED for neither input nor output, IN for input only, OUT for output only, and IO for both input and output. This file attribute can be read anytime or changed when the file is closed.

PARITY
Specifies the parity used for TAPE or PAPERTAPE files. Mnemonic values are EVEN or ODD. This attribute may be read anytime or changed when the file is closed.

PRINTDISPOSITION
This attribute is valid for printer files only. The PRINTDISPOSITION file attribute specifies whether and when a backup print file is to be automatically printed. The DONTPRINT mnemonic value will suppress the printing of the file when the system option AUTOPRINT is in effect. The mnemonic value CLOSE will cause the file to be automatically printed as soon as the file is closed. The EOJ mnemonic value will cause the file to be included in the job summary if it belongs to a task within a WFL job. Otherwise, EOJ is the same as CLOSE. The default value is EOJ. This attribute may be read anytime.

RECORD
Indicates the current relative record number of a file. This attribute may be read anytime.

SAVEFACTOR
Specifies the number of days following the creation date, after which the retention period for a tape file is considered to have expired. This attribute may be read anytime or changed when the file is closed.

SECURITYTYPE
Specifies what kind of security applies to the file. Mnemonic values are PRIVATE for access by only a privileged user or the owner, or PUBLIC for access by any user who references the file using the "(usercode)/file-name" form of the file name. This file attribute can be read anytime or changed when the file is closed.

SERIALNO
Specifies the serial number of the labeled tape or base member of the disk family to which the logical file is assigned. The serial number is an EBCDIC string of six characters, left-justified in a field of blanks. This attribute may be read anytime or changed when the file is closed.

SUBFILERROR
Indicates the current status of an error register of a subfile of a port file following an I-O operation on the subfile. This is a read only attribute and is always set. Mnemonic values and their descriptions follow.

ATTRIBUTE NAME DESCRIPTIONS

Mnemonic	Description
DATALOST	A close operation or an abort occurred before all messages were sent.
DISCONNECTED	The subfile has been disconnected from the remote subfile connected with it.
NOBUFFER	The last operation of this subfile was a WRITE and the attempt to place the message in a queue failed.
NOERROR	There were no errors on the last I-O on the subfile.
NOFILEFOUND	An OPEN AVAILABLE was attempted and no matching subfile was found.
UNREACHABLEHOST	An OPEN operation was started but the remote host became unavailable while the OPEN was in process.

This attribute can be read at any time.

TITLE

Specifies the external file name with the form "B/C ON A". The "B/C" portion represents the multi-file-id and file-id. The "A" portion represents the pack-id. Valid TITLEs may be:

```
B
B/C
B ON A
B/C ON A
```

When creating multifile tapes, the multi-file-id must be the same for all files on the tape. Only the file-id may change.

For indexed sequential files (ISAM), only the global file receives the name specified in the VALUE OF TITLE statement. The data and index files conform to the naming convention described in appendix G of this manual.

The TITLE attribute, when used with the USERBACKUPNAME file attribute will assign a user declared name to a backup print file.

This attribute may be read anytime or changed when the file is closed. Refer to the B 1000 Systems Software Operation Guide, Volume 1, for the formation of file names.

TRANSLATE

Specifies whether software translation takes place when using tables. Mnemonic values are FORCESOFT or NOSOFT. This file attribute can be read anytime or changed when the file is closed.

ATTRIBUTE NAME DESCRIPTIONS

TRANSLATING
Indicates when software translation is being performed on the records of the file. Mnemonic values are TRUE or FALSE. This file attribute may be read anytime or changed when the file is closed.

UPDATEFILE
Specifies whether or not the disk file is used with an update I-O access method. Mnemonic values are TRUE or FALSE. This attribute may be read anytime or changed when the file is closed.

USEDATE
Indicates the Julian date that a disk file was last used. This attribute may be read when the file is opened.

USERBACKUPNAME
When the USERBACKUPNAME file attribute is set, a user specified name can be declared for a print file. The file name can be declared either with the VALUE OF TITLE statement, a file equation at execution time, or a run-time dynamic modify. This attribute may be read anytime or changed when the file is closed.

VOLUMEINDEX
Specifies the reel number of a tape file. This attribute may be read anytime or changed when the file is closed.

YOURNAME
Specifies the name of a corresponding subfile of a port file with which this subfile of a port file is to be matched. This subfile attribute can be read anytime or changed when the subfile is closed.

YOURUSERCODE
Specifies the usercode of a corresponding subfile of a port file with which this subfile of a port file is to be matched. This subfile attribute can be read anytime or changed when the subfile is closed.

SECTION 9
DATA BASE MANAGEMENT

GENERAL

The Data Base section provides the ability to store, retrieve, and delete information in a data base composed of storage structures which may have hierarchical or other associations relating one to another. It provides the ability to communicate with the Data Management System (DMSII) which controls access to the data base. Refer to the B 1000 Systems Data Management System II (DMSII) Reference Manual for further details on Data Management System implementation.

DATA-BASE SECTION

The DATA-BASE SECTION serves the purpose of supplying the COBOL74 compiler with a description of all or selected portions of one data base.

Data Base Structure

In a COBOL74 program the DATA-BASE SECTION must appear in the DATA DIVISION immediately following the FILE SECTION, if specified, and immediately preceding the WORKING-STORAGE SECTION, if specified. The DATA-BASE SECTION header is followed by one data base reference, consisting of a level indicator (DB), a data base name, and references to data sets within the data base. Each entry is terminated by a period.

Data Base Reference Format

```
DATA-BASE SECTION.

DB [ logical-data-base-name { IN } ] data-base-name.
                            { OF }

01   internal-data-set-name   INVOKE   data-set-name.
```

Syntax Rules:

1. The level indicator DB is used to select a particular data base.
2. The level number 01 is used to select particular data sets from a data base.
3. All subsequent references to a particular invocation of a data set must use the internal-data-set-name.
4. The data base name may be used as a qualifier of data set names. Similarly, a data set name may be used as a qualifier of data names, set names, or names of embedded structures.
5. Data-base-name must be the external name of a data base defined to the system. Logical-data-base-name must be the name of a logical data base declared within the description of the data base.

DATA BASE SECTION

General Rules:

1. The compiler will produce on the COBOL74 listing, in COBOL74 format, all record formats and item, set, and key descriptions.

2. All disjoint data sets to be used in a program must be invoked explicitly in the DATA-BASE SECTION. A structure embedded within another structure is invoked implicitly when its containing structure is invoked. Only structures at the disjoint (outermost) level may be invoked explicitly. Disjoint data sets referred to by embedded subsets must be selected explicitly by the user if referenced.

3. The internal-name options allow the PROCEDURE DIVISION to reference data sets by synonyms.

4. One record area exists for each invoked data set. Using the internal name, a data set may be invoked more than once, resulting in multiple record areas. All associated sets are invoked.

Operations on Data Items

The record area for a data set contains the data items. The data items are generated in a format resembling an 01 working storage entry; thus, all data manipulation statements, including group moves and MOVE CORRESPONDING, can operate on data management items.

If variable format records are used, only those items in the fixed portion of the record are candidates in a MOVE CORRESPONDING statement.

If variable format records are used, a group move at the 01 level moves the maximum size record area regardless of the type of record in the record area.

Operations on Structures

Items such as sets, data sets, and subsets may only be manipulated by special data management COBOL74 verbs.

Qualification

Data base items may be qualified in a manner similar to COBOL74 record items, using the data base name or data set name. Because the data base name is the outermost level of qualification, it must be unique. If the program already contains an identifier which is the same as a data set name, the internal-data-set-name option can be used to give the data set a unique name.

A variable may be declared with the same name as a data base item if the item can be qualified: for instance, not a file-name or 77 level item.

Set, subset, and data set names used to find records may require qualification. If a data set is invoked more than once, using internal naming so that two paths are invoked, then the path must be qualified by the data set which it spans. In addition, embedded structures may be qualified by master structure name.

Selection Expressions

A selection expression is used in FIND, LOCK, and DELETE statements to identify a particular record in a data set.

General Format:

```
{ [data-set-name VIA ] set-selection-expression }
{ [FIRST ]                                      }
{ [LAST  ]                                      }
{ [NEXT  ]  data-set-name                       }
{ [PRIOR ]                                      }
```

Syntax Rules:

1. The VIA phrase need be specified only when the data set has been invoked more than once and is not uniquely qualified by the set-selection-expression.

2. Use of the VIA phrase is allowed with a data set which has been invoked only once, although it is redundant in such a case.

General Rules:

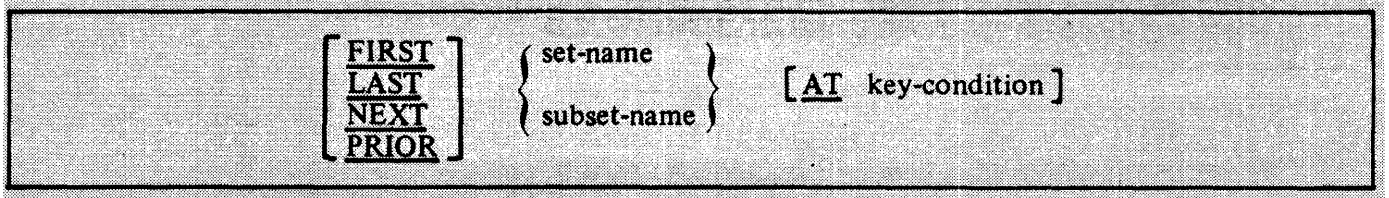
1. The VIA phrase may be used to access a data set by a set or subset invoked with another invocation of the data set.

2. The adjectives FIRST, LAST, NEXT, and PRIOR are used to access a structure sequentially. By stating FIND FIRST followed by FIND NEXT, or FIND LAST followed by FIND PRIOR, all records of the data set are retrieved according to internal ordering of the structure.

3. If neither FIRST, LAST, NEXT, nor PRIOR is specified, the record retrieved is that selected by the existing path of the named structure.

Set Selection Expression

General Format:



General Rules:

1. Set selection expression allows selection of a particular access path. In addition, it also allows specification of a key value to be located.

2. The adjectives FIRST, LAST, NEXT, and PRIOR are used to access a structure sequentially. By stating FIND FIRST followed by FIND NEXT, or FIND LAST followed by FIND PRIOR, all records referenced by the set or subset are retrieved according to internal ordering of the structure.

3. If neither FIRST, LAST, NEXT, PRIOR, nor AT key-condition is specified, the record retrieved is that selected by the existing path of the named structure.

4. The adjectives FIRST, LAST, and PRIOR are not allowed with a key-condition of a set selection expression.

5. If the first operation on a set or data set after an OPEN is a FIND NEXT or LOCK NEXT, the first record is found or locked.

Key Condition

The key-condition option specifies values to be used to locate specific records in a data set referenced by a particular set or subset.

There are two categories of key conditions: simple key conditions and complex key conditions. Each may be enclosed within any number of paired parentheses, in which case the category is not changed.

Simple Key Condition

General Format:

```
                                        { identifier }
data-name relational-operator           { literal    }
```

Syntax Rules:

1. Data-name must be a data-name in the key.
2. Relational-operator is any relational operator as described in the subsection entitled Relation Condition in section 7.

General Rules:

1. If an AT key-condition is used, then the first record satisfying that condition is retrieved. By using NEXT with key-condition in an iterative loop, all records satisfying the key-condition are retrieved. Order of retrieval is not specified.

Complex Key Condition

The rules for forming a complex key condition are the same as those for complex conditions with three exceptions:

1. Each reference to "condition" must be taken to mean "key condition."
2. All items of the key must be in order.
3. The connective used in complex key conditions must be AND.

Generalized Selection Expression

The generalized selection expression feature allows programs to request records from a path, while specifying as much or as little information about the key for that path as is available.

KEY CONDITION

The use of complex keys in a FIND verb can be coded as follows:

```
FIND FTR AT
  GROUP-CODE = GROUPNO AND
  COUNTRY-DISTRICT = DISTNO AND
  BRANCH = BRANCHNO AND
  SEQ = SEQNO.
```

When using the generalized selection expression feature, the FIND verb can be coded as follows:

```
FIND FTR3 AT
GROUP-CODE IS GREATER THAN 99.
```

For further information on the generalized selection expression feature, refer to the B 1000 Systems Data Management System II (DMSII) Reference Manual.

Exception Type

There exists for each COBOL74 program one special register called DMSTATUS which contains exception condition information.

DMSTATUS is set by the system at completion of each data management statement. Procedures for handling exception conditions depend on the type of exception condition encountered. To isolate the exception condition, a number of attributes exist for DMSTATUS.

The attribute DMERROR is TRUE if any exception condition occurred on the most recent data management operation. This test would be:

IF DMSTATUS(DMERROR) statement.

Another attribute yields a boolean value to indicate that a particular category of exception condition occurred. This test has the form:

IF DMSTATUS(category-name) statement.

Each exception condition is represented not only by category name, but also by a numeric value when the attribute DMCATEGORY is interrogated. The numeric value of the error category is contained within the DMSTATUS register and may be referenced, but not modified, by PROCEDURE DIVISION statements. This attribute test has the form:

IF DMSTATUS(DMCATEGORY) = numeric-value statement.

The data item represented by numeric-value can be a literal or an unsigned elementary numeric data item as defined in WORKING-STORAGE.

Exception handling procedures can be declared in any of the above entries for <statement> or by specifying the ON EXCEPTION clause for any of the data management statements.

Example:

```
DELETE <data-set-name>
    ON EXCEPTION
        IF DMSTATUS (category-name-1)      statement
        ELSE  IF DMSTATUS (category-name-2)   statement . . .
```

When the value of DMERROR is true, the ON EXCEPTION branch of the program logic is followed.

NOTE

B 1000 COBOL74 does not allow the ON EXCEPTION... ELSE clause. DMSII programs being converted from COBOL68 to COBOL74 will not follow the expected program logic if they contain this construct.

Table 9-1 provides a list of the available exception category names and numeric values.

EXCEPTION TYPE

Table 9-1. Exception Category Names and Values

Category Name	Numeric Value	Meaning
ABORT	16	Data base activity has been aborted and the data base has been rolled back.
AUDITERROR	15	Attempt to BEGIN-TRANSACTION when already in transaction state.
		Attempt to END-TRANSACTION when not in transaction state.
		UPDATE attempted while not in transaction state.
		CLOSE attempted while in transaction state.
CLOSEERROR	12	Data base not open.
		CLOSE requested while in transaction state.
DATAERROR	4	One or more required item is NULL or a DASDL VERIFY condition was not met.
		A key to a manual subset is NULL.
		Invalid record type.
DEADLOCK	3	Deadlock detected. Note that the system has automatically performed a FREE operation on all records for this program.
		Timeout while attempting to LOCK a record.
		Program already has the record locked in another work area.
DMERROR	Any nonzero number	This attribute is TRUE if any exception has occurred on the most recent data management operation.
DUPLICATES	2	Duplicate not allowed.
FATALERROR	19	Failure in data base software.

Table 9-1. Exception Category Names and Values (Cont)

Category Name	Numeric Value	Meaning
INTEGRITYERROR	20	Key compare option set, KEY and DATA mismatch.
		Automatic set or subset entry missing during STORE or DELETE operation.
		Automatic set or subset points to deleted record. Inconsistent structure control information encountered.
INUSE	14	Attempt to DELETE master when substructure has members.
IOERROR	9	An irrecoverable I-O error occurred while processing a DMSII operation.
KEYCHANGED	6	An attempt was made to store a record when the value of an item, used as a key in a set, was illegally changed (duplicates not allowed).
LIMITERROR	10	Data exceeds size of physical structure.
		Too many entries for a set.
		Attempted to split a fourth-level index sequential coarse table.
NORECORD	13	Current-record pointer not valid for an INSERT operation.
		Current-record pointer not valid for a FIND manual subset-identifier operation.
		Current-record of master not valid for a FIND NEXT or FIND PRIOR on an embedded structure.

EXCEPTION TYPE

Table 9-1. Exception Category Names and Values (Cont)

Category Name	Numeric Value	Meaning
NOTFOUND	1	No record satisfied condition.
		No more records are in NEXT direction.
		No more records are in PRIOR direction.
		Key value in record does not match key of manual subset.
		No current record (it was deleted).
		Record to be referenced is not defined (INSERT, REMOVE).
NOTLOCKED	5	Path state must be valid or created.
OPENERROR	11	Data base not initialized.
		Already opened by this program.
		Run-time description of data base does not match compile-time description.
		Data management operation attempted before data base open.
READONLY	8	An attempt was made to update a data base opened for inquiry only.
		Attempt to update read only items.
SECURITYERROR	17	Program is not authorized to access the information requested.
SYSTEMERROR	7	Attempted to open a data base when six data bases were already active.
VERSIONERROR	18	Run-time description of data base does not match compile-time description.

BEGIN-TRANSACTION

The BEGIN-TRANSACTION statement is used to place a program in transaction state.

General Format:

```
BEGIN-TRANSACTION { AUDIT    } restart-data-set-name
                  { NO-AUDIT }

                                [ ON EXCEPTION statement ]
```

General Rules:

1. Transactions may not be nested. A program may not execute a BEGIN-TRANSACTION statement when it is already in transaction state.

2. If AUDIT is specified, the restart area is preserved for possible restart. If NO-AUDIT is specified, the restart area is not preserved for possible restart.

3. Any exception, except AUDITERROR (the program is already in transaction state), which occurs on execution of this function, results in the program not being placed in transaction state.

Possible Exception Category Names:

ABORT
AUDITERROR
DATAERROR
DEADLOCK
DUPLICATES
FATALERROR
INTEGRITYERROR
IOERROR
KEYCHANGED
LIMITERROR
NOTLOCKED
OPENERROR
READONLY

CLOSE

The CLOSE statement terminates program processing of a data base.

General Format:

```
CLOSE data-base-name [ON EXCEPTION statement]
```

General Rules:

1. CLOSE may be used to close a data base when further access is no longer required. CLOSE is optional, since the system closes any open data base upon termination of the program.
2. An implicit FREE is performed on any records of the data base locked by the program.
3. If the data base is not open, the operation terminates with an exception.
4. When any program closes a data base, a syncpoint occurs.
5. If the data base is closed while in transaction state, an exception is returned and the program remains in transaction state.
6. If an exception occurs the data base is not closed.

Possible Exception Category Names:

ABORT
AUDITERROR
CLOSEERROR
FATALERROR
IOERROR

CREATE

The CREATE statement is used to initialize the record area of a data set in a program.

General Format:

```
CREATE  data-set-name  [ (arithmetic-statement) ]
        [ ON EXCEPTION statement ]
```

General Rules:

1. Execution of the CREATE statement causes the record area of the named data set to be initialized. The following actions take place:

   a. An implicit FREE is performed on the current record of the data set.

   b. The current record path remains unchanged.

   c. The initialized contents of each data item in the record are:

      1) DASDL declared INITIALVALUE if present, or

      2) Default NULL (all bits on).

   d. Normally, a CREATE statement is eventually followed by a STORE which places the new record into the data set. However, if a subsequent STORE statement is not desired, the CREATE can be nullified by a subsequent FREE, FIND, LOCK, DELETE, CREATE, or RECREATE.

   e. An expression is required for variable format records. If the value of the expression does not represent a valid record type an exception occurs.

2. The CREATE operation only initializes a record area. If the record description contains embedded structures, the master record must be stored before storing entries in the embedded structures. When embedded structures are entered in the data base (through a sequence of CREATE and STORE operations) the master need not be re-stored.

Possible Exception Category Names:

DATAERROR
FATALERROR
INTEGRITYERROR
IOERROR
OPENERROR

DELETE

The DELETE statement removes a record from the data base.

General Format:

```
DELETE  data-set-name  [ON EXCEPTION statement ]
```

General Rules:

1. DELETE eliminates a specified record from a data base.
2. An implicit LOCK operation is performed using the data-set-name. Any exception condition associated with LOCK may occur. If the LOCK operation succeeds, the appropriate paths are changed and the record area is loaded.
3. If the record contains a nonempty embedded structure, the record is not deleted, resulting in an exception.
4. If the record can be deleted, it is first removed from all automatic subsets and is then removed from the data set. The current record path becomes vacant, but the data remains unaltered in the record area.
5. A current master must be defined if the record is contained within an embedded set or data set.
6. The system does not remove the record from any manual subsets that may point to the data set member being deleted. The responsibility rests with the application programmer to remove the record from any manual subsets to which it belongs before deleting it from the data base.

Possible Exception Category Names:

AUDITERROR
DEADLOCK
FATALERROR
INTEGRITYERROR
INUSE
IOERROR
NORECORD
NOTFOUND
OPENERROR
READONLY

END-TRANSACTION

The END-TRANSACTION statement takes a program out of transaction state when updates to the data base are being audited.

General Format:

```
END-TRANSACTION { AUDIT    } restart-data-set-name [ SYNC ]
                { NO-AUDIT }
                                [ ON EXCEPTION statement ]
```

General Rules:

1. END-TRANSACTION takes a program out of transaction state.
2. If AUDIT is specified, the contents of the restart area are written to the audit file. If NO-AUDIT is specified, the contents are not written to the audit file. The default is NO-AUDIT.
3. If SYNC is specified, a syncpoint is forced, and the program waits until the syncpoint is complete. For an implicit syncpoint, the program waits at BEGIN-TRANSACTION.
4. For any exception other than ABORT or AUDITERROR, the program remains in transaction state.
5. When the program exits transaction state, an implicit FREE is performed on all records of that data base which the program may have locked.

Possible Exception Category Names:

ABORT
AUDITERROR
DATAERROR
DEADLOCK
DUPLICATES
INTEGRITYERROR
IOERROR
KEYCHANGED
LIMITERROR
NOTLOCKED
OPENERROR
READONLY

FIND

The FIND statement is used to retrieve a particular record from the data base without locking the information.

General Format:

```
FIND { selection-expression } [ON EXCEPTION statement ]
```

General Rules:

1. A FIND selection-expression performs two functions:

   a. Locates the record satisfying the selection expression.
   b. Transfers the data from the data base to the record area so that it may be accessed by the program.

2. Prior to the FIND operation, any previous record in that area is unlocked.

3. A current master must be defined if the record is contained within an embedded data set.

4. If no record satisfying the selection expression can be found, the operation terminates with an exception. In this case, the record area, the current record path, and the current set path, if any, retain original values.

5. If a record is found, it is transferred to the record area and the current record path is altered to refer to the found record. If a set or subset is involved, its path is altered to refer to the found record.

Possible Exception Category Names:

FATALERROR
INTEGRITYERROR
IOERROR
NOTFOUND
NORECORD
OPENERROR

FREE

The FREE statement unlocks the current record.

General Format:

```
FREE  data-set-name  [ ON EXCEPTION statement ]
```

General Rules:

1. FREE unlocks the current record.
2. FREE may follow any data management operation. If the actions are unnecessary or if there is no current record, FREE is ignored.
3. FREE is optional in many situations because CREATE and RECREATE (and sometimes FIND, LOCK, and DELETE) perform an implicit FREE before their actions. In general, an implicit FREE is performed before any operation that establishes a new current record path.
4. The current record path and current record area are unaffected.

Possible Exception Category Names:

FATALERROR
IOERROR
OPENERROR

INSERT

The INSERT statement places a record in a manual subset.

General Format:

```
INSERT data-set-name INTO subset-name [ON EXCEPTION statement]
```

Syntax Rules:

1. Data-set-name must be the declared source of records for the manual subset. Subset-name must be a subset of data-set-name.

2. Subset-name must be a manual subset.

General Rules:

1. The current record path for the data set must be defined. If it is not, the operation is terminated with an exception.

2. If the manual subset is embedded in a data set, the latter must have a current master record defined. If not, the operation is terminated with an exception.

3. If duplicates are not allowed, a duplicate key results in an exception.

Possible Exception Category Names:

AUDITERROR
DATAERROR
DUPLICATES
FATALERROR
IOERROR
LIMITERROR
NORECORD
OPENERROR
READONLY

LOCK

The LOCK statement retrieves a particular record from the data base with a lock which prevents another program from modifying that record.

General Format:

```
LOCK { selection-expression } [ ON EXCEPTION statement ]
```

General Rules:

1. The LOCK statement is identical to the FIND statement with the exception that if a record is found it is locked against a concurrent modification by another program.

2. If the found record is already locked by another user, a deadlock analysis is performed by the system. Normally, the requesting program waits until the record is unlocked. However, if it is determined that a wait would result in a deadlock, or a maximum wait time has been exceeded, all records locked by one deadlocked program are unlocked and its operation terminated with an exception.

3. Since no other program may LOCK the record once it is locked, it is important to FREE the record when it is no longer necessary to keep it locked. This may be accomplished explicitly by a FREE statement or by a subsequent LOCK, FIND, DELETE, CREATE, or RECREATE operation on the same data set. A subsequent STORE operation leaves the record locked.

4. If the data set, set or subset is embedded, a current master record must be defined.

Possible Exception Category Names:

DEADLOCK
FATALERROR
INTEGRITYERROR
IOERROR
NORECORD
NOTFOUND
OPENERROR

OPEN

The OPEN statement initiates the processing of a data base by a program.

General Format:

```
OPEN { { INQUIRY } data-base-name } [ON EXCEPTION statement ]
     { { UPDATE  }                }
```

General Rules:

1. The OPEN statement is used to open the invoked portions of a data base for subsequent access and to specify allowable functions.
2. An OPEN statement must be executed prior to the first access to the data base; otherwise, the program receives an OPENERROR exception.
3. If the data base is open before the OPEN statement is executed, the operation is terminated with an exception.
4. The system attempts to open an existing data base. Appropriate messages are displayed to the system operator if invoked structures and associated structures are not present. The program waits until all invoked portions of the data base are available.
5. If INQUIRY is specified, no update operations are to be performed on the data base. An exception occurs if the following verbs are used:

   BEGIN-TRANSACTION
   DELETE
   END-TRANSACTION
   INSERT
   REMOVE
   STORE

6. Since no changes to the data base can occur, the data management system does not perform an audit if OPEN INQUIRY has been specified by all programs which are running.
7. If UPDATE is specified, the programs may change an existing data base.
8. Immediately after opening the data base, all subsets, sets, and data sets are initialized to retrieve the first record of the subset, set, or data set if a FIND NEXT is executed.

Possible Exception Category Names:

FATALERROR
IOERROR
OPENERROR
SECURITYERROR
SYSTEMERROR
VERSIONERROR

RECREATE

The RECREATE statement sets manual subsets and embedded data sets to NULL; however, all data items remain unaltered.

General Format:

```
RECREATE  data-set-name  [ (arithmetic-expression) ]
                         [ ON EXCEPTION statement ]
```

General Rules:

1. The RECREATE statement is identical to the CREATE statement with the single exception that the record area for the data set is not completely initialized. All data items remain unaltered; however, manual subsets and embedded data sets are unconditionally set to NULL.

Possible Exception Category Names:

DATAERROR
FATALERROR
IOERROR
OPENERROR

REMOVE

The REMOVE statement removes a record from an embedded manual subset.

General Format:

```
REMOVE CURRENT FROM    subset-name    [ ON EXCEPTION statement ]
```

Syntax Rules:

1. Subset-name must be an embedded manual subset.

General Rules:

1. The embedded manual subset must have a defined path; if it does not, the operation is terminated with an exception.

Possible Exception Category Names:

AUDITERROR
FATALERROR
IOERROR
NOTFOUND
NORECORD
OPENERROR
READONLY

STORE

The STORE statement is used to return a modified record to the data set or to place a newly created record into the data set.

General Format:

```
STORE  data-set-name  [ON EXCEPTION statement ]
```

General Rules:

1. The data stored is in the record area of the named data set. Prior to being stored, the data is checked for validity (if specified in DASDL). If any validity checks fail, the STORE operation terminates with an exception.

2. If the last previous operation on the data set was a CREATE or a RECREATE, the data becomes a new record in the data set in a locked state. The current record path is defined to refer to the new record.

3. If the current record path is defined and the current record is locked by this program, the data replaces the current record in the data set. The record remains locked. If the current record path is defined but unlocked (for example, as it would be if the record had been retrieved using FIND), the operation terminates with an exception.

4. The following additional actions are performed, depending on the prior operation:

   a. STORE after CREATE or RECREATE

      1) For each set which spans the data set, the record is tested for validity for insertion (for example, duplicates not allowed).

      2) For each automatic subset (subset with a WHERE condition), the condition is evaluated. The subset is either marked for insertion or it is not marked for insertion.

      3) If the record cannot be inserted into any automatic set or subset for some reason (for example, duplicates not allowed), the operation terminates with an exception. In that case, the record is not inserted into the data set or any set or subset.

      4) If no exceptions exist, then the execution of the STORE statement is successful and all necessary set and subset insertions are made.

   b. STORE after LOCK:

      1) Since this is not a new record, it already exists in all automatic sets.

      2) If items involved in the insertion conditions of any subsets have changed, the conditions must be re-evaluated. For each automatic subset already containing the record, if the condition is no longer met, the record is removed from that subset. For each automatic subset not already containing the record, if the condition is met, the record is inserted into that subset.

STORE

3) If a key of an automatic set or subset is modified such that the record must be moved to a different position in the order of that set or subset, the record is deleted from the set or subset and reinserted in the proper position. It is illegal to modify a key if duplicates are not allowed.

4) The system does not reorder manual subsets. It is the programmer's responsibility to maintain such subsets.

Possible Exception Category Names:

AUDITERROR
DATAERROR
DUPLICATES
FATALERROR
INTEGRITYERROR
IOERROR
KEYCHANGED
LIMITERROR
NORECORD
NOTLOCKED
OPENERROR
READONLY

SECTION 10
DEBUG

GENERAL

The Debug section provides a means by which the user can describe a debugging algorithm, including the conditions under which data items or procedures are to be monitored during the execution of the object program.

The decisions of what to monitor and what information to display on the output device are explicitly in the domain of the user. The COBOL74 debug facility simply provides a convenient access to pertinent information.

LANGUAGE CONCEPTS

The features of the COBOL74 language that support the Debug section are:

1. A compile-time switch called WITH DEBUGGING MODE.
2. An object-time switch.
3. A USE FOR DEBUGGING statement.
4. A special register called DEBUG-ITEM.
5. Debugging lines.

DEBUG-ITEM

The reserved word DEBUG-ITEM is the name for a special register generated automatically. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

A 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 all the debugging statements written in the program.

An Object-Time Switch

An object-time switch (SW9), dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL74 environment. If the switch is ON (SW9 = 1), all the effects of the debugging language written in the source program are permitted. If the switch is OFF (SW9 = 0), all the effects described in the USE FOR DEBUGGING statement are inhibited. The value of SW9 has no effect on debugging lines in the source program. Recompilation of the source program is not required in order to provide or take away this facility.

The object-time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time.

Debugging Lines

A debugging line is any line with the letter D in the indicator area (column 7), of the line. Any debugging line that consists solely of spaces from columns 8 to 72 is considered the same as a blank 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 is considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain the letter D in the indicator area, and character-strings may not be broken across two lines.

The OBJECT-COMPUTER paragraph must precede any debugging line in a program.

ENVIRONMENT DIVISION

To use the debugging features, you must specify the WITH DEBUGGING MODE clause in the ENVIRONMENT DIVISION.

WITH DEBUGGING MODE

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. Refer to Debugging Lines and USE FOR DEBUGGING in this section. If this clause is not specified, all debugging lines and sections are compiled as comment lines.

General Format:

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

General Rules:

1. 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 all debugging lines are compiled.

2. 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 comment lines.

PROCEDURE DIVISION

Within the PROCEDURE DIVISION, procedures and data-names can be selectively monitored through the USE FOR DEBUGGING statement.

USE FOR DEBUGGING

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

General Format:

```
section-name SECTION [ segment-number ] .

                                  ( cd-name-1                                    )
                                  ( [ALL REFERENCES OF]   identifier-1           )
USE FOR DEBUGGING ON              ( file-name-1                                  )
                                  ( procedure-name-1                             )
                                  ( ALL PROCEDURES                               )

[ cd-name-2                                                                      ]
[   [ALL REFERENCES OF]   identifier-2                                           ]
[ , file-name-2                                                                  ] ...
[   procedure-name-2                                                             ]
[   ALL PROCEDURES                                                               ]
```

Syntax Rules:

1. Debugging section(s), if specified, must appear together immediately after the DECLARATIVES header.

2. Except in the USE FOR DEBUGGING statement, there must be no reference to any nondeclarative procedure within the debugging section.

3. Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.

4. Except for the USE FOR DEBUGGING statement, statements appearing within a given debugging section may reference procedure-names defined within a different USE procedure only with a PERFORM statement.

5. P rocedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

6. Any given identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

7. The ALL PROCEDURES phrase can appear only once in a program.

8. When the ALL PROCEDURES phrase is specified, procedure-name-1, procedure-name-2, ... must not be specified in any USE FOR DEBUGGING statement.

9. If the data description entry of the data item referenced by identifier-1, identifier-2, ..., contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1, identifier-2, ..., must be specified without the subscripting or indexing normally required.

10. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

General Rules:

1. In the following general rules all references to cd-name-1, identifier-1, procedure-name-1, and file-name-1 apply equally to cd-name-2, identifier-2, procedure-name-2, and file-name-2, respectively.

2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

3. When file-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. After the execution of any OPEN or CLOSE statement that references file-name-1, and

   b. 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, and

   c. After the execution of any DELETE or START statement that references file-name-1.

4. When procedure-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. Immediately before each execution of the named procedure;

   b. Immediately after the execution of an ALTER statement which references procedure-name-1.

5. The ALL PROCEDURES phrase causes the effects described in General Rule 4 to occur for every procedure-name in the program, except those appearing within a debugging section.

6. When the ALL REFERENCES OF identifier-1 phrase is specified, that debugging section is executed for every statement that explicitly references identifier-1 at each of the following times:

   a. In the case of a WRITE or REWRITE statement, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

PROCEDURE DIVISION

b. In the case of a GO TO statement with a DEPENDING ON phrase, immediately before control is transferred and prior to the execution of any debugging section associated with the procedure-name to which control is to be transferred.

c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification, or evaluation of the contents of the data item referenced by identifier-1.

d. In the case of any other COBOL74 statement, immediately after execution of that statement.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

7. When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:

a. In the case of a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

b. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification, or evaluation of the contents of the data item referenced by identifier-1.

c. Immediately after the execution of any other COBOL74 statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

8. The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement which causes iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

9. When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1,

b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative-statement, and

c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.

10. A reference to file-name-1, identifier-1, procedure-name-1, or cd-name-1 as a qualifier does not constitute reference to that item for the debugging described in the previous general rules.

11. Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides 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 S9999  SIGN IS LEADING SEPARATE
                                     CHARACTER
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-2     PICTURE IS S9999  SIGN IS LEADING SEPARATE
                                     CHARACTER
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-3     PICTURE IS S9999  SIGN IS LEADING SEPARATE
                                     CHARACTER
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-CONTENTS  PICTURE IS X(n).
```

12. Prior to each execution of a debugging section, the contents of the data items referenced by DEBUG-ITEM are space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following General Rules, immediately before control is passed to that debugging section. The contents of any data item not specified in the following general rules remain spaces.

    Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS when the move is treated exactly as an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

13. DEBUG-LINE is a data item that identifies the particular source statement that caused the debugging section to be executed. It contains the line number of the source image, a number which starts from 1 and is incremented by 1 for every source image compiled. This line number is printed on the source listing during compilation.

14. 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 in DEBUG-NAME by the word IN or OF. Subscripts/indices, if any, are not entered into DEBUG-NAME.

15. If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3 respectively as necessary.

PROCEDURE DIVISION

If the data item is subscripted with more than three subscripts or indices, only the occurrence numbers of the first three levels are entered into the DEBUG-ITEM.

16. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following General Rules 17 through 25.

17. If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the first statement of that procedure.

    b. DEBUG-NAME contains the name of that procedure.

    c. DEBUG-CONTENTS contains 'START PROGRAM'.

18. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

19. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

20. If reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT or MERGE statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the SORT or MERGE statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains:

        1) 'SORT INPUT' if the reference to procedure-name-1 is in the INPUT phrase of a SORT statement.

        2) 'SORT OUTPUT' if the reference to procedure-name-1 is in the OUTPUT phrase of a SORT statement.

        3) 'MERGE OUTPUT' if the reference to procedure-name-1 is in the OUTPUT phrase of a MERGE statement.

21. If the transfer of control from the control mechanism associated with a PERFORM statement caused the debugging section associated with procedure-name-1 to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the PERFORM statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'PERFORM LOOP'.

22. If procedure-name-1 is a USE procedure that is to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'USE PROCEDURE'.

23. If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the previous statement.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains 'FALL THROUGH'.

24. If references to file-name-1, cd-name-1 cause the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references file-name-1, cd-name-1.

    b. DEBUG-NAME contains the name of file-name-1, cd-name-1.

    c. For READ, DEBUG-CONTENTS contains the entire record read.

    d. For all other references to file-name-1, DEBUG-CONTENTS contains spaces.

    e. For any reference to cd-name-1, DEBUG-CONTENTS contains the contents of the area associated with the cd-name.

25. If a reference to identifier-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references identifier-1.

    b. DEBUG-NAME contains the name of identifier-1, and

    c. DEBUG-CONTENTS contains the contents of the data item referenced by identifier-1 at the time that control passes to the debugging section (refer to General Rules 6 and 7).

Example:

```
000100   IDENTIFICATION DIVISION.
000200   PROGRAM-ID.  DEBUGTEST.
000300 *
000400 *     THIS PROGRAM IS AN EXAMPLE OF THE COBOL74 DEBUGGING AIDS.
000500 *     NOTE THAT THE "WITH DEBUGGING MODE" STATEMENT IN THE SOURCE
000600 *     COMPUTER PARAGRAPH INSTRUCTS THE COMPILER THAT DEBUGGING
000700 *     STATEMENTS ARE TO BE COMPILED INTO THE OBJECT CODE. SETTING
000800 *     SW9=1 AT EXECUTION TIME ENABLES THE EFFECTS DESCRIBED IN
000810 *     THE "USE FOR DEBUGGING" STATEMENT.
000900 *
001000   ENVIRONMENT DIVISION.
001100   CONFIGURATION SECTION.
001200   SOURCE-COMPUTER.  B1985 WITH DEBUGGING MODE.
001300   OBJECT-COMPUTER.  B1985.
001400   INPUT-OUTPUT SECTION.
001500   FILE-CONTROL.
001600       SELECT PRINTFILE ASSIGN TO PRINTER.
001700 D     SELECT DBGFILE ASSIGN TO PRINTER.
001800   DATA DIVISION.
001900   FILE SECTION.
002000   FD  PRINTFILE.
002100   01  PRINT-REC                       PIC X(132).
002200 D FD  DBGFILE.
002300 D 01  DEBUG-REC                       PIC X(132).
002400   WORKING-STORAGE SECTION.
002500   77  ALFA                            PIC 9 VALUE 4.
002600   77  BRAVEO                          PIC 99 VALUE 16.
002700   77  CHARLIE                         PIC 99.
002800 D 77  FLAG                            PIC 9.
002900 D     88  NEED-IT                     VALUE ZERO.
003000 D     88  FILE-OPEN                   VALUE 1.
003100   01  BARBARA.
003200       03  CHERYL.
003300           05  DIANNE                  PIC 99 VALUE 16.
003400   77  ELVA                            PIC 9(18).
003500   77  FRAN     PIC X(32) VALUE "TEST PROGRAM FOR COBOL 74 MANUAL".
003600   77  GOLDIE                          PIC 9 VALUE 1.
003700   01  HEIDI.
003800       03  ILENE                       PIC 9(8) OCCURS 5 TIMES.
003900       03  JANIS                       PIC 9.
004000 *
004100   PROCEDURE DIVISION.
004200   DECLARATIVES.
004300   D-BUG SECTION.
004400       USE DEBUGGING ON
004500           ALL REFERENCES OF CHARLIE
004600           ALL REFERENCES OF BRAVEO
004700           ALL PROCEDURES.
004800 * RECORD DESCRIPTION FOR DEBUG-ITEM.
004900 *
```

```
005000 * 01  DEBUG-ITEM.
005100 *     02  DEBUG-LINE                PIC X(6).
005200 *     02  FILLER                    PIC X VALUE SPACE.
005300 *     02  DEBUG-NAME                PIC X(30).
005400 *     02  FILLER                    PIC X VALUE SPACE.
005500 *     02  DEBUG-SUB-1               PIC S9(4) SIGN IS LEADING
005600 *                                              SEPARATE CHARACTER.
005700 *     02  FILLER                    PIC X VALUE SPACE.
005800 *     02  DEBUG-SUB-2               PIC S9(4) SIGN IS LEADING
005900 *                                              SEPARATE CHARACTER.
006000 *     02  FILLER                    PIC X VALUE SPACE.
006100 *     02  DEBUG-SUB-3               PIC S9(4) SIGN IS LEADING
006200 *                                              SEPARATE CHARACTER.
006300 *     02  FILLER                    PIC X VALUE SPACE.
006400 *     02  DEBUG-CONTENTS            PIC X(n).
006500 *
006600   DEBUG-OUT.
006700       IF  NEED-IT
006800           OPEN OUTPUT DBGFILE
006900           MOVE 1 TO FLAG.
007000       WRITE DEBUG-REC FROM DEBUG-ITEM.
007100   END DECLARATIVES.
007200   WORK SECTION.
007300   START-IT.
007400       SUBTRACT ALFA FROM BRAVEO GIVING CHARLIE.
007500       DIVIDE BRAVEO BY CHARLIE GIVING ALFA.
007600       SUBTRACT ALFA FROM BRAVEO GIVING CHARLIE.
007700   FOLLOW-UP.
007800       SUBTRACT 4 FROM ANN.
007900       DIVIDE ANN INTO DIANNE GIVING JANIS.
008000       ACCEPT ILENE (GOLDIE) FROM TIME.
008100       OPEN OUTPUT PRINTFILE.
008200       WRITE PRINT-REC FROM FRAN.
008300       WRITE PRINT-REC FROM ILENE (GOLDIE).
008400       WRITE PRINT-REC FROM JANIS.
008500   CLOSE-SHOP.
008600       CLOSE PRINTFILE RELEASE.
008650 D     IF FILE-OPEN
008700 D         CLOSE DBGFILE RELEASE.
008800       STOP RUN.
```

The output from the file DBGFILE follows.

```
000074 WORK                                    START PROGRAM
000074 START-IT                                FALL THROUGH
000074 BRAVEO                                  16
000074 CHARLIE                                 12
000075 BRAVEO                                  16
000075 CHARLIE                                 12
000076 BRAVEO                                  16
000076 CHARLIE                                 15
000076 FOLLOW-UP                               FALL THROUGH
000084 CLOSE-SHOP                              FALL THROUGH
```

DIAGNOSTICS

Debugging and Diagnostic Facilities.

The following compile-time facilities are available:

1. Syntax error messages are printed before the line in error prior to the PROCEDURE DIVISION and after the line in error thereafter. A pointer indicates the location of the possible error.

2. Warning messages are optionally printed before the line in error prior to the PROCEDURE DIVISION and after the line in error thereafter. A pointer indicates the location of the possible error for some of these messages.

3. Various informational messages are printed.

4. No code is generated if syntax errors are present.

All error messages are specific and are sufficient to determine the cause of the error.

Compiler Limits

The maximum compiler limits are:

16,777,215
Source images.

16,383
Data-names – includes all data-names, one for each FD, SD, and CD, and one for each "section" in the DATA DIVISION (for example, WORKING-STORAGE SECTION).

16,383
Procedure-names.

1,022
Pseudo data dictionary entries – one per FD, SD, or CD, one per 01 level in the WORKING-STORAGE, LINKAGE, or COMMUNICATIONS SECTION, one per group of contiguous 77 items in the WORKING-STORAGE SECTION, one per 77 entry in the LINKAGE SECTION, and one for DEBUG.

128
IPC parameters.

16,383
Referenced field length in digits. All 01 levels in the FILE SECTION are referenced.

16,383
Referenced field length in digits of any dimension of a subscripted variable.

2,097,151
Unreferenced field length in digits.

1,048,575
Displacement in digits of a referenced field within an unreferenced field.

1,048,575
OCCURS value – maximum for a single OCCURS value and for total accumulated OCCURS values.

126
Code segments.

16,383
Unique PICTURE strings.

4095
Terminal paragraphs of a PERFORM range – PERFORM A THRU C and PERFORM B THRU C count as one terminal paragraph.

98
Alternate keys allowed for Indexed Files.

COMPILER LIMITS

5
Index-names in the INDEXED BY clause of a CD for OUTPUT.

48
Levels of subscripting.

45
Nested IF statements.

25
Default perform stack entries.

5
Index-names in the INDEXED BY clause of a CD for OUTPUT.

SECTION 11
COBOL74 COMPILER CONTROL

GENERAL

The COBOL74 compiler, in conjunction with the Master Control Program (MCP), allows for various types of actions during compilation which are explained in the text that follows.

INPUT

The compiler uses one or more source input files to create a complete, updated COBOL74 program. Punched cards, disk, diskpack, or magnetic tape can be specified as source language input media for a single input file or for a master file. If two input files are used, the compiler merges them on a sequence number basis.

Records of input files are 80 bytes in length. Default blocking is used for all input files. The format of each 80-byte record is defined in the COBOL74.

Library Files

COBOL74 source text which is common to several programs may be included in a program by use of a COBOL74 COPY statement. A library file is included in the logical program to be compiled but not in a new source language file if one is created.

A library file may not be label equated.

OUTPUT

The following paragraphs discuss the output of the COBOL74 compiler.

New Source Language Files

An updated master symbolic output file, acceptable as input to the COBOL74 compiler, may be created by use of the Compiler Control Image (CCI) option NEW.

Output Listings

Line printer listings provided by this compiler include:

Diagnostic messages
Input source language (may be inhibited)
Indication of inserted, replaced, or deleted lines
Generated object code (upon request)
Summary information
Cross reference listing (upon request)

Generated Code

The object code generated is written to disk.

COMPILATION SOURCE FILE

Compilation and control of a COBOL74 source language program is achieved by presenting a compilation source file to the MCP. This source file is illustrated in figure 11-1. The following paragraphs define the elements of this file.

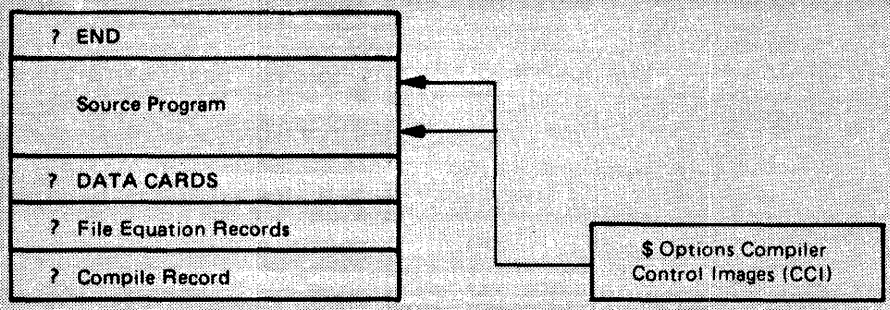


Figure 11-1. Compilation Control File

? COMPILE Record

The first input control record instructs the MCP to call the COBOL74 compiler and to compile the indicated program-name.

General Format:

```
? { CO      }  <program-name>  WITH  COBOL74 [ { LI      } ]
  { COMPILE }                                [ { LIBRARY } ]
                                             [   SYNTAX  ]
```

If the program is to be compiled from the ODT, delete the question mark (?).

The LI or LIBRARY option instructs the MCP to compile program-name and place the resultant object code into the Disk Directory.

The SYNTAX option instructs the MCP to compile the program-name for a syntax check only.

The SAVE option instructs the MCP to compile program-name, place the resultant object code into the Disk Directory, and then execute the object code.

If neither LI, LIBRARY, SYNTAX, nor SAVE is specified, the MCP compiles and then executes program-name.

Label Equation Records

The internal file-names and external file-ids of the COBOL74 compiler are used for label equation as follows:

Internal File-name	External File-name	Function
CARD	CARD	Source or update input. Default device is card reader.
SOURCE	SOURCE	Source program input. Default device is disk.
NEWSOURCE	NEWSOURCE	Updated source program output. Default device is disk.
LINE	LINE	Printed output listing. Default device is printer.
CODE	PROGRAM-NAME	Object code file.
ERRORS	ERRORS	Printed diagnostic listing. Default device is printer.

For example, to change the internal file-name on a new source language file, the label equation record used is:

```
?FILE NEWSOURCE NAME <file-id>
```

Source Program

The language acceptable as input to this compiler is based on the Burroughs COBOL74 as described in this manual.

Increasing Program Code File Sizes

Large code files may cause the COBOL74 compiler to terminate because the allocated space was exceeded for the file CODE. On the B 1000 system, all code files must be contained in one disk area, however the BLOCKS.PER.AREA file attribute of the CODE file can be increased in the following way:

```
COMPILE <program-name> COBOL74 LIBRARY;
FILE CODE BLOCKS.PER.AREA = <number>;
```

Compiler Control Images

The Compiler Control Image (CCI) is used to control certain options which are available during the compilation process. A dollar sign ($) in position 7 of the record indicates a CCI.

Any number of CCIs may be used. With the few exceptions noted separately, options may appear anywhere in the source files and become active or inactive at that point. When CCIs appear as part of a LIBRARY file, the CCI LIBRDOLLAR must be set in the primary source file.

A boolean option may be SET on, RESET off, or set to the result of a boolean expression. For example, using boolean expressions, an option status may be set according to that of other boolean options and the operations AND, OR, EQV, and NOT may be used.

Temporary boolean variables may also be created and used during compilation.

A stack is associated with each boolean option and user-declared variable. Whenever a boolean option or user-declared variable is SET, RESET, or set to an expression, the old value is pushed into a stack and may be retrieved for later use.

CLEAR and POP are also used to control the value of booleans.

Boolean Expressions and User Defined Options

Syntax:

```
DECLARE user-defined-option-1 [user-defined-option-2] . . .

option-1 = [NOT] option-2 [ { AND } [NOT] option-3 ] . . .
                           { OR  }
                           { EQV }
```

Semantics:

1. DECLARE allows the definition of user-defined-options. The option name may be any name that is not already an option. A user-defined-option is a boolean and the default value is FALSE (disabled).

2. Boolean expressions are evaluated from left to right, with all operators having equal precedence.

3. The setting of option-1 is the result of the evaluation of the boolean expression to the right of the equal sign.

CCI Options

The following CCI options are allowed:

SET option-1
    Sets option-1 ON.

option-1
    Sets option-1 ON.

RESET option-1 SKLN
    Sets option-1 OFF.

POP option-1
    Sets option-1 to a previously stacked value.

option-1 = option-2
Sets option-1 to the same value as option-2.

option-1 = NOT option-2
Sets option-1 to the inverse of the value of option-2.

CLEAR
Resets all booleans except MERGE, and if the NEW option is ON, it resets it only if no records have been written to the new source file. All previously stacked settings are discarded.

Normal Boolean Options

CODE
Lists the object code following each line of source code from point of insertion.

DELETE
Causes the compiler to ignore source language images and CCIs from the secondary merged (MERGE) input (disk) until disabled. Images are ignored for compilation purposes and are not made part of a new output SOURCE file.

DEBUG ERRMESS
Causes the compiler to print all the error messages it is capable of producing.

DEBUG TIME
Causes the compiler to print a summary of the time spent during each phase of compilation.

DOUBLE
Causes the output listing to be printed in a double-spaced format.

LIBRDOLLAR
Allows a CCI which occurs as part of a LIBRARY file to be processed. RESET CCIs within a LIBRARY file are ignored.

LIST
Creates an output listing of the source image input during the final phase of compilation with error and warning messages inserted where required. This option is SET on by default and must be RESET if the listing is not wanted.

LIST1
Causes the compiler to print a listing of source images as soon as they are read, and print error messages as the errors are detected.

LISTDELETED
When the LIST option is set, it causes a listing of deleted (DELETE) images.

LISTDOLLAR
When the LIST option is set, it causes a listing of Compiler Control Images. This option is set ON by default.

LISTOMITTED
When the LIST option is set, it causes a listing of omitted (OMIT) images.

LISTP
Causes the compiler to print a listing of images from the primary (card) source file as they are read.

MAP
Causes the compiler to include relative addresses of allocated variables in the program listing (the CODE option includes this too). This option is set ON by default.

MERGE
Causes a secondary (usually disk) file to be opened by the compiler and merged with the primary (card) file. Once SET, this option cannot be RESET, POP'ed, SET again, or SET to an expression. The CLEAR option does not affect this option.

NEW
Creates a new output source file with changes, if any, entered through use of the MERGE and DELETE options. Compiler Control Images are not included in the new file unless marked with an additional $ sign in column 8. Once SET, this option may not be RESET, POP'ed, SET again, or SET to a boolean expression. The CLEAR option may reset the NEW option but only if no records have yet been written.

NEWID
Causes the nonnumeric literal associated with this option to be inserted in columns 73-80 of the source language image for program listing and in the new source file, if any. When this option is RESET the contents of columns 73-80 of the input source file are left in the corresponding columns of the listing and new source.

Nonnumeric literal
Causes the compiler to store the literal information for use with the NEWID option. The default value for this option is all blanks.

OMIT
Causes the compiler to ignore source language images and Compiler Control Images from the secondary merged (MERGE) input (disk) until disabled. Images are ignored for compilation purposes but they are included in a new output SOURCE file, if one is being produced.

PAGE
If a list is being produced, causes the compiler to skip to the top of the next page. This option is reset automatically as soon as it is used.

SEQCHECK
Causes the compiler to check for sequence errors and print a warning message for each sequence error detected.

SEQUENCE
Causes the compiler to assign new sequence numbers to the source language images accepted for compilation or inclusion in a new source file. Sequence numbers begin with the value of the sequence base integer associated with this option. If the sequence base integer is not specified the sequence numbering begins with 000010. Sequence numbers are incremented by the value of the sequence increment integer associated with this option. If the sequence increment integer is not specified sequence numbers are incremented by 10.

integer
Associated with the SEQUENCE option, the integer is assigned to the sequence base.

+integer
Associated with the SEQUENCE option, the integer is assigned to the sequence increment.

STATISTICS
Causes the compiler to print the known statistics of compilation normally included in a listing. Needed when LIST is RESET.

SUMMARY
Causes the compiler to print a summary of the information about the compilation normally included as part of a listing. Needed when LIST is RESET.

WARNFATAL
Causes the compiler to treat warning messages as syntax errors.

WARNSUPR
Inhibits the compiler from printing warning messages.

XREF
Causes the compiler to print a cross-reference listing for the source language accepted for compilation by line number. XREF, if used, must be the first card image in the file CARD.

XSEQ
When cross-reference output is generated (refer to XREF), XSEQ changes the format so that sequence numbers, rather than line numbers are printed.

Miscellaneous Compiler Control Options

FEDLEVEL = integer-1
Causes the compiler to test the COBOL74 source language code for compliance with United States Federal standards as described by FIPS publication 21-1.

Integer-1 may contain a value from 1 to 5 representing the following levels:

1 Low level COBOL74
2 Low-intermediate level COBOL74
3 High-intermediate level COBOL74
4 High level COBOL74
5 Burroughs extended level COBOL74

When a Federal Level is chosen, COBOL74 codes that exceed those included in that level are identified with warning messages. This option is set to 5 by default.

VOID
Allowed only in primary (card) file images, this option causes a secondary file image bearing the same sequence number to be ignored for compilation purposes and excluded from the output new source file, if any. If integer-1 is specified, subsequent images are also ignored until a secondary (disk) file image is encountered with a sequence number value greater than integer-1.

APPENDIX A
RESERVED WORDS

All reserved words known to the B 1000 COBOL74 compiler are listed in this Appendix, which includes notations (X) of the DIVISION(s) wherein the words are used. Also, the notation (X) indicates that the word is used for DATA MANAGEMENT (DMSII).

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
ACCEPT	–	–	–	X	–
ACCESS	–	X	–	–	–
ACTUAL	–	X	X	–	–
ADD	–	–	–	X	–
ADVANCING	–	–	–	X	–
AFTER	–	–	–	X	–
ALARM	–	–	–	X	–
ALL	–	–	X	X	–
ALPHABETIC	–	–	–	X	–
ALSO	–	X	–	–	–
ALTER	–	–	–	X	–
ALTERNATE	–	X	–	–	–
AND	–	–	–	X	–
ARE	–	–	X	–	–
AREA	–	X	–	–	–
AREAS	–	X	–	–	–
ASCENDING	–	–	X	X	–
ASSIGN	–	X	–	–	–
AT	–	–	X	X	–
ATTRIBUTE	–	–	–	X	–
AUDIT	–	–	–	–	X
AUTHOR	X	–	–	–	–
AVAILABLE	–	–	–	X	–
BEFORE	–	–	–	X	–
BEGIN-TRANSACTION	–	–	–	–	X
BEGINNING	–	–	–	–	X
BLANK	–	–	X	–	–
BLOCK	–	–	X	–	–
BOTTOM	–	–	X	–	–
BREAKOUT	–	–	X	–	–
BY	–	–	X	X	–
CALL	–	–	–	X	–
CANCEL	–	–	–	X	–
CD	–	–	X	–	–
F	–	–	X	–	–
CH	–	–	X	–	–
CHANGE	–	–	–	X	–
CHARACTER	–	–	X	–	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
CHARACTERS	–	X	X	X	–
CLOCK-UNITS	–	X	–	–	–
CLOSE	–	–	–	X	–
COBOL	–	–	–	–	–
CODE	–	–	X	–	–
CODE-SET	–	–	X	–	–
COLLATING	–	X	–	X	–
COLUMN	–	–	X	–	–
COMMA	–	X	–	–	–
COMMANDKEYS	–	–	–	X	–
COMMUNICATION	–	–	X	–	–
COMP	–	–	X	–	–
COMPUTATIONAL	–	–	X	–	–
COMPUTE	–	–	–	X	–
CONFIGURATION	–	X	–	–	–
CONTAINS	–	X	X	–	–
CONTROL	–	–	X	–	–
CONTROLS	–	–	X	–	–
COPY	X	X	X	X	–
CORR	–	–	–	X	–
CORRESPONDING	–	–	–	X	–
COUNT	–	–	X	X	–
CREATE	–	–	–	–	X
CRUNCH	–	–	–	X	–
CURRENCY	–	X	–	–	–
CURRENT	–	–	–	–	X
DATA	–	–	X	X	–
DATA-BASE	–	–	–	–	X
DATE	–	–	X	X	–
DATE-COMPILED	X	–	–	–	–
DATE-WRITTEN	X	–	–	–	–
DAY	–	–	–	X	–
DB	–	–	–	–	X
DE	–	–	X	–	–
DEBUG-CONTENTS	–	–	–	X	–
DEBUG-ITEM	–	–	–	X	–
DEBUG-LINE	–	–	–	X	–
DEBUG-SUB-1	–	–	–	X	–
DEBUG-SUB-2	–	–	–	X	–
DEBUG-SUB-3	–	–	–	X	–
DEBUGGING	–	X	–	X	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
DECIMAL-POINT	–	X	–	–	–
DECLARATIVES	–	–	–	X	–
DELETE	–	–	–	X	–
DELIMITED	–	–	–	X	–
DELIMITER	–	–	–	X	–
DEPENDING	–	–	X	X	–
DESCENDING	–	–	X	X	–
DESTINATION	–	–	X	–	–
DETAIL	–	–	X	–	–
DISABLE	–	–	–	X	–
DISPLAY	–	–	X	X	–
DIVIDE	–	–	–	X	–
DIVISION	X	X	X	X	–
DMCATEGORY	–	–	–	–	X
DMERROR	–	–	–	–	X
DMSTATUS	–	–	–	–	X
DOWN	–	–	–	X	–
DUPLICATES	–	X	–	–	–
DYNAMIC	–	X	–	–	–
EGI	–	–	–	X	–
ELSE	–	–	–	X	–
EMI	–	–	–	X	–
ENABLE	–	–	–	X	–
END	–	X	X	X	–
END-OF-PAGE	–	–	–	X	–
END-TRANSACTION	–	–	–	–	X
ENDING	–	–	–	–	X
ENVIRONMENT	–	X	–	–	–
EOP	–	–	–	X	–
EQUAL	–	–	–	X	–
ERROR	–	–	X	X	–
ESI	–	–	–	X	–
EVERY	–	X	–	–	–
EXCEPTION	–	–	–	X	–
EXIT	–	–	–	X	–
EXTEND	–	–	–	X	–
FALSE	–	–	–	X	–
FD	–	–	X	–	–
FIELD	–	–	–	X	–
FILE	–	X	X	–	–
FILE-CONTROL	–	X	–	–	–
FILLER	–	–	X	–	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
FINAL	–	–	X	–	–
FIND	–	–	–	–	X
FIRST	–	–	X	–	–
FOOTING	–	–	X	–	–
FOR	–	X	X	X	–
FREE	–	–	–	–	X
FROM	–	–	–	X	–
GENERATE	–	–	–	X	–
GIVING	–	–	–	X	–
GO	–	–	–	X	–
GREATER	–	–	–	X	–
GROUP	–	–	X	–	–
HEADING	–	–	X	–	–
HIGH-VALUE	–	–	X	X	–
HIGH-VALUES	–	–	X	X	–
I-O	–	–	–	X	–
I-O-CONTROL	–	X	–	–	–
IDENTIFICATION	X	–	–	–	–
IF	–	–	–	X	–
IN	–	–	–	X	–
INDEX	–	–	X	–	–
INDEXED	–	X	X	–	–
INDICATE	–	–	X	–	–
INITIAL	–	–	X	X	–
INITIALIZE	–	–	–	–	X
INITIATE	–	–	–	X	–
INPUT	–	–	X	X	–
INPUT-OUTPUT	–	X	–	X	–
INQUIRY	–	–	–	–	X
INSERT	–	–	–	–	X
INSPECT	–	–	–	X	–
INSTALLATION	X	–	–	–	–
INTO	–	–	–	X	–
INVALID	–	–	–	X	–
INVOKE	–	–	–	–	X
IS	–	X	X	X	–
JUST	–	–	X	–	–
JUSTIFIED	–	–	X	–	–
KEY	–	X	X	X	–
LABEL	–	–	X	–	–
LAST	–	–	X	–	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
LEADING	–	–	X	X	–
LEFT	–	–	X	–	–
LENGTH	–	–	X	–	–
LESS	–	–	–	X	–
LIMIT	–	–	X	–	–
LIMITS	–	–	X	–	–
LINAGE	–	–	X	–	–
LINAGE-COUNTER	–	–	–	X	–
LINE	–	–	X	X	–
LINE-COUNTER	–	–	–	X	–
LINES	–	–	X	X	–
LINKAGE	–	–	X	–	–
LOCK	–	–	–	X	–
LOW-VALUE	–	–	X	X	–
LOW-VALUES	–	–	X	X	–
MEMORY	–	X	–	–	–
MERGE	–	–	–	X	–
MESSAGE	–	–	X	X	–
MODE	–	X	–	–	–
MODULES	–	X	–	–	–
MOVE	–	–	–	X	–
MULTIPLE	–	X	–	–	–
MULTIPLY	–	–	–	X	–
NATIVE	–	X	–	–	–
NEGATIVE	–	–	–	X	–
NEXT	–	–	X	X	–
NO	–	–	–	X	–
NO-AUDIT	–	–	–	–	X
NONE	–	–	–	–	X
NOT	–	–	–	X	–
NULL	–	–	–	–	X
NUMBER	–	–	X	–	–
NUMERIC	–	–	–	X	–
OBJECT-COMPUTER	–	X	–	–	–
OCCURS	–	–	X	–	–
ODT	–	X	–	–	–
ODT-INPUT-PRESENT	–	–	–	X	–
OF	–	X	X	X	–
OFF	–	X	–	–	–
OFFER	–	–	–	X	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
OFFSET	–	–	–	X	–
OMITTED	–	–	X	–	–
ON	–	X	X	X	–
OPEN	–	–	–	X	–
OPTIONAL	–	X	–	–	–
OR	–	–	–	X	–
ORGANIZATION	–	X	–	–	–
OUTPUT	–	–	X	X	–
OVERFLOW	–	–	–	X	–
PAGE	–	–	X	X	–
PAGE-COUNTER	–	–	–	X	–
PERFORM	–	–	–	X	–
PF	–	–	X	–	–
PH	–	–	X	–	–
PIC	–	–	X	–	–
PICTURE	–	–	X	–	–
PLUS	–	–	X	–	–
POINTER	–	–	–	X	–
POSITION	–	X	–	–	–
POSITIVE	–	–	–	X	–
PRINTING	–	–	–	X	–
PRIOR	–	–	–	–	X
PROCEDURE	–	–	–	X	–
PROCEDURES	–	–	–	X	–
PROCEED	–	–	–	X	–
PROGRAM	–	X	–	X	–
PROGRAM-ID	X	–	–	–	–
PURGE	–	–	–	X	–
QUEUE	–	–	X	–	–
QUOTE	–	–	X	X	–
QUOTES	–	–	X	X	–
RANDOM	–	X	–	–	–
RD	–	–	X	–	–
READ	–	–	–	X	–
READ-OK	–	–	–	X	–
READ-WRITE	–	–	–	X	–
RECEIVE	–	–	–	X	–
RECORD	–	X	X	X	–
RECORDS	–	X	X	–	–
RECREATE	–	–	–	–	X
REDEFINES	–	–	X	–	–
REEL	–	X	–	X	–
REFERENCES	–	–	–	X	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				DMSII
	IDEN	ENVI	DATA	PROC	
RELATIVE	–	X	–	–	–
RELEASE	–	–	–	X	–
REMAINDER	–	–	–	X	–
REMOVAL	–	–	–	X	–
REMOVE	–	–	–	X	–
RENAMES	–	–	X	–	–
REPLACING	–	–	–	X	–
REPORT	–	–	X	–	–
REPORTING	–	–	–	X	–
REPORTS	–	–	X	–	–
RERUN	–	X	–	–	–
RESERVE	–	X	–	–	–
RESET	–	–	X	–	–
RETURN	–	–	–	X	–
REVERSED	–	–	–	X	–
REWIND	–	–	–	X	–
REWRITE	–	–	–	X	–
RF	–	–	X	–	–
RH	–	–	X	–	–
RIBBON	–	–	–	X	–
RIGHT	–	–	X	–	–
ROUNDED	–	–	–	X	–
RUN	–	–	–	X	–
SAME	–	X	–	–	–
SAVE	–	X	–	–	–
SD	–	–	X	–	–
SEARCH	–	–	–	X	–
SECTION	–	X	X	X	–
SECURITY	X	–	–	–	–
SEEK	–	–	–	X	–
SEGMENT	–	–	–	X	–
SEGMENT-LIMIT	–	X	–	–	–
SELECT	–	X	–	–	–
SEND	–	–	–	X	–
SENTENCE	–	–	–	X	–
SEPARATE	–	–	X	–	–
SEQUENCE	–	X	–	X	–
SEQUENTIAL	–	X	–	–	–
SET	–	–	–	X	–
SIGN	–	X	X	–	–
SIZE	–	X	–	X	–
SORT	–	X	–	X	–
SORT-MERGE	–	X	–	–	–
SOURCE	–	–	X	–	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
SOURCE-COMPUTER	–	X	–	–	–
SPACE	–	–	X	X	–
SPACES	–	–	X	X	–
SPECIAL-NAMES	–	X	–	–	–
STACK	–	X	–	–	–
STANDARD	–	–	X	X	–
STANDARD-1	–	X	–	–	–
START	–	–	–	X	–
STATUS	–	X	–	–	–
STOP	–	–	–	X	–
STORE	–	–	–	–	X
STRING	–	–	–	X	–
SUB-QUEUE-1	–	–	X	–	–
SUB-QUEUE-2	–	–	X	–	–
SUB-QUEUE-3	–	–	X	–	–
SUBTRACT	–	–	–	X	–
SUM	–	–	X	–	–
SUPPRESS	–	–	–	X	–
SYMBOLIC	–	–	X	–	–
SYNC	–	–	X	–	–
SYNCHRONIZED	–	–	X	–	–
SYSTEM	–	–	X	–	–
TABLE	–	–	X	–	–
TAG-KEY	–	–	–	X	–
TAG-SEARCH	–	–	–	X	–
TALLYING	–	–	–	X	–
TAPE	–	X	–	–	–
TERMINAL	–	–	–	X	–
TERMINATE	–	–	–	X	–
TEXT	–	–	X	–	–
THAN	–	–	–	X	–
THROUGH	–	X	X	X	–
THRU	–	X	X	X	–
TIME	–	–	X	X	–
TIMER	–	–	–	X	–
TIMES	–	–	X	X	–
TO	–	X	X	X	–
TODAYS-DATE	–	–	–	X	–
TODAYS-NAME	–	–	–	X	–
TOP	–	–	X	–	–
TRAILING	–	–	X	–	–

RESERVED WORDS	REASON FOR RESERVATIONS				
	DIVISION				
	IDEN	ENVI	DATA	PROC	DMSII
TRACTORS	–	–	X	X	–
TRANSPORT	–	–	–	X	–
TRUE	–	–	–	X	–
TYPE	–	–	X	–	–
UNITS	–	X	–	X	–
UNSTRING	–	–	–	X	–
UNTIL	–	–	–	X	–
UP	–	–	–	X	–
UPDATE	–	–	–	–	X
UPON	–	–	X	X	–
USAGE	–	–	X	–	–
USE	–	–	–	X	–
USING	–	–	–	X	–
VALUE	–	–	X	–	–
VALUES	–	–	X	–	–
VARYING	–	–	–	X	–
VIA	–	–	–	–	X
WAIT	–	–	–	X	–
WHEN	–	–	X	X	–
WHERE	–	–	–	–	X
WITH	–	X	X	X	–
WORDS	–	X	–	–	–
WORKING-STORAGE	–	–	X	–	–
WRITE	–	–	–	X	–
WRITE-OK	–	–	–	X	–
ZERO	–	–	X	X	–
ZEROES	–	–	X	X	–
ZEROS	–	–	X	X	–
ZIPSB	–	–	–	X	–
+	–	–	–	X	–
–	–	–	–	X	–
*	–	–	–	X	–
/	–	–	–	X	–
**	–	–	–	X	–
>	–	–	–	X	–
<	–	–	–	X	–
=	–	–	–	X	–

APPENDIX B
COBOL74 SYNTAX SUMMARY

This appendix contains the composite language skeleton of COBOL74. Shaded items indicate extensions to the American National Standard, 1974 (ANSI-74).

IDENTIFICATION DIVISION

General Format

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

ENVIRONMENT DIVISION

General Format

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

SOURCE-COMPUTER

Format 1:

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

Format 2:

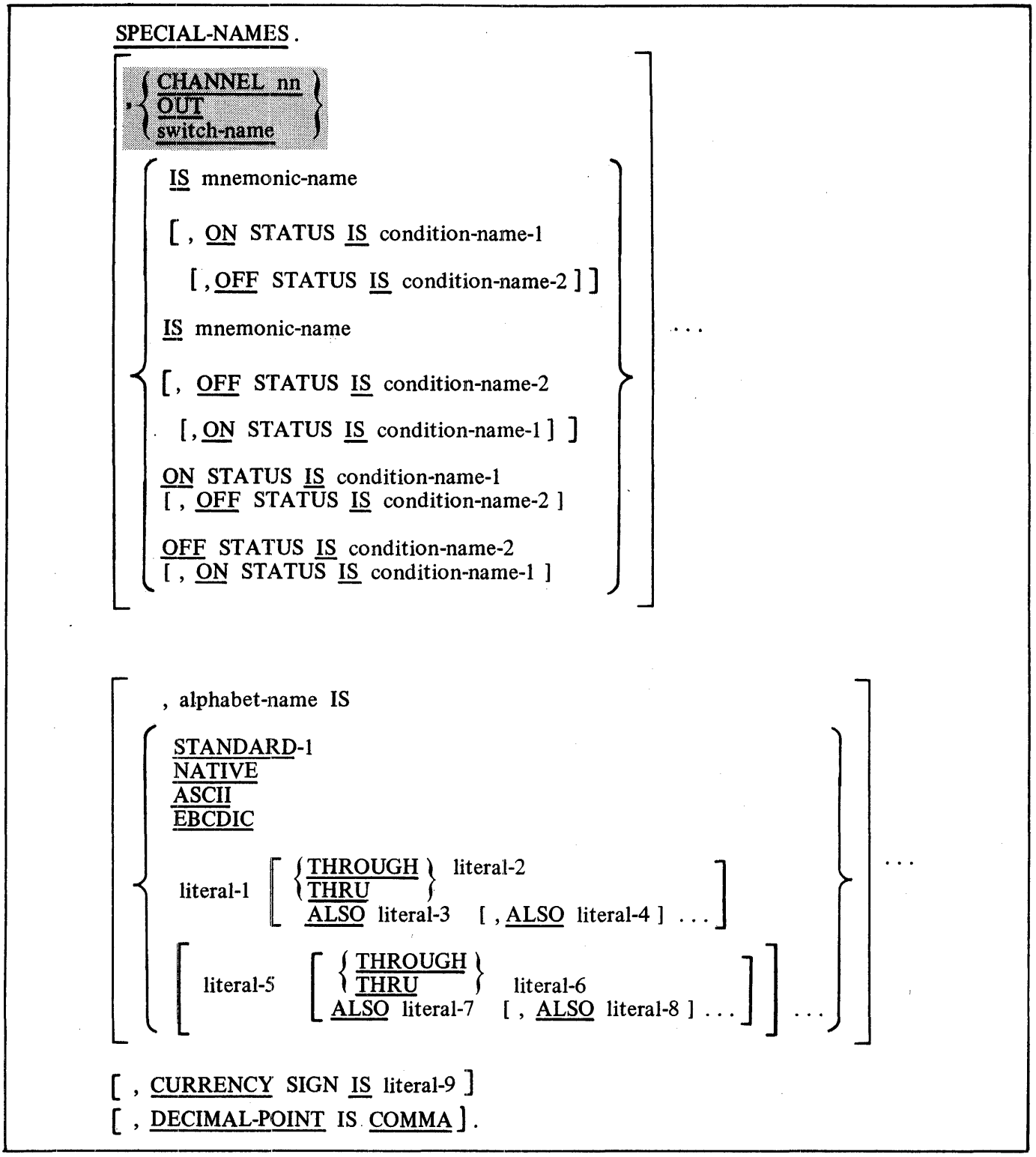
```
SOURCE-COMPUTER.    computer-name  [ WITH DEBUGGING MODE ]
```

OBJECT-COMPUTER

```
OBJECT-COMPUTER.    computer-name  [ , MEMORY SIZE integer  { WORDS      } ]
                                                           { CHARACTERS }
                                                           { MODULES    }
    [ , STACK SIZE IS integer-2 ]
    [ , PROGRAM COLLATING SEQUENCE IS alphabet-name ]
    [ , SEGMENT-LIMIT IS segment-number ] .
```

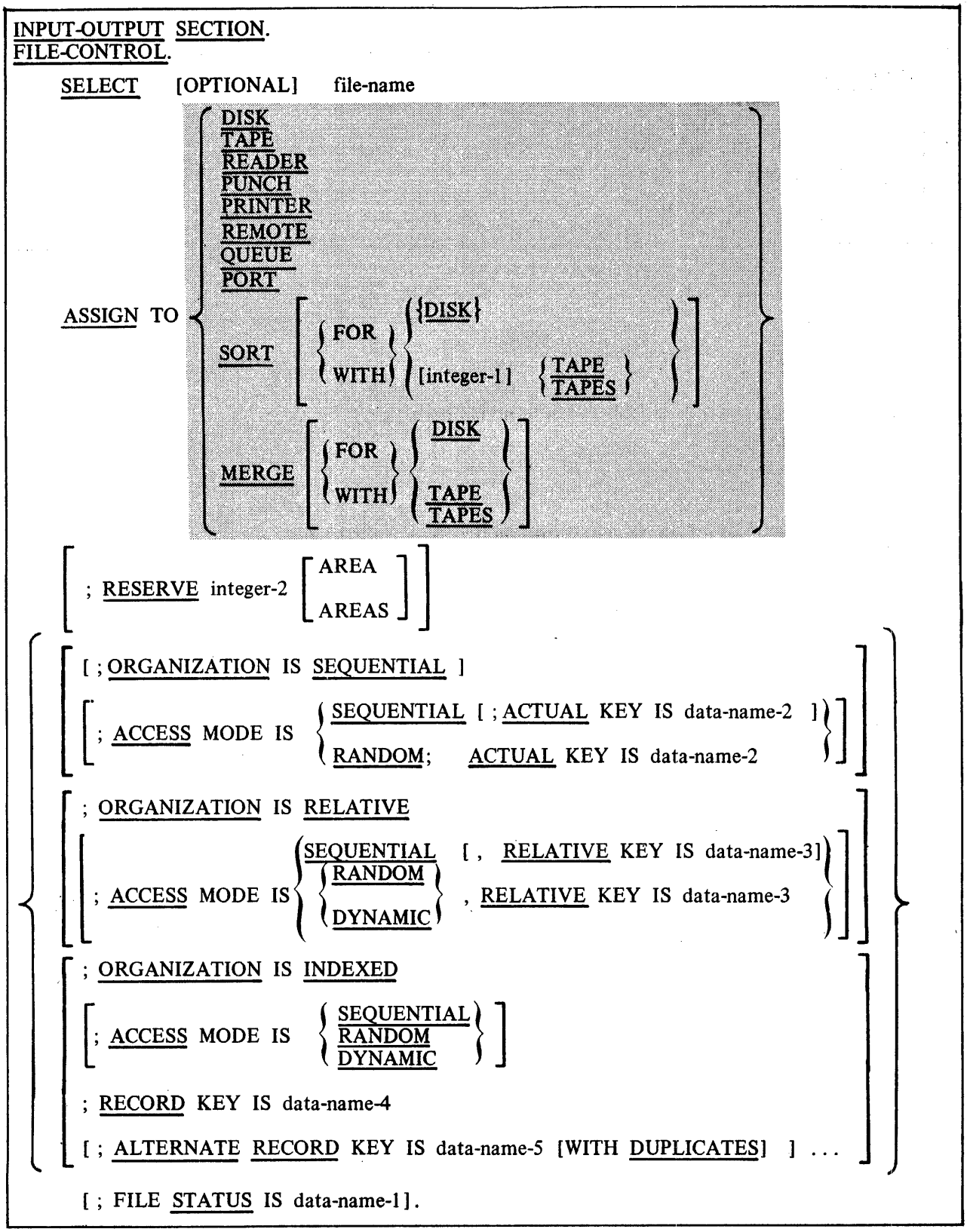
ENVIRONMENT DIVISION

SPECIAL-NAMES

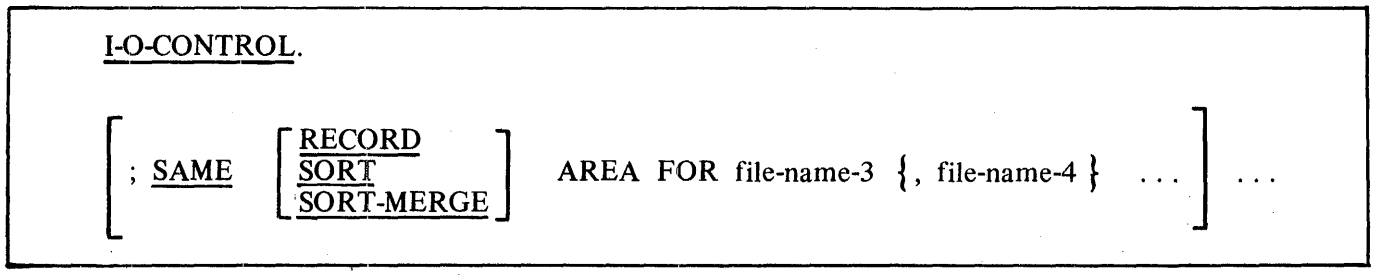


ENVIRONMENT DIVISION

INPUT-OUTPUT SECTION



I-O-CONTROL


I-O-CONTROL.
; SAME
RECORD
SORT
SORT-MERGE
AREA FOR file-name-3 , file-name-4 ...
...

DATA DIVISION

General Format

```
DATA DIVISION.
[ FILE SECTION.                                                          ]
[   [ file-description-entry [record-description-entry] ...             ]
[   [ sort-merge-file-description-entry (record-description-entry) ... ] ... ]

[ WORKING-STORAGE SECTION.                                               ]
[   [ 77-level-description-entry ]                                       ]
[   [ record-description-entry   ] ...                                   ]

[ LINKAGE SECTION.                                                       ]
[   [ 77-level-description-entry ]                                       ]
[   [ record-description-entry   ] ...                                   ]

[ COMMUNICATION SECTION.                                                 ]
[   [ communication-description-entry  [record-description-entry] ... ] ... ]
```

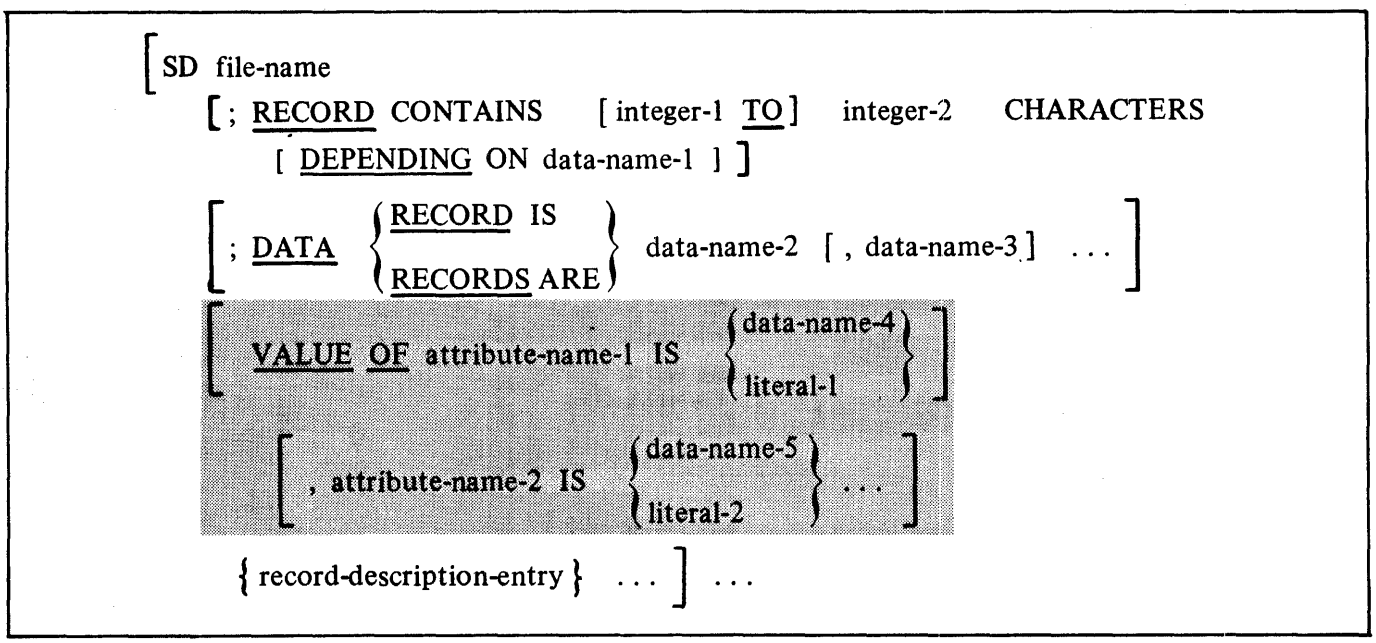
FILE SECTION

FD file-name

```
[FD file-name
    [ ; BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS    } ]
                                                { CHARACTERS }
    [ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
        [ DEPENDING ON data-name-1 ] ]
    [ ; LABEL { RECORD IS  } { STANDARD } ]
              { RECORDS ARE} { OMITTED  }
    [ ; VALUE OF attribute-name-1 IS { data-name-2 }
                                     { literal-1   }
        [ , attribute-name-2 IS { data-name-3 } ] ... ] ...
                                { literal-2   }
    [ ; DATA { RECORD IS  } data-name-4 [, data-name-5] ... ]
             { RECORDS ARE}
    [ ; LINAGE IS { data-name-6 } LINES [, WITH FOOTING AT { data-name-7 } ]
                  { integer-5   }                          { integer-6   }
        [ , LINES AT TOP { data-name-8 } ] [ , LINES AT BOTTOM { data-name-9 } ] ]
                         { integer-7   }                       { integer-8   }
    [ ; CODE-SET IS alphabet-name ].
      {record-description-entry} ... ] ...
```

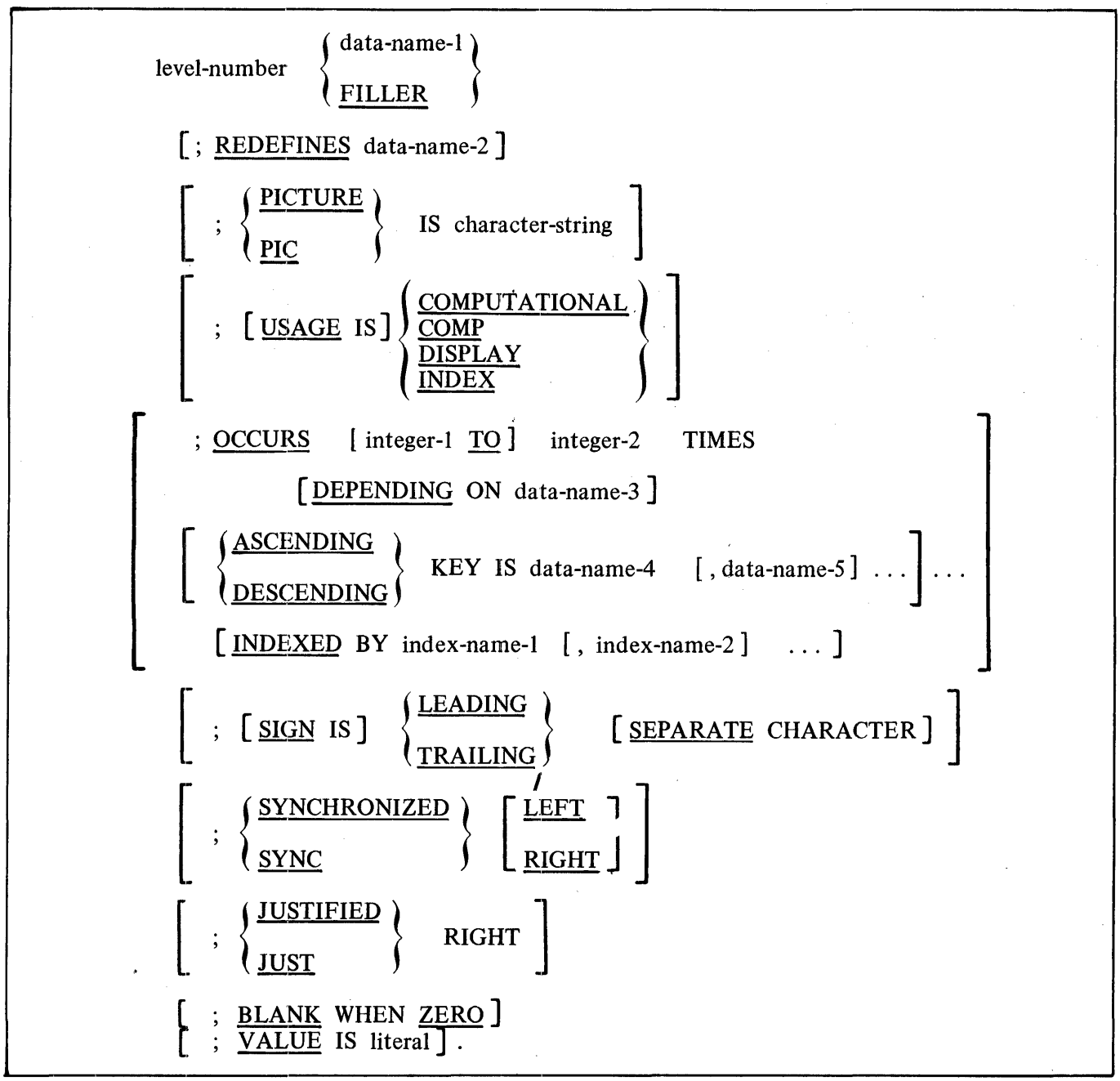
DATA DIVISION

SD file-name



Data Description Entry

Format 1:



DATA DIVISION

Data Description Entry

Format 2:

```
66 data-name-1; RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                     { THRU    }
```

Format 3:

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

[ , literal-3 [ { THROUGH } literal-4 ] ] ... .
              { THRU    }
```

WORKING-STORAGE SECTION

```
WORKING-STORAGE SECTION.
77      data-name-1
        88  condition-name-1
            .
            .
77      data-name-n
01      data-name-2
        02  data-name-3
            .
            .
        66  data-name-m RENAMES data-name-3
01      data-name-4
        02  data-name-5
            03  data-name-n
                88  condition-name-2
```

DATA DIVISION

LINKAGE SECTION

```
LINKAGE SECTION.
77    data-name-1
      88 condition-name-1
            .
            .
77    data-name-n
01    data-name-2
      02 data-name-3
            .
            .
      66 data-name-m RENAMES data-name-3
01    data-name-4
      02 data-name-5
         03   data-name-n
              88   condition-name-2
```

COMMUNICATION SECTION

Format 1:

```
CD cd-name;

                                  [ [ [ ; SYMBOLIC QUEUE IS data-name-1 ]
                                      [ ; SYMBOLIC SUB-QUEUE-1 IS data-name-2 ]
                                      [ ; SYMBOLIC SUB-QUEUE-2 IS data-name-3 ]
                                      [ ; SYMBOLIC SUB-QUEUE-3 IS data-name-4 ]
FOR [ INITIAL ] INPUT                 [ ; MESSAGE DATE IS data-name-5 ]
                                      [ ; MESSAGE TIME IS data-name-6 ]
                                      [ ; SYMBOLIC SOURCE IS data-name-7 ]
                                      [ ; TEXT LENGTH IS data-name-8 ]
                                      [ ; END KEY IS data-name-9 ]
                                      [ ; STATUS KEY IS data-name-10 ]
                                      [ ; MESSAGE COUNT IS data-name-11 ] ]
                                  [ data-name-1, data-name-2, . . . , data-name-11 ] ]
```

Format 2:

```
CD cd-name; FOR OUTPUT
    [ ; DESTINATION COUNT IS data-name-1 ]
    [ ; TEXT LENGTH IS data-name-2 ]
    [ ; STATUS KEY IS data-name-3 ]
    [ ; DESTINATION TABLE OCCURS integer-2 TIMES
        [ ; INDEXED BY index-name-1 [ , index-name-2 ] ... ] ]
    [ ; ERROR KEY IS data-name-4 ]
    [ ; SYMBOLIC DESTINATION IS data-name-5 ] .
```

PROCEDURE DIVISION

General Format

```
PROCEDURE DIVISION [USING data-name-1 [, data-name-2] ... ] .
```

Format 1:

```
[DECLARATIVES.

{section-name SECTION [segment-number]. declarative-sentence

[paragraph-name. [sentence] ... ] ... } ...

 END DECLARATIVES.]

{section-name SECTION [segment-number].

[paragraph-name. [sentence] ... ] ... } ...
```

Format 2:

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

DECLARATIVES

```
section-name SECTION [ segment-number ] .

                                  ( cd-name-1                                  )
                                  (  [ALL REFERENCES OF]   identifier-1        )
USE FOR DEBUGGING ON              ( file-name-1                                )
                                  ( procedure-name-1                           )
                                  ( ALL PROCEDURES                             )

[   cd-name-2                                                                  ]
[    [ALL REFERENCES OF]   identifier-2                                        ]
[, file-name-2                                                                 ]  ... .
[   procedure-name-2                                                           ]
[   ALL PROCEDURES                                                             ]
```

VERBS

ACCEPT

Format 1:

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

Format 2:

```
ACCEPT identifier FROM { DATE
                       { DAY
                       { TIME
                       { TIMER
                       { TODAYS-DATE
                       { TODAYS-NAME
```

Format 3:

```
ACCEPT  cd-name  MESSAGE COUNT
```

ADD

Format 1:

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

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

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

Format 2:

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

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

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

Format 3:

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

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

ALTER

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

CALL

Format 1:

```
CALL { identifier-1 } [ USING data-name-1 [ , data-name-2 ] . . . ]
     { literal-1    }
[ ; ON OVERFLOW imperative-statement ]
```

Format 2:

```
CALL SYSTEM DUMP
```

Format 3:

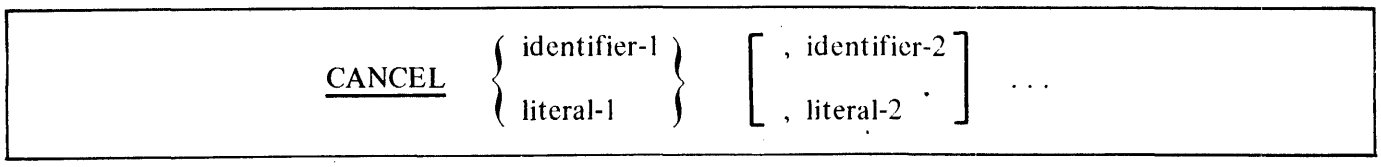
```
CALL SYSTEM ZIPSB USING { identifier-2 }
                        { literal-2    }
```

Format 4:

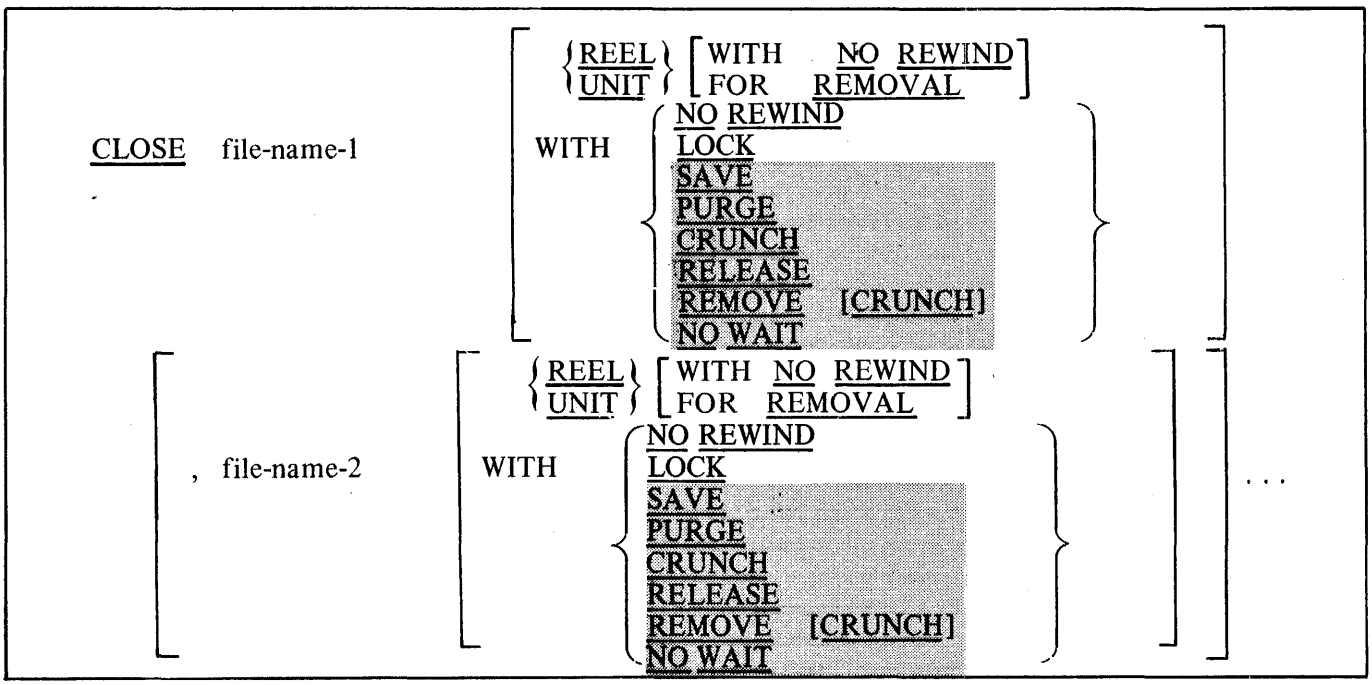
```
CALL SYSTEM WFL USING { identifier-3 }
                      { literal-3    }
```

CANCEL, CLOSE, COMPUTE

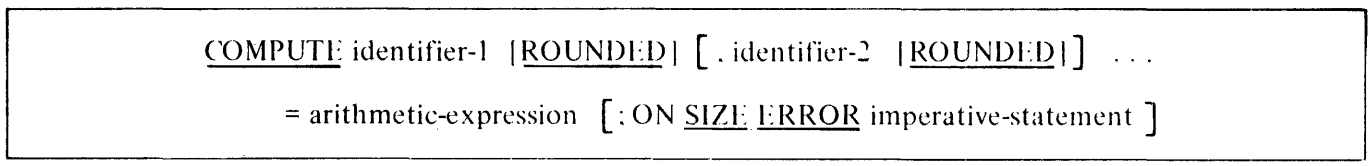
CANCEL



CLOSE



COMPUTE



COPY

```
COPY { { file-id      [ { OF } multi-file-id ] ... [ ON pack-id ] }
     { { literal-3      { IN }                                   }

                [ FROM integer-1 [ TO integer-2 ] ]

[ REPLACING { , { ==pseudo-text-1== } } BY { ==pseudo-text-2== } } ... ]
            {   { identifier-1      }     { identifier-2      } }
            {   { literal-1         }     { literal-2         } }
            {   { word-1            }     { word-2            } }
```

DELETE

```
DELETE file-name RECORD [ ; INVALID KEY imperative-statement ]
```

DISABLE

```
DISABLE { INPUT [ TERMINAL ] } cd-name WITH KEY { identifier-1 } .
        { OUTPUT             }                  { literal-1    }
```

DISPLAY

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

[ UPON mnemonic-name ]
```

DIVIDE, ENABLE

DIVIDE

Format 1:

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

Format 2:

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

Format 3:

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

ENABLE

```
ENABLE { INPUT [TERMINAL] } cd-name WITH KEY { identifier-1 }
       { OUTPUT           }                  { literal-1    }
```

EXIT

```
EXIT.
```

EXIT PROGRAM

```
EXIT PROGRAM.
```

GO TO

Format 1:

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

Format 2:

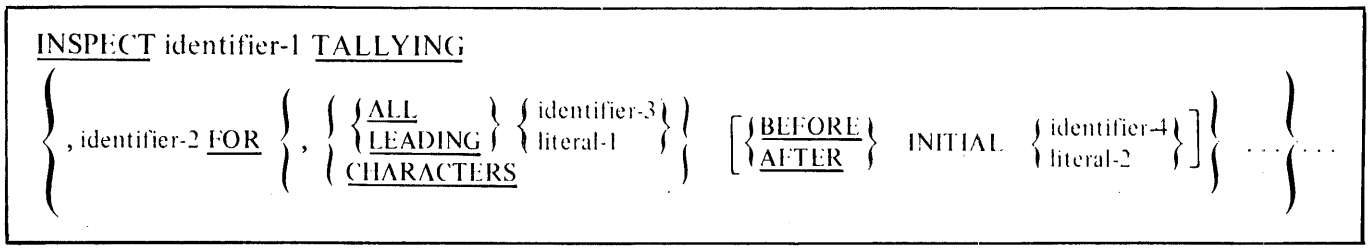
```
GO TO procedure-name-1 [, procedure-name-2] . . . , procedure-name-n
     DEPENDING ON identifier
```

IF

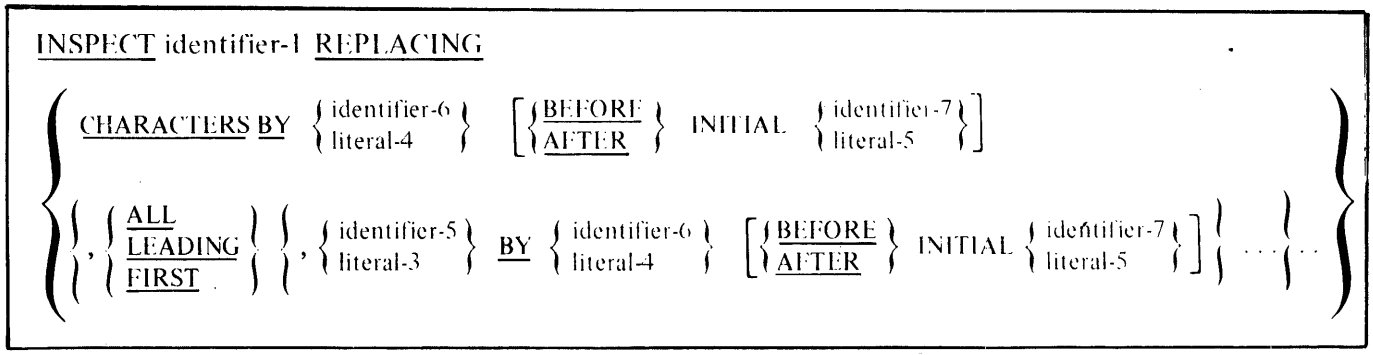
```
IF condition; { statement-1      } { ; ELSE statement-2      }
              { NEXT SENTENCE    } { ; ELSE NEXT SENTENCE    }
```

INSPECT

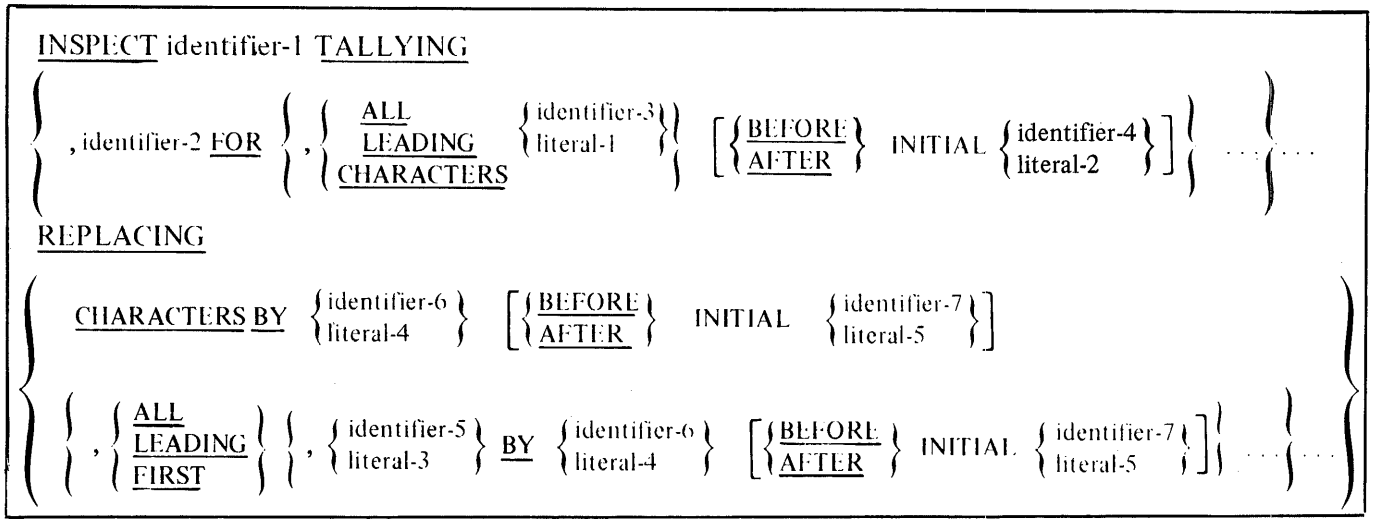
Format 1:



Format 2:



Format 3:



MERGE

```
MERGE
    file-name-1                [ { PURGE } ON ERROR ]
                               [ { END   }          ]

        ON { ASCENDING  } KEY data-name-1 [ , data-name-2 ] ...
           { DESCENDING }

     [ ON { ASCENDING  } KEY data-name-3 [ , data-name-4 ] ... ] ...
          { DESCENDING }

     [ COLLATING SEQUENCE IS alphabet-name ]

USING
    file-name-2 [ { PURCH   } ] , file-name-3 [ { PURGE   } ] [ , file-name-4 [ { PURGE   } ] ] ...
                  { RELEASE }                 { RELEASE }                     { RELEASE }

    { OUTPUT PROCEDURE IS procedure-name-1 [ { THROUGH } procedure-name-2 ] }
    {                                      [ { THRU    }                  ] }
    { GIVING file-name-5 [ { SAVE       } ]                                }
    {                    [ { RELEASE    } ]                                }
    {                    [ { NO REWIND  } ]                                }
    {                    [ { CRUNCH     } ]                                }
```

MOVE

Format 1:

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

Format 2:

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

MULTIPLY, OPEN

MULTIPLY

Format 1:

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

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

Format 2:

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

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

OPEN

```
       { INPUT          file-name-1   [ WITH LOCK [ACCESS] ]                       }
       {                              [ REVERSED           ]                       }
       {                              [ WITH NO REWIND     ]                       }
       {                                                                           }
       {              [ , file-name-2 [ WITH LOCK [ACCESS] ] ] ...                 }
       {              [               [ REVERSED           ] ]                     }
       {              [               [ WITH NO REWIND     ] ]                     }
       {                                                                           }
OPEN   { OUTPUT         file-name-3   [WITH NO REWIND]                             } ...
       {              [ , file-name-4 [WITH NO REWIND]  ] ...                      }
       { I-O            file-name-5   [WITH LOCK [ACCESS] ]                        }
       {              [ , file-name-6 [WITH LOCK [ACCESS] ] ] ...                  }
       { EXTEND         file-name-7   [WITH LOCK [ACCESS] ]                        }
       {              [ , file-name-8 [WITH LOCK [ACCESS] ] ] ...                  }
       { OFFER          file-name-9   [ , file-name-10]                            }
       {                                                                           }
       { AVAILABLE      file-name-11  [ , file-name-12]                            }
```

PERFORM

Format 1:

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }
```

Format 2:

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

Format 3:

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

Format 4:

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }

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

        BY { identifier-4 }   UNTIL condition-1
           { literal-2    }

    [ AFTER { identifier-5 } FROM { identifier-6 }
            { index-name-3 }      { index-name-4 }
                                  { literal-3    }

        BY { identifier-7 }   UNTIL condition-2 ] ...
           { literal-4    }
```

READ

Format 1:

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

Format 2:

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

Format 3:

```
READ file-name RECORD [WITH NO WAIT] [INTO identifier]
    [; AT END imperative-statement]
```

Format 4:

```
READ file-name RECORD [WITH NO WAIT] [INTO identifier]
    [; INVALID KEY imperative-statement]
```

RECEIVE

```
RECEIVE    cd-name { MESSAGE } INTO    identifier-1
                   { SEGMENT }
                   [ ; NO DATA imperative-statement ]
```

RELEASE

```
RELEASE record-name [FROM identifier]
```

RETURN

```
RETURN file-name RECORD [INTO identifier]  ; AT END imperative-statement
```

REWRITE

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

SEARCH, SEEK

SEARCH

Format 1:

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

    [; AT END imperative-statement-1]

     ; WHEN condition-1 { imperative-statement-2 }
                        { NEXT SENTENCE          }

    [; WHEN condition-2 { imperative-statement-3 } ]   ...
                        { NEXT SENTENCE          }
```

Format 2:

```
SEARCH ALL identifier-1  [; AT END imperative-statement-1]

    ; WHEN { data-name-1  { IS EQUAL TO } { identifier-3 }   }
           {              { IS =        } { literal-1    }   }
           {                              { arithmetic-expression-1 } }
           { condition-name-1                                        }

   [ AND { date-name-2  { IS EQUAL TO } { identifier-4 }   }   ]   ...
         {              { IS =        } { literal-2    }   }
         {                              { arithmetic-expression-2 } }
         { condition-name-2                                        }

    { imperative-statement-2 }
    { NEXT SENTENCE          }
```

SEEK

```
SEEK file-name RECORD
```

SEND

Format 1:

```
SEND  cd-name  [ FROM  identifier-1 ]
```

Format 2:

```
                                        { WITH identifier-2 }
SEND cd-name  [ FROM identifier-1 ]     { WITH ESI          }
                                        { WITH EMI          }
                                        { WITH EGI          }

[                          { { { identifier-3 }  [ LINE  ] } } ]
[ { BEFORE }  ADVANCING    { { { integer      }  [ LINES ] } } ]
[ { AFTER  }               {                                 } ]
[                          { { mnemonic-name }               } ]
[                          { { PAGE          }               } ]
```

SET

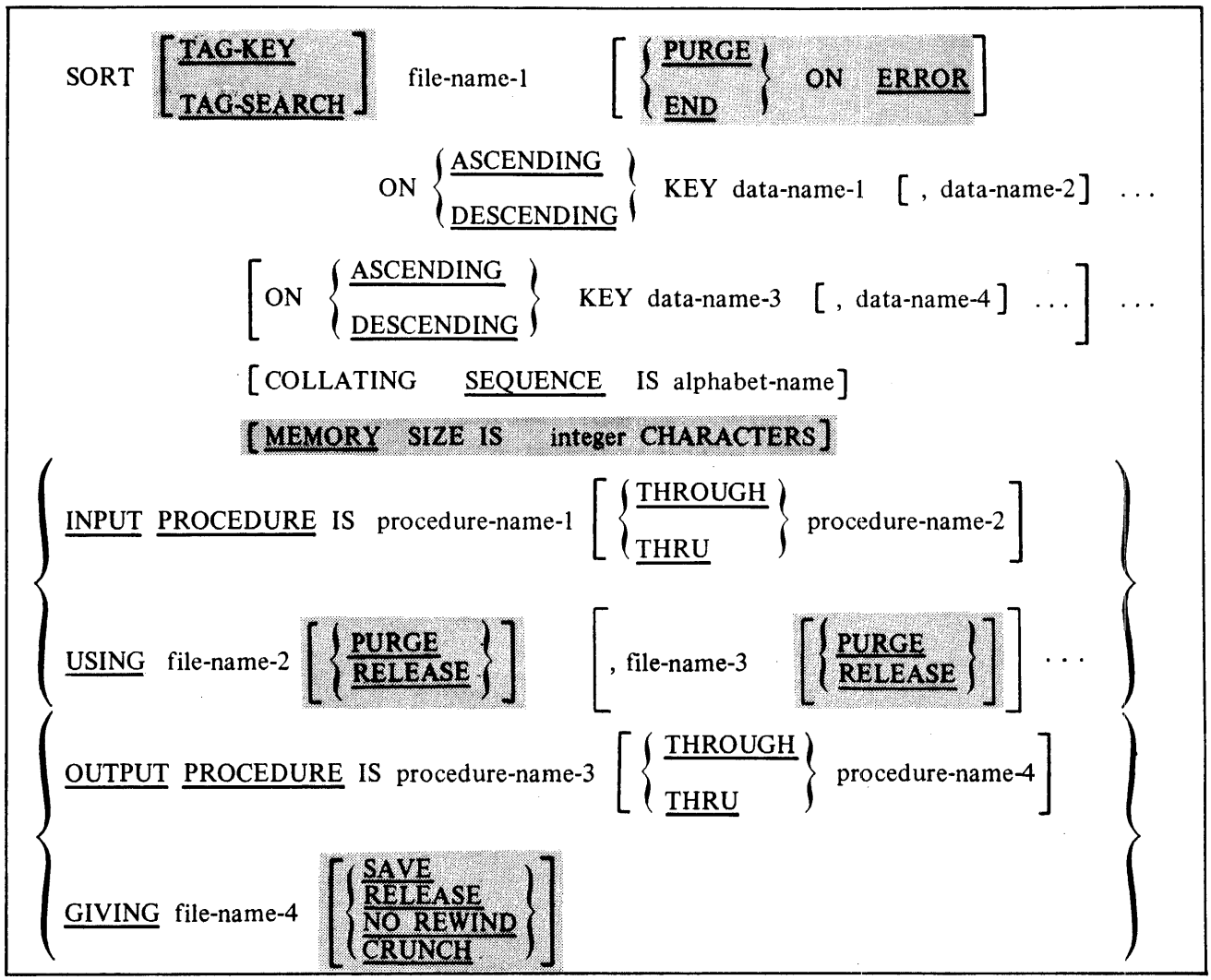
Format 1:

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

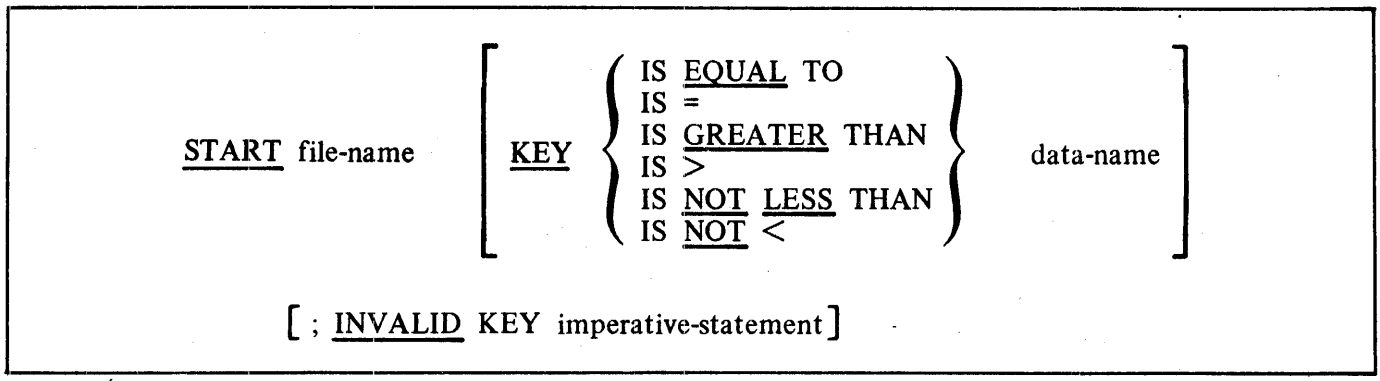
Format 2:

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

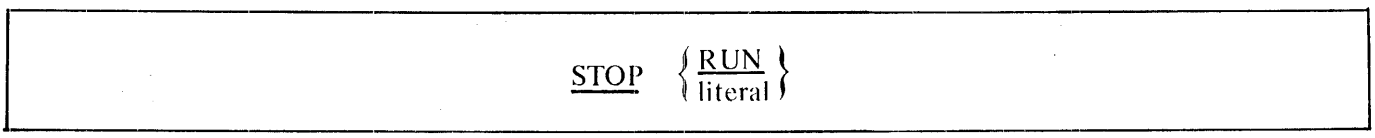
SORT


SORT
TAG-KEY
TAG-SEARCH
file-name-1
PURGE
END
ON ERROR
ON ASCENDING DESCENDING KEY data-name-1 [, data-name-2] ...
ON ASCENDING DESCENDING KEY data-name-3 [, data-name-4] ...
[COLLATING SEQUENCE IS alphabet-name]
[MEMORY SIZE IS integer CHARACTERS]
INPUT PROCEDURE IS procedure-name-1 THROUGH THRU procedure-name-2
USING file-name-2 PURGE RELEASE , file-name-3 PURGE RELEASE ...
OUTPUT PROCEDURE IS procedure-name-3 THROUGH THRU procedure-name-4
GIVING file-name-4 SAVE RELEASE NO REWIND CRUNCH

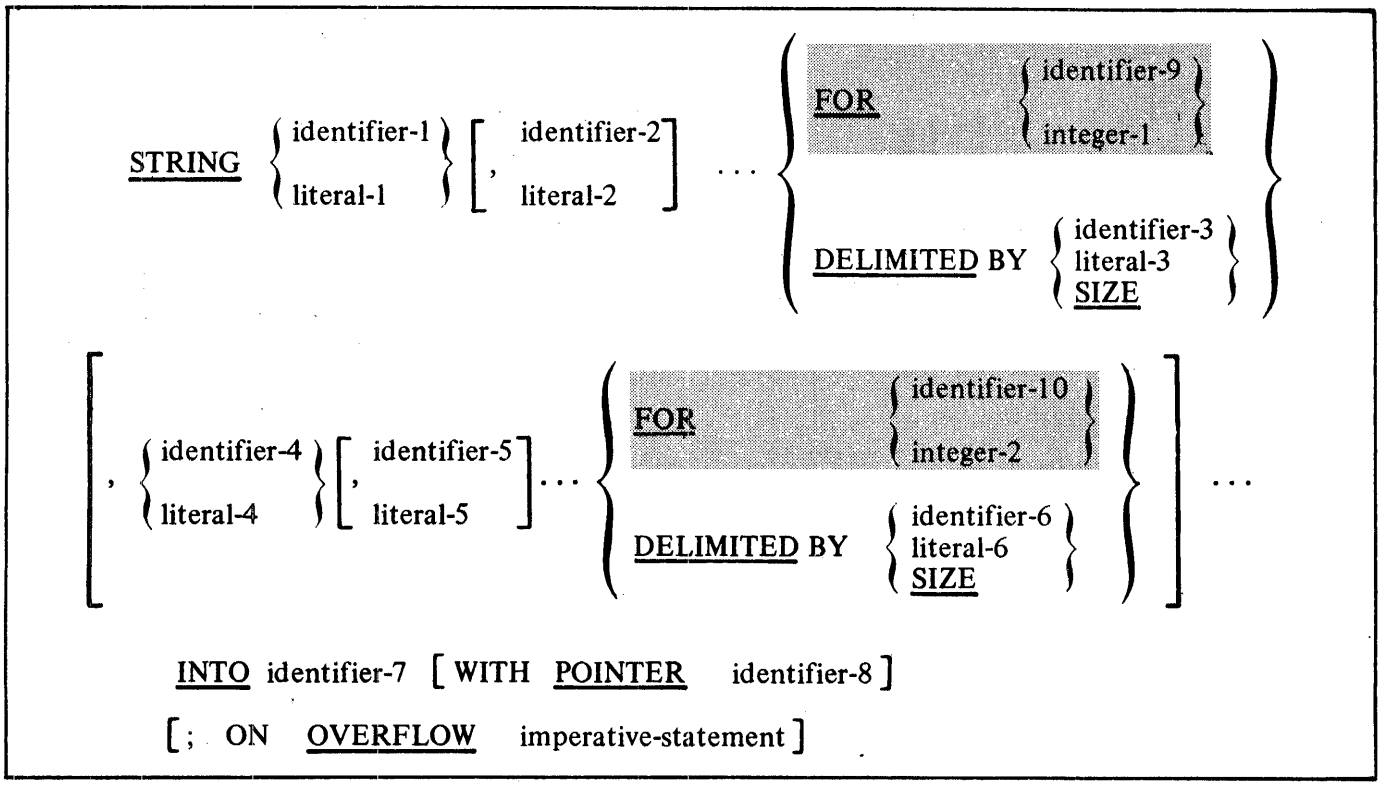
START



STOP



STRING



SUBTRACT

Format 1:

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

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

Format 2:

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

GIVING identifier-n [ROUNDED] [ , identifier-o [ROUNDED] ] ...

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

Format 3:

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ROUNDED]
         { CORR          }

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

UNSTRING

Format 1:

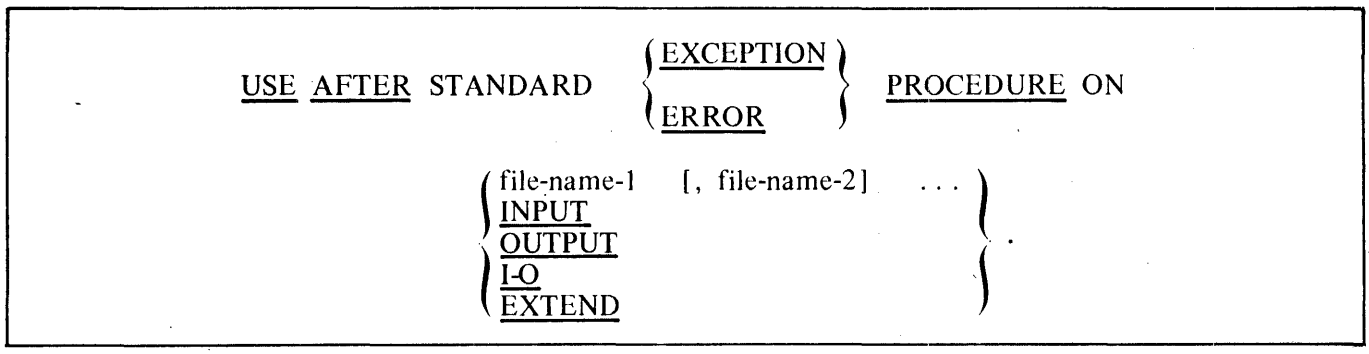
```
UNSTRING identifier-1
[ DELIMITED BY [ALL] { identifier-2 } [ , OR [ALL] { identifier-3 } ] ... ]
                     { literal-1   }               { literal-2   }
INTO identifier-4 [, DELIMITER IN identifier-5] [, COUNT IN identifier-6]
[ , identifier-7 [, DELIMITER IN identifier-8] [, COUNT IN identifier-9] ] ...
[WITH POINTER identifier-10] [ TALLYING IN identifier-11]
[; ON OVERFLOW imperative-statement]
```

Format 2:

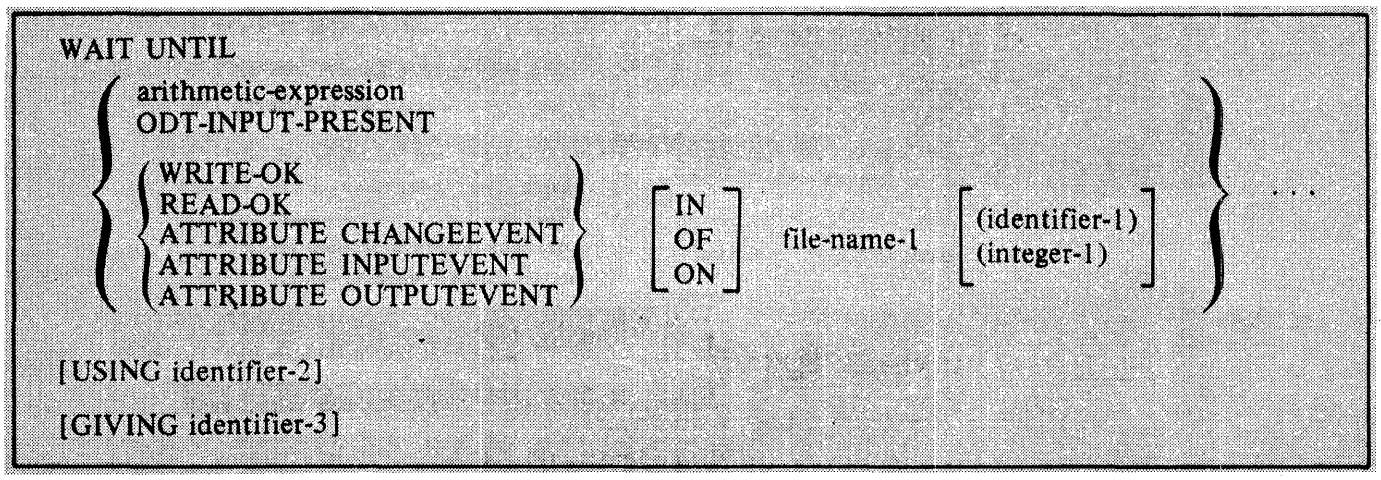
```
UNSTRING identifier-1
    INTO identifier-4 FOR { identifier-12 }
                          { integer-1     }
        [ , identifier-7 FOR { identifier-13 } ] ...
                             { integer-2     }
    [WITH POINTER identifier-10] [TALLYING IN identifier-11]
    [; ON OVERFLOW imperative-statement]
```

USE AFTER, WAIT UNTIL

USE AFTER



WAIT UNTIL



WRITE

Format 1:

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

Format 2:

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

Format 3:

```
WRITE record-name [WITH NO WAIT] [FROM identifier-1]
    [; INVALID KEY imperative-statement]
```

DATA BASE MANAGEMENT

DATA DIVISION

General Format

```
DATA-BASE SECTION.
DB [ logical-data-base-name { IN } ] data-base-name.
                            { OF }
01  internal-data-set-name  INVOKE  data-set-name.
```

PROCEDURE DIVISION

Format for Selection Expression

```
{ [data-set-name VIA] set-selection-expression }
{ [FIRST ]                                     }
{ [LAST  ] data-set-name                       }
{ [NEXT  ]                                     }
{ [PRIOR ]                                     }
```

Format for Set Selection Expression

```
[FIRST ]  { set-name    }  [AT key-condition]
[LAST  ]  { subset-name }
[NEXT  ]
[PRIOR ]
```

DATA BASE MANAGEMENT VERBS

BEGIN-TRANSACTION

```
BEGIN-TRANSACTION { AUDIT    } restart-data-set-name
                  { NO-AUDIT }
                                [ ON EXCEPTION statement ]
```

CLOSE

```
CLOSE data-base-name [ON EXCEPTION statement]
```

CREATE

```
CREATE data-set-name [ (arithmetic-statement) ]
                     [ ON EXCEPTION statement ]
```

DELETE

```
DELETE data-set-name [ON EXCEPTION statement ]
```

END-TRANSACTION

```
END-TRANSACTION { AUDIT    } restart-data-set-name [ SYNC ]
                { NO-AUDIT }
                                [ ON EXCEPTION statement ]
```

FIND

```
FIND { selection-expression } [ON EXCEPTION statement ]
```

DATA BASE MANAGEMENT

FREE

```
FREE  data-set-name  [ ON EXCEPTION statement ]
```

INSERT

```
INSERT  data-set-name  INTO  subset-name  [ ON EXCEPTION statement ]
```

LOCK

```
LOCK  { selection-expression }  [ ON EXCEPTION statement ]
```

OPEN

```
OPEN  { { INQUIRY }  data-base-name }  [ ON EXCEPTION statement ]
      { { UPDATE  }                }
```

RECREATE

```
RECREATE  data-set-name  [ (arithmetic-expression) ]
                         [ ON EXCEPTION statement ]
```

REMOVE CURRENT FROM

```
REMOVE  CURRENT  FROM  subset-name  [ ON EXCEPTION statement ]
```

STORE

```
STORE  data-set-name  [ ON EXCEPTION statement ]
```

APPENDIX C
COBOL74 GRAPHICS

Table C-1 lists the 8-bit EBCDIC internal code together with the USASCII-7, 80-column card codes and the associated control or special codes or graphics. Table C-2 presents similar information, but restricted to and in sequence of the USASCII-7 code. Table C-3 explains all the control or special codes.

Column headings have the following meanings in tables C-1 and C-2:

Column Heading	Meaning
A	EBCDIC sequence number and decimal value
B	EBCDIC hexadecimal representation
C	ASCII-7 sequence number and decimal value
D	ASCII-7 hexadecimal representation
E	80-column card code
F	Graphic or code

Hexadecimal representation means that standard convention is followed for the 8-bit internal codes. These codes are shown translated as pairs of hexadecimal numbers in the following examples.

Examples:

Hex Number Pair		8-Bit Internal Code	
		8 4 2 1	8 4 2 1
@39@	=	0 0 1 1	1 0 0 1
@BE@	=	1 0 1 1	1 1 1 0
@0F@	=	0 0 0 0	1 1 1 1

Table C-1. B 1000 Codes in EBCDIC Sequence

A	B	C	D	E		F
0	00	0	00	12	0-1-8-9	NUL (Null)
1	01	1	01	12	1-9	SOH (Start of Heading)
2	02	2	02	12	2-9	STX (Start of Text)
3	03	3	03	12	3-9	ETX (End of Text)
4	04			12	4-9	
5	05	9	09	12	5-9	HT (Horizontal Tab)
6	06			12	6-9	
7	07	127	7F	12	7-9	DEL (Delete)
8	08			12	8-9	
9	09			12	1-8-9	
10	0A			12	2-8-9	
11	0B	11	0B	12	3-8-9	VT (Vertical Tab)
12	0C	12	0C	12	4-8-9	FF (Form Feed)

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
13	0D	13	0D	12 5-8-9	CR (Carriage Return)
14	0E	14	0E	12 6-8-9	SO (Shift Out)
15	0F	15	0F	12 7-8-9	SI (Shift In)
16	10	16	10	12-11-1-8-9	DLE (Data Link Escape)
17	11	17	11	11-1-9	DC1 (Device Control 1)
18	12	18	12	11-2-9	DC2 (Device Control 2)
19	13	19	13	11-3-9	DC3 (Device Control 3)
20	14			11-4-9	
21	15			11-5-9	NL (New Line)
22	16	8	08	11-6-9	BS (Backspace)
23	17			11-7-9	
24	18	24	18	11-8-9	CAN (Cancel)
25	19	25	19	11-1-8-9	EM (End of Medium)
26	1A			11-2-8-9	
27	1B			11-3-8-9	
28	1C	28	1C	11-4-8-9	FS (File Separator)
29	1D	29	1D	11-5-8-9	GS (Group Separator)
30	1E	30	1E	11-6-8-9	RS (Record Separator)
31	1F	31	1F	11-7-8-9	US (Unit Separator)
32	20			11-0-1-8-9	
33	21			0-1-9	
34	22			0-2-9	
35	23			0-3-9	
36	24			0-4-9	
37	25	10	0A	0-5-9	LF (Line Feed)
38	26	23	17	0-6-9	ETB (End of Transmission Block)
39	27	27	1B	0-7-9	ESC (Escape)
40	28			0-8-9	
41	29			0-1-8-9	
42	2A			0-2-8-9	
43	2B			0-3-8-9	
44	2C			0-4-8-9	
45	2D	5	05	0-5-8-9	ENQ (Enquiry)
46	2E	6	06	0-6-8-9	ACK (Acknowledge)
47	2F	7	07	0-7-8-9	BEL (Bell)
48	30			12-11-0-1-8-9	
49	31			1-9	
50	32	22	16	2-9	SYN (Synchronous Idle)
51	33			3-9	
52	34			4-9	
53	35			5-9	
54	36			6-9	
55	37	4	04	7-9	EOT (End of Transmission)0

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
56	38			8-9	
57	39			1-8-9	
58	3A			2-8-9	
59	3B			3-8-9	
60	3C	20	14	4-8-9	DC4 (Device Control 4)
61	3D	21	15	5-8-9	NAK (Negative Acknowledge)
62	3E			6-8-9	
63	3F	26	1A	7-8-9	
64	40	32	20	(No Punches)	(SPACE)
65	41			12 0-1-8-9	
66	42			12 0-2-9	
67	43			12 0-3-9	
68	44			12 0-4-9	
69	45			12 0-5-9	
70	46			12 0-6-9	
71	47			12 0-7-9	
72	48			12 0-8-9	
73	49			12 1-8	
74	4A	91	5B	12 2-8	[(LEFT BRACKET)
75	4B	46	2E	12 3-8	. (PERIOD, DECIMAL POINT)
76	4C	60	3C	12 4-8	< (LESS-THAN SIGN)
77	4D	40	28	12 5-8	((LEFT PARENTHESIS)
78	4E	43	2B	12 6-8	+ (PLUS SIGN)
79	4F	33	21	12 7-8	| (VERTICAL BAR)
80	50	38	26	12	& (AMPERSAND)
81	51			12-11-1-9	
82	52			12-11-2-9	
83	53			12-11-3-9	
84	54			12-11-4-9	
85	55			12-11-5-9	
86	56			12-11-6-9	
87	57			12-11-7-9	
88	58			12-11-8-9	
89	59			11-1-8	
90	5A	93	5D	11-2-8	] (RIGHT BRACKET)
91	5B	36	24	11-3-8	$ (DOLLAR SIGN)
92	5C	42	2A	11-4-8	* (ASTERISK)
93	5D	41	29	11-5-8	) (RIGHT PARENTHESIS)
94	5E	59	3B	11-6-8	; (SEMICOLON)
95	5F	94	5E	11-7-8	¬ (LOGICAL NOT)
96	60	45	2D	11	- (MINUS, HYPHEN)
97	61	47	2F	0-1	/ (SLASH)
98	62			11-0-2-9	
99	63			11-0-3-9	

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
100	64			11-0-4-9	
101	65			11-0-5-9	
102	66			11-0-6-9	
103	67			11-0-7-9	
104	68			11-0-8-9	
105	69			0-1-8	
106	6A	124	7C	12-11	
107	6B	44	2C	0-2-8	, (COMMA)
108	6C	37	25	0-4-8	% (PERCENT SIGN)
109	6D	95	5F	0-5-8	_ (UNDERSCORE)
110	6E	62	3E	0-6-8	> (GREATER-THAN SIGN)
111	6F	63	3F	0-7-8	? (QUESTION MARK)
112	70			12-11-0	! (EXCLAMATION POINT)
113	71			12-11-0-1-9	
114	72			12-11-0-2-9	
115	73			12-11-0-3-9	
116	74			12-11-0-4-9	
117	75			12-11-0-5-9	
118	76			12-11-0-6-9	
119	77			12-11-0-7-9	
120	78			12-11-0-8-9	
121	79	96	60	1-8	
122	7A	58	3A	2-8	: (COLON)
123	7B	35	23	3-8	# (NUMBER OR POUND SIGN)
124	7C	64	40	4-8	@ (AT SIGN)
125	7D	39	27	5-8	' (APOSTROPHE)
126	7E	61	3D	6-8	= (EQUAL SIGN)
127	7F	34	22	7-8	" (QUOTATION MARK)
128	80			12 0-1-8	
129	81	97	61	12 0-1	a
130	82	98	62	12 0-2	b
131	83	99	63	12 0-3	c
132	84	100	64	12 0-4	d
133	85	101	65	12 0-5	e
134	86	102	66	12 0-6	f
135	87	103	67	12 0-7	g
136	88	104	68	12 0-8	h
137	89	105	69	12 0-9	i
138	8A				
139	8B				
140	8C				
141	8D				
142	8E				
143	8F				

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
144	90			12-11-1-8	
145	91	106	6A	12-11-1	j
146	92	107	6B	12-11-2	k
147	93	108	6C	12-11-3	l
148	94	109	6D	12-11-4	m
149	95	110	6E	12-11-5	n
150	96	111	6F	12-11-6	o
151	97	112	70	12-11-7	p
152	98	113	71	12-11-8	q
153	99	114	72	12-11-9	r
154	9A			12-11-2-8	
155	9B			12-11-3-8	
156	9C			12-11-4-8	
157	9D			12-11-5-8	
158	9E			12-11-6-8	
159	9F			12-11-7-8	
160	A0			11-0-1-8	
161	A1	126	7E	11-0-1	¢ (CENTS SIGN)
162	A2	115	73	11-0-2	s
163	A3	116	74	11-0-3	t
164	A4	117	75	11-0-4	u
165	A5	118	76	11-0-5	v
166	A6	119	77	11-0-6	w
167	A7	120	78	11-0-7	x
168	A8	121	79	11-0-8	y
169	A9	122	7A	11-0-9	z
170	AA			11-0-2-8	
171	AB			11-0-3-8	
172	AC			11-0-4-8	
173	AD			11-0-5-8	
174	AE			11-0-6-8	
175	AF			11-0-7-8	
176	B0			12-11-0-1-8	
177	B1			12-11-0-1	
178	B2			12-11-0-2	
179	B3			12-11-0-3	
180	B4			12-11-0-4	
181	B5			12-11-0-5	
182	B6			12-11-0-6	
183	B7			12-11-0-7	
184	B8			12-11-0-8	
185	B9			12-11-0-9	
186	BA			12-11-0-2-8	

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
187	BB			12-11-0-3-8	
188	BC			12-11-0-4-8	
189	BD			12-11-0-5-8	
190	BE			12-11-0-6-8	
191	BF			12-11-0-7-8	
192	C0	123	7B	12 0	{ (LEFT BRACE)
193	C1	65	41	12 1	A
194	C2	66	42	12 2	B
195	C3	67	43	12 3	C
196	C4	68	44	12 4	D
197	C5	69	45	12 5	E
198	C6	70	46	12 6	F
199	C7	71	47	12 7	G
200	C8	72	48	12 8	H
201	C9	73	49	12 9	I
202	CA				
203	CB				
204	CC				
205	CD				
206	CE				
207	CF				
208	D0	125	7D		} (RIGHT BRACE)
209	D1	74	4A	11 1	J
210	D2	75	4B	11 2	K
211	D3	76	4C	11 3	L
212	D4	77	4D	11 4	M
213	D5	78	4E	11 5	N
214	D6	79	4F	11 6	O
215	D7	80	50	11 7	P
216	D8	81	51	11 8	Q
217	D9	82	52	11 9	R
218	DA				
219	DB				
220	DC				
221	DD				
222	DE				
223	DF				
224	E0	92	5C		\ (REVERSE SLASH)
225	E1				
226	E2	83	53	0-2	S
227	E3	84	54	0-3	T
228	E4	85	55	0-4	U
229	E5	86	56	0-5	V

Table C-1. B 1000 Codes in EBCDIC Sequence (Cont)

A	B	C	D	E	F
230	E6	87	57	0-6	W
231	E7	88	58	0-7	X
232	E8	89	59	0-8	Y
233	E9	90	5A	0-9	Z
234	EA				
235	EB				
236	EC				
237	ED				
238	EE				
239	EF				
240	F0	48	30	0	0
241	F1	49	31	1	1
242	F2	50	32	2	2
243	F3	51	33	3	3
244	F4	52	34	4	4
245	F5	53	35	5	5
246	F6	54	36	6	6
247	F7	55	37	7	7
248	F8	56	38	8	8
249	F9	57	39	9	9
250	FA				
251	FB				
252	FC				
253	FD				
254	FE				
255	FF				

Table C-2 lists the 8-bit EBCDIC internal code together with the USASCII-7, and 80-column card codes in sequence of the USASCII-7 code.

Table C-2. B 1000 Codes in ASCII-7 Sequence

A	B	C	D	E	F
0	00	0	00	12 0-1-8-9	NUL (Null)
1	01	1	01	12 1-9	SOH (Start of Heading)
2	02	2	02	12 2-9	STX (Start of Text)
3	03	3	03	12 3-9	ETX (End of Text)
55	37	4	04	7-9	EOT (End of Transmission)
45	2D	5	05	0-5-8-9	ENQ (Enquiry)
46	2E	6	06	0-6-8-9	ACK (Acknowledge)
47	2F	7	07	0-7-8-9	BEL (Bell)
22	16	8	08	11-6-9	BS (Backspace)
5	05	9	09	12 5-9	HT (Horizontal Tab)
37	25	10	0A	0-5-9	LF (Line Feed)

Table C-2. B 1000 Codes in ASCII-7 Sequence (Cont)

A	B	C	D	E	F
11	0B	11	0B	12 3-8-9	VT (Vertical Tab)
12	0C	12	0C	12 4-8-9	FF (Form Feed)
13	0D	13	0D	12 5-8-9	CR (Carriage Return)
14	0E	14	0E	12 6-8-9	SO (Shift Out)
15	0F	15	0F	12 7-8-9	SI (Shift In)
16	10	16	10	12-11-1-8-9	DLE (Data Link Escape)
17	11	17	11	11-1-9	DC1 (Device Control 1)
18	12	18	12	11-2-9	DC2 (Device Control 2)
19	13	19	13	11-3-9	DC3 (Device Control 3)
60	3C	20	14	4-8-9	DC4 (Device Control 4)
61	3D	21	15	5-8-9	NAK (Negative Acknowledge)
50	32	22	16	2-9	SYN (Synchronous Idle)
38	26	23	17	0-6-9	ETB (End of Transm. Block)
24	18	24	18	11-8-9	CAN (Cancel)
25	19	25	19	11-1-8-9	EM (End of Medium)
63	3F	26	1A	7-8-9	SUB (Substitute)
39	27	27	1B	0-7-9	ESC (Escape)
28	1C	28	1C	11-4-8-9	FS (File Separator)
29	1D	29	1D	11-5-8-9	GS (Group Separator)
30	1E	30	1E	11-6-8-9	RS (Record Separator)
31	1F	31	1F	11-7-8-9	US (Unit Separator)
64	40	32	20	[NO PUNCHES]	(SPACE)
79	4F	33	21	12 7-8	| (VERTICAL BAR)
127	7F	34	22	7-8	" (QUOTATION MARK)
123	7B	35	23	3-8	# (NUMBER OR POUND SIGN)
91	5B	36	24	11-3-8	$ (DOLLAR SIGN)
108	6C	37	25	0-4-8	% (PERCENT SIGN)
80	50	38	26	12	& (AMPERSAND)
125	7D	39	27	5-8	' (APOSTROPHE)
77	4D	40	28	12 5-8	((LEFT PARENTHESIS)
93	5D	41	29	11-5-8	) (RIGHT PARENTHESIS)
92	5C	42	2A	11-4-8	* (ASTERISK)
78	4E	43	2B	12 6-8	+ (PLUS SIGN)
107	6B	44	2C	0-2-8	, (COMMA)
96	60	45	2D	11	- (MINUS SIGN, HYPHEN)
75	4B	46	2E	12 3-8	. (PERIOD)
97	61	47	2F	0-1	/ (SLASH)
240	F0	48	30	0	0
241	F1	49	31	1	1
242	F2	50	32	2	2
243	F3	51	33	3	3
244	F4	52	34	4	4
245	F5	53	35	5	5
246	F6	54	36	6	6
247	F7	55	37	7	7
248	F8	56	38	8	8

Table C-2. B 1000 Codes in ASCII-7 Sequence (Cont)

A	B	C	D	E	F
249	F9	57	39	9	9
122	7A	58	3A	2-8	: (COLON)
94	5E	59	3B	11-6-8	; (SEMICOLON)
76	4C	60	3C	12 4-8	< (LESS-THAN SIGN)
126	7E	61	3D	6-8	= (EQUAL SIGN)
110	6E	62	3E	0-6-8	> (GREATER-THAN SIGN)
111	6F	63	3F	0-7-8	? (QUESTION MARK)
124	7C	64	40	4-8	@ (AT SIGN)
193	C1	65	41	12 1	A
194	C2	66	42	12 2	B
195	C3	67	43	12 3	C
196	C4	68	44	12 4	D
197	C5	69	45	12 5	E
198	C6	70	46	12 6	F
199	C7	71	47	12 7	G
200	C8	72	48	12 8	H
201	C9	73	49	12 9	I
209	D1	74	4A	11-1	J
210	D2	75	4B	11-2	K
211	D3	76	4C	11-3	L
212	D4	77	4D	11-4	M
213	D5	78	4E	11-5	N
214	D6	79	4F	11-6	O
215	D7	80	50	11-7	P
216	D8	81	51	11-8	Q
217	D9	82	52	11-9	R
226	E2	83	53	0-2	S
227	E3	84	54	0-3	T
228	E4	85	55	0-4	U
229	E5	86	56	0-5	V
230	E6	87	57	0-6	W
231	E7	88	58	0-7	X
232	E8	89	59	0-8	Y
233	E9	90	5A	0-9	Z
74	4A	91	5B	12 2-8	[(LEFT BRACKET)
224	E0	92	5C		\ (REVERSE SLASH)
90	5A	93	5D	11-2-8	] (RIGHT BRACKET)
95	5F	94	5E	11-7-8	¬ (LOGICAL NOT)
109	6D	95	5F	0-5-8	_ (UNDERSCORE)
121	79	96	60	1-8	
129	81	97	61	12 0-1	a
130	82	98	62	12 0-2	b
131	83	99	63	12 0-3	c
132	84	100	64	12 0-4	d
133	85	101	65	12 0-5	e

Table C-2. B 1000 Codes in ASCII-7 Sequence (Cont)

A	B	C	D	E	F
134	86	102	66	12 0-6	f
135	87	103	67	12 0-7	g
136	88	104	68	12 0-8	h
137	89	105	69	12 0-9	i
145	91	106	6A	12-11-1	j
146	92	107	6B	12-11-2	k
147	93	108	6C	12-11-3	l
148	94	109	6D	12-11-4	m
149	95	110	6E	12-11-5	n
150	96	111	6F	12-11-6	o
151	97	112	70	12-11-7	p
152	98	113	71	12-11-8	q
153	99	114	72	12-11-9	r
162	A2	115	73	11-0-2	s
163	A3	116	74	11-0-3	t
164	A4	117	75	11-0-4	u
165	A5	118	76	11-0-5	v
166	A6	119	77	11-0-6	w
167	A7	120	78	11-0-7	x
168	A8	121	79	11-0-8	y
169	A9	122	7A	11-0-9	z
192	C0	123	7B	12 0	{ (LEFT BRACE)
106	6A	124	7C	12-11	
208	D0	125	7D		} (RIGHT BRACE)
161	A1	126	7E	11-0-1	¢ (CENTS SIGN)
7	07	127	7F	12 7-9	DEL (Delete)

Table C-3 explains all of the control and special codes. The symbols are presented in the order of appearance in table C-1 and C-2.

Table C-3. Description of Control and Special Characters

Symbol	Name/Function
NUL	Null: The all zeros character which may serve to accomplish time fill and media fill.
SOH	Start of Heading: A communication control character used at the beginning of a sequence of characters which constitutes a machine-sensible address or routing information. Such a sequence is referred to as the "heading." An STX character has the effect of terminating a heading.
STX	Start of Text: A communication control character which precedes a sequence of characters which are to be treated as an entity and which are entirely transmitted to the ultimate destination. Such a sequence is referred to as "text." STX may be used to terminate a sequence of characters started by SOH.
ETX	End of Text: A communication control character used to terminate a sequence of characters started with STX and transmitted as an entity.
EOT	End of Transmission: A communication control character used to indicate the conclusion of a transmission which may have contained one or more texts and any associated headings.
ENQ	Enquiry: A communication control character used in data communication systems as a request for a response from a remote station. It may be used as a "Who Are You" (WRU) to obtain identification, or may be used to obtain station status, or both.
ACK	Acknowledge: A communication control character transmitted by a receiver as an affirmative response to a sender.
BEL	Bell: A character for use when there is a need to call for human attention. It may control alarm or attention devices.
BS	Backspace: A format effector that controls the movement of the printing mechanism one print position backward on the same print line.
HT	Horizontal Tabulation: A format effector that controls the movement of the printing mechanism to the next position in a series of predetermined positions along the print line.

Table C-3. Description of Control and Special Characters (Cont)

Symbol	Name/Function
LF	Line Feed: A format effector that controls movement one line at a time.
VT	Vertical Tabulation: A format effector that controls movement to the next in a series of predetermined lines.
FF	Form Feed: A format effector that controls the movement of the printing position to the first predetermined printing line on the next form or page.
CR	Carriage Return: A format effector that controls the movement of the print mechanism to the first print position on the same print line.
SO	Shift Out: A control character indicating that the code combinations that follow shall be interpreted as outside of the character set of the standard code table until a Shift In character is detected.
SI	Shift In: A control character indicating that the code combinations that follow shall be interpreted according to the standard code table.
DLE	Date Link Escape: A communication control character that will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communication networks.
DC1 DC2 DC3 DC4	Device Controls: Characters for the control of ancillary devices associated with data processing or telecommunication systems, more especially switching devices ON or OFF. If a single "stop" control is required to interrupt or turn off ancillary devices, DC4 is the preferred assignment.
NAK	Negative Acknowledge: A communication control character transmitted by a receiver as a negative response to the sender.
SYN	Synchronous Idle: A communication control character used by a synchronous transmission system in the absence of any other character to provide a signal from which synchronism may be achieved or retained.
ETB	End of Transmission Block: A communication control character used to indicate the end of a block of data for communication purposes. ETB is used for blocking data where the block structure is not necessarily related to the processing format.

Table C-3. Description of Control and Special Characters (Cont)

Symbol	Name/Function
CAN	Cancel: A control character used to indicate that the data with which it is sent is in error and is to be disregarded.
EM	End of Medium: A control character associated with the sent data which may be used to identify the physical end of the medium, or the end of the used, or wanted, portion of information recorded on a medium. The position of this character does not necessarily correspond to the physical end of the medium.
SUB	Substitute: A character that may be substituted for a character which is determined to be invalid or in error.
ESC	Escape: A control character intended to provide code extension (supplementary characters) in general information interchange. The Escape character itself is a prefix affecting the interpretation of a limited number of contiguously following characters.
FS GS RS US	File Separator, Group Separator, Record Separator, and Unit Separator. These information separators may be optionally used within data, except that their hierarchical relationship shall be such that FS is the most inclusive, then GS, then RS, and US is least inclusive. The content and length of a File, Group, Record, or Unit are not specified.
SP	Space: A normally non-printing graphic character used to separate words. It is also a format effector which controls the movement of the printing position, one printing position forward.
DEL	Delete: This character is used primarily to "erase" or "obliterate" erroneous or unwanted characters in perforated tape. In the strict sense, DEL is not a control character.

APPENDIX D
GLOSSARY

INTRODUCTION

The terms in this glossary are defined in accordance with meaning as used in this document describing COBOL74 and may not have the same meaning for other languages.

Data Base Management definitions are not included in this glossary. Refer to the B 1000 Systems Data Management System II (DMSII) Reference Manual for these definitions.

DEFINITIONS

Abbreviated Combined Relation Condition
The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode
The manner in which records are to be operated upon within a file.

Actual Decimal Point
The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Actual Key
A key whose contents identify a logical record in a sequential file.

Alphabet-Name
A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character
A character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

Alphanumeric Character
Any character in the computer character set.

Alternate Record Key
A key, other than the prime record key, whose contents identify a record within an Indexed File.

Arithmetic Expression
An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator
A single character, or a fixed 2-character combination, that belongs to the following set:

Character	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Ascending Key
A key upon the values of which data is ordered starting with the lowest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point
A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition
A condition caused:

a. During the execution of a READ statement for a sequentially accessed file.

b. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

c. During the execution of a SEARCH statement, when the search operator terminates without satisfying the condition specified in any of the associated WHEN phrases.

Block
A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Burroughs Network Architecture (BNA)
A system which provides a means to perform distributed data processing such as remote data base access, file transfer, remote task initiation and control, logical I/O, interprocess communication, and resource sharing.

Called Program
A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program
A program which executes a CALL to another program.

Cd-Name
A user-defined word that names an MCS interface area described in a communication description entry within the COMMUNICATION SECTION of the DATA DIVISION.

Character
The basic indivisible unit of the language.

Character Position
A character position is the amount of physical storage required to store a single standard data format character described as usage is DISPLAY.

Character-String
A sequence of contiguous characters which form a COBOL74 word, a literal, a PICTURE character-string, or a comment-entry.

Class Condition
The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause
A clause is an ordered set of consecutive COBOL74 character-strings whose purpose is to specify an attribute of an entry.

COBOL74 Character Set
The complete COBOL74 character set consists of the 52 characters listed below:

Character	Meaning
0,1,...,9	Digit
A,B,...,Z	Letter
	Space (blank)
+	Plus sign
–	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
@	Commercial at

COBOL74 Word
Refer to Word.

Collating Sequence
The sequence in which the characters are acceptable in a computer are ordered for purposes of sorting, merging, and comparing.

Column
A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

Combined Condition
A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

Comment-Entry
An entry in the IDENTIFICATION DIVISION that may be any combination of characters from the computer character set.

Comment Line
A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in Area A and Area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line cause page ejection before printing a comment.

Communication Description Entry
An entry in the COMMUNICATION SECTION of the DATA DIVISION that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. It describes the interface between the Data Communication Subsystem and the COBOL74 program.

Communication Device
A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

Communication Section
The section of the DATA DIVISION that describes the interface areas between the MCS and the program, composed of one or more CD description entries.

Compile Time
The time when a COBOL74 source program is translated to a COBOL74 object program by the COBOL74 compiler.

Compiler-Directing Statement
A statement that begins with a compiler-directing verb and causes the compiler to take a specific action during compilation.

Complex Condition
A condition in which one or more logical operators act upon one or more conditions. Refer to the terms Negated Simple Condition, Combined Condition, and Negated Combined Condition.

Computer-Name
A system name that identifies the computer upon which the program is to be compiled or run.

Condition
A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in this language in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name
A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of a switch or device.

Condition-Name Condition
The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression
A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. Refer to the terms Simple Condition and Complex Condition.

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

Conditional Variable
A data item one or more values of which has a condition-name assigned to it.

Configuration Section
A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object computers.

Connective
A reserved word that is used to:

a. Associate a data-name, paragraph-name, condition-name, or text-name with the qualifier.
b. Link two or more operands written in a series.
c. Form conditions (logical connectives). Refer to the term Logical Operator.

Contiguous Items
Items that are described by consecutive entries in the DATA DIVISION, and that bear a definite hierarchic relationship to each other.

Counter
A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency Sign
The character '$' of the COBOL74 character set.

Currency Symbol
The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL74 source program, the currency symbol is identical to the currency sign.

Current Record
The record which is available in the record area associated with the file.

Current Record Pointer
A conceptual entity that is used in the selection of the next record.

Data Clause
A clause that appears in a data description entry in the DATA DIVISION and provides information describing a particular attribute of a data item.

Data Communication Subsystem
Consists of a Network Controller and one or more programs (optional MCS). The Network Controller is responsible for handling all the details of the data communications line discipline.

Data Description Entry
An entry in the DATA DIVISION that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item
A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL74 program.

Data-Name
A user-defined word that names a data item described in a data description entry in the DATA DIVISION. When used in the general formats, data-name represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line
A debugging line is any line with D in the indicator area of the line.

Debugging Section
A debugging section is a section that contains a USE FOR DEBUGGING statement.

Declaratives
A set of one or more special purpose sections, written at the beginning of the PROCEDURE DIVISION, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative-Sentence
A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter
A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key
A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, according to the rules for comparing data items.

Destination
The symbolic identification of the receiver of a transmission from a queue.

Digit Position
A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage of the data item describing the digit position.

Division
A set of zero, one or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four divisions in a COBOL74 program: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE.

Division Header
A combination of words followed by a period and a space that indicates that beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING data-name-1 [data-name-2] ... ] .
```

Dynamic Access
An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

Editing Character
A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	Space
0	Zero
+	Plus
–	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Stroke (virgule, slash)

Elementary Item
A data item that is described as not being further logically subdivided.

End of PROCEDURE DIVISION
The physical position in a COBOL74 source program after which no further procedures appear.

Entry
Any descriptive set of consecutive clauses terminated by a period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL74 source program.

Environment Clause
A clause that appears as part of an ENVIRONMENT DIVISION entry.

Execution Time
See Object Time.

Extend Mode
The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Figurative Constant
A compiler-generated value referenced through the use of certain reserved words.

File
A collection of records.

File Clause
A clause that appears as part of any of the following DATA DIVISION entries:

File description (FD)
Sort-Merge file description (SD)
Communication description (CD)

FILE-CONTROL
The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

File Description Entry
An entry in the FILE SECTION of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name
A user-defined word that names a file described in a file description entry within the FILE SECTION of the DATA DIVISION.

File Organization
The permanent logical file structure established at the time that a file is created.

File Section
The section of the DATA DIVISION that contains file description entries and Sort-Merge file description entries together with associated record descriptions.

Format
A specific arrangement of a set of data.

Group Item
A named contiguous set of elementary or group items.

Hexadecimal Literal
A character-string bounded by @ signs. The string of characters must consist of one or more characters chosen from the set of hexadecimal digits consisting of the digits 0 through 9, A, B, C, D, E, and F. The characters A through F are the hexadecimal digit representation for the decimal values 10 through 15.

High Order End
The leftmost character of a string of characters.

I-O-Control
The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I-O Mode
The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

Identifier
A data-name, followed as required by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

Imperative Statement
A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Index
A computer storage position or register, the contents of which represent the identification of a particular element in a table.

Index Data Item
A data item in which the value associated with an index-name can be stored.

Index-Name
A user-defined word that names an index associated with a specific table.

Indexed Data-Name
An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File
A file with indexed organization.

Indexed Organization
The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Input File
A file that is opened in the input mode.

Input Mode
The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output
A file that is opened in the I-O mode.

INPUT-OUTPUT SECTION
The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Input Procedure
A set of statements that is executed each time a record is released to the sort file.

Integer
A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed nor zero unless explicitly allowed by the rules of that format.

Intermediate Data Item
A signed numeric data item provided by the compiler that contains the results developed in the course of an arithmetic operation prior to the final result being moved to the resultant-identifier, if any.

Invalid Key Condition
A condition, at object time, caused when a specific value of the key associated with an Indexed or Relative File is determined to be invalid.

Key
A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference
The key, either prime or alternate, currently being used to access records within an Indexed File.

Key Word
A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name
A system name that specifies a particular programming language.

Level Indicator
Two alphabetic characters that identify a specific type of file or a position in hierarchy.

Level-Number
A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one-or two-digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

Library-Name
A user-defined word that names a COBOL74 library that is to be used by the compiler for a given source program compilation.

Library-Text
A sequence of character-strings and/or separators in a COBOL74 library.

Line Number
An integer that denotes the vertical position of a report line on a page.

Linkage Section
The section in the DATA DIVISION of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal
A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator
One of the reserved words AND, OR, or NOT. In the formation of a condition, AND and/or OR can be used as logical connectives. NOT can be used for logical negation.

Logical Record
The most inclusive data item. The level-number for a record is 01.

Low Order End
The rightmost character of a string of characters.

Mass Storage
A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage Control System (MSCS)
An input-output control system that directs, or controls, the processing of mass storage files.

Mass Storage File
A collection of records that is assigned to a mass storage medium.

MCS
Refer to Message Control System.

Merge File
A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Message
Data associated with an end of message indicator or an end of group indicator. Refer to Message Indicators.

Message Control System (MCS)
A communication control system that supports the processing of messages.

Message Count
The count of the number of complete messages that exist in the designated queue of messsages.

Message Indicators
EGI (end of group indicator), EMI (end of message indicator), and ESI (end of segment indicator) are conceptual indications that serve to notify the MCS that a specific condition exists.
Within the hierarchy of EGI, EMI, ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an EMI or EGI.

Message Segment
Data that forms a logical subdivision of a message normally associated with an end of segment indicator. Refer to Message Indicators.

Mnemonic-Name
A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified name.

MSCS
Refer to Mass Storage Control System.

Native Character Set
The character set associated with the computer for which object code is generated.

Native Collating Sequence
The collating sequence associated with the native character set.

Negated Combined Condition
The NOT logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition
The NOT logical operator immediately followed by a simple condition.

Network Controller
The Network Controller program is a data communications operating system and is responsible for handling all of the details of the data communications line discipline.

Next Executable Sentence
The next sentence to which control will be transferred after execution of the current statement is complete.

Next Record
The record which logically follows the current record of a file.

Noncontiguous Items
Elementary data items, in the WORKING-STORAGE and LINKAGE SECTIONS, which bear no hierarchic relationship to other data items.

Nonnumeric Item
A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal
A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character
A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item
A data item whose description restricts its contents to a value represented by characters chosen from the digits 0 through 9; if signed, the item may also contain a +, -, or other representation of an operational sign.

Numeric Literal
A literal composed of one or more numeric characters that also may contain either a decimal point, an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

OBJECT-COMPUTER
The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the object program is executed, is described.

Object of Entry
A set of operands and reserved words, within a DATA DIVISION entry, that immediately follows the subject of the entry.

Object Program
A set or group of executable instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the result of the operation of a COBOL74 compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

Object Time
The time at which an object program is executed.

ODT
Operator Display Terminal.

Open Mode
The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

Operand
Whereas the general definition of operand is 'that component which is operated upon,' for the purposes of this manual, any lower-case word or words that appear in a statement or entry format may be considered to be an operand and, as such, imply reference to the data indicated by the operand.

Operational Sign
An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

NE 3 Optional Word
A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File
A file that is opened in either the output mode or extend mode.

Output Mode
The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Output Procedure
A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

Page
A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

Page Body
That part of the logical page in which lines can be written and/or spaced.

Page Footing
A report group that is presented at the end of a report page.

Page Heading
A report group that is presented at the beginning of a report page.

Paragraph
In the PROCEDURE DIVISION, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONS, a paragraph header followed by zero, one, or more entries.

Paragraph Header
A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONS. The permissible paragraph headers are:

In the IDENTIFICATION DIVISION:
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN
DATE-COMPILED.
SECURITY.

In the ENVIRONMENT DIVISION:
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.

Paragraph-Name
A user-defined word that identifies and begins a paragraph in the PROCEDURE DIVISION.

Phrase
A phrase is an ordered set of one or more consecutive COBOL74 character-strings that form a portion of a COBOL74 procedural statement or of a COBOL74 clause.

Physical Record
Refer to Block.

Port File
A type of file used by the Burroughs Network Architecture (BNA) system to provide paths between two or more processes.

Prime Record Key
A key whose contents uniquely identify a record within an Indexed File.

Procedure
A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the PROCEDURE DIVISION.

Procedure-Name
A user-defined word which is used to name a paragraph or section in the PROCEDURE DIVISION. It consists of a paragraph-name (which may be qualified) or a section-name.

Program-Name
A user-defined word that identifies a COBOL74 source program.

Pseudo-Text
A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters.

Pseudo-Text Delimiter
Two contiguous equal sign (=) characters used to delimit pseudo-text.

Punctuation Character
A character that belongs to the following set:

Character	Meaning
,	Comma
;	Semicolon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space
=	Equal sign
@	Commercial at

Qualified Data-Name
An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier

1. A data-name which is used in a reference together with another data-name at a lower level in the same hierarchy.

2. A section-name which is used in a reference together with a paragraph-name specified in that section.

3. A library-name which is used in a reference together with a text-name associated with that library.

Queue
A logical collection of messages awaiting transmission or processing.

Queue Name
A symbolic name that indicates to the MCS the logical path by which a message or a portion of a completed message may be accessible in a queue.

Random Access
An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a Relative or Indexed file.

Record
Refer to Logical Record.

Record Area
A storage area allocated for the purpose of processing the record described in a record description entry in the FILE SECTION.

Record Description
Refer to Record Description Entry.

Record Description Entry
The total set of data description entries associated with a particular record.

Record Key
A key, either the prime record key or an alternate record key, whose contents identify a record within an Indexed File.

Record-Name
A user-defined word that names a record described in a record description entry in the DATA DIVISION.

Reference Format
A format that provides a standard method for describing COBOL74 source programs.

Relation
Refer to Relational Operator.

Relation Character
A character that belongs to the following set:

Character	Meaning
>	Greater than
<	Less than
=	Equal to

Relation Condition
The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item. Refer to Relational Operator.

Relational Operator
A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

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

Relative File
A file with relative organization.

Relative Key
A key whose contents identify a logical record in a Relative File.

Relative Organization
The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word
A COBOL74 word specified in the list of words which may be used in COBOL74 source programs, but which must not appear in the programs as user-defined words.

Resultant-Identifier
A user-defined data item that is to contain the result of an arithmetic expression.

Run Unit
A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

Section
A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header
A combination of words followed by a period and a space that indicates the beginning of a section in the ENVIRONMENT, DATA, and PROCEDURE DIVISIONS.
In the ENVIRONMENT and DATA DIVISIONS, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the ENVIRONMENT DIVISION:
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

In the DATA DIVISION:
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
COMMUNICATION SECTION.

In the PROCEDURE DIVISION, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name
A user-defined word which names a section in the PROCEDURE DIVISION.

Segment-Number
A user-defined word which classifies sections in the PROCEDURE DIVISION for purposes of segmentation. Segment-numbers may contain only the characters "0", "1", ..., "9". A segment-number may be expressed as a 1-, 2-, 3-, or 4-digit number.

Sentence
A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator
A punctuation character used to delimit character-strings.

Sequential Access
An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of the records in the file.

Sequential File
A file with sequential organization.

Special Character
A character that belongs to the following set:

Character	Meaning
+	Plus sign
–	Minus sign
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol
@	Commercial at

Special-Character Word
A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES
The name of an ENVIRONMENT DIVISION paragraph in which names are related to user specified mnemonic-names.

Special Registers
Compiler-generated storage areas whose primary use is to store information produced in conjunction with the user of specific COBOL74 features.

Standard Data Format
Describes the characteristics of data in a COBOL74 DATA DIVISION under which the characteristics or properties of the data are expressed by the appearance of the data on a printed page of infinite length and breadth. Standard Data Format is not a form oriented to the manner in which data is stored internally in the computer, or on a particular external medium.

Statement
A syntactically valid combination of words and symbols written in the PROCEDURE DIVISION beginning with a verb.

Sub-Queue
A logical hierarchical division of a queue.

Subfile
The unique connection to an individual process in a Burroughs Network Architecture (BNA) port file.

Subject of Entry
An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

Subprogram
Refer to Called Program.

Subscript
An integer whose value identifies a particular element in a table.

Subscripted Data-Name
An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Switch-Status Condition
The proposition, for which a truth value can be determined, that a switch, capable of being set to an 'on' or 'off' status, is set to a specific status.

System Name
A COBOL74 word which is used to communicate with the operating environment.

Table
A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

Table Element
A data item that belongs to the set of repeated items comprising a table.

Terminal
The originator of a transmission to a queue, or the receiver of a transmission from a queue.

Text-Name
A user-defined word which identifies library text.

Text-Word
Any character-string or separator, except space, in a COBOL74 library or in pseudo-text.

Truth Value
The representation of the result of the evaluation of a condition in terms of one of two values:

TRUE
FALSE

Unary Operator
A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression of +1 or -1, respectively.

Unit
A module of mass storage.

User-Defined Word
A COBOL74 word that must be supplied by the user to satisfy the format of a clause or statement.

Variable
A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb
A word that expresses an action to be taken by a COBOL74 compiler or object program.

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

WORKING-STORAGE SECTION
The section of the DATA DIVISION that describes working-storage data items, composed either of noncontiguous items or of working-storage records or of both.

77-Level-Description-Entry
A data description entry that describes a noncontiguous data item with the level-number 77.

APPENDIX E
COBOL74 S-LANGUAGE

GENERAL

B 1000 COBOL74 S-language provides the virtual machine interface between the code generated by the COBOL74 compiler and the COBOL74 interpreter. This appendix includes a description of the format of COBOL74 S-instructions and an explanation of each operator as a member of one of the following classes:

ARITHMETIC
DATA MOVEMENT
BRANCHING
CONDITIONAL BRANCHING
MISCELLANEOUS
CHARACTER STRINGS
INTERPROGRAM COMMUNICATION

S-LANGUAGE PROGRAMS

All COBOL74 S-language programs have a base register and a limit register associated with them. The area between the base and the limit is to be used as data space only. All program code, organized in segment form, is stored at any available location in memory, according to the memory management algorithms used by the B operating system.

The data space includes a nonoverlayable area which contains various parameters such as edit masks, literals, and record areas.

Various parameters, necessary for the running of the S-language object code are stored beyond the Limit Register in the Run Structure Nucleus (RSN).

A typical COBOL74 program layout in memory is shown in figure E-1.

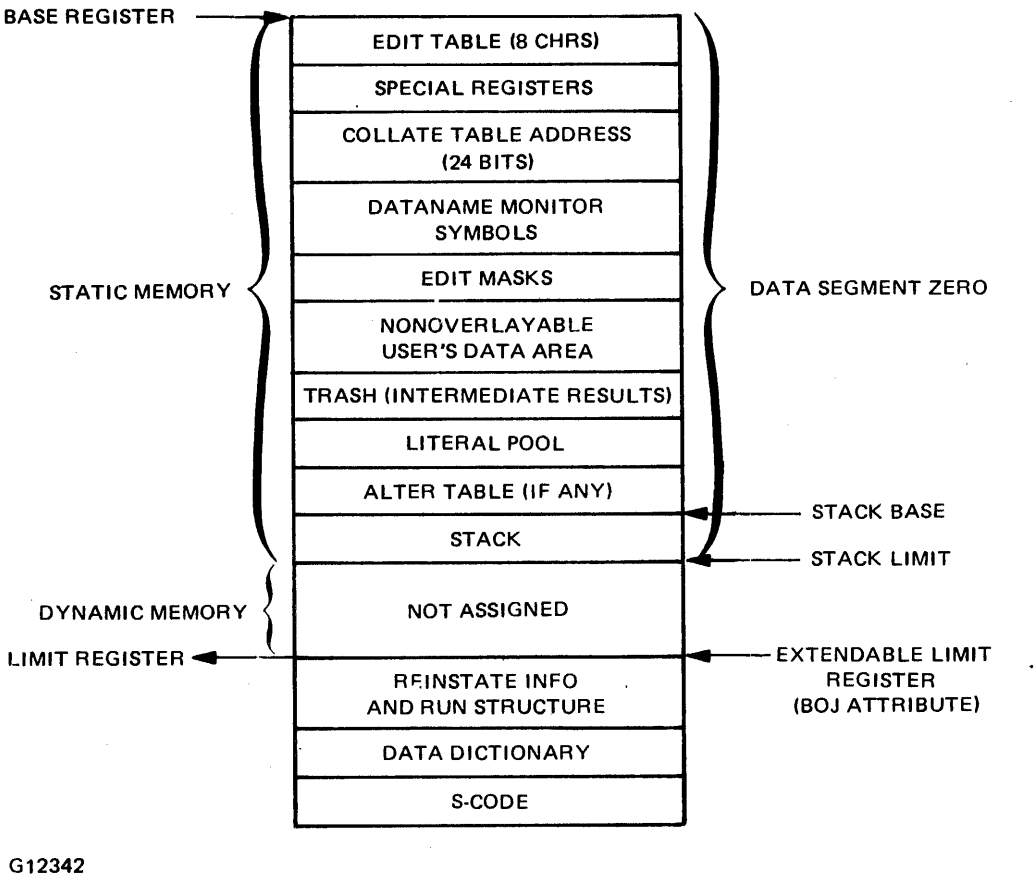


Figure E-1. COBOL74 Program Layout

Special registers are listed in Table E-1.

Table E-1. Special Registers

ADDRESS	NAME	PICTURE
0	SW1	9 CMP
:	:	:
7	SW8	9 CMP
49	DM-STATUS	9 (2)

CONTAINER SIZE

Container size is a field size in bits necessary to contain the maximum value required for that field. For example, a container size of five bits allows a field size to house a 32-bit address (0-31).

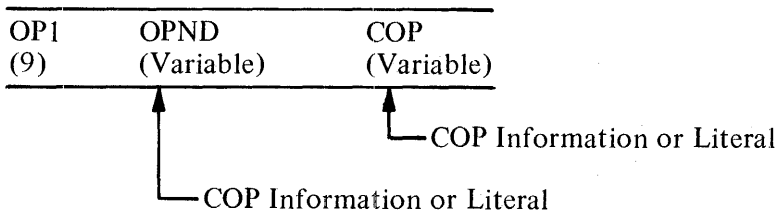
All container sizes in the COBOL74 S-Machine are fixed length and shown in table E-2.

Table E-2. Container Sizes

Abbreviation	Container	Length
OP	Opcode	9 bits
LEN	Data length op	14 bits
DISP	Data displacement	20 bits
SEG	Data segment number	10 bits
BADDR	Branch address	21 bits
DADDR	Data address	20 bits

S-INSTRUCTION FORMAT

Each COBOL74 S-instruction consists of an S-operator followed by arguments consisting of a variable number of bits. The format and interpretation of these arguments are specified by the S-operator and are described in detail by the specification of the individual operators. An example of one such instruction format is illustrated below.



S-Operators

All S-operators are encoded in a 9-bit S-operator field denoted as OP1.

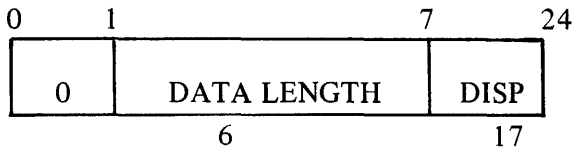
COP and OPND

A COP is an inline descriptor pointing to a data item or a literal. An operand (OPND) may be an inline COP or an inline literal. All data items are placed in data segment zero.

There are three types of descriptors: a short COP, a long COP with segment numbers, and a long COP without segment numbers.

Short COP

Format:

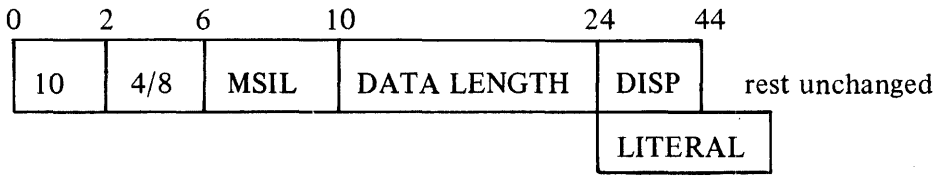


A short COP cannot be generated for subscripts, indexes, IPC data, or literals. Because a short COP does not contain information on data types, it can be used only with the following S-operators:

MVA	CPN
MVN	CPZ
MVZ	INC
CPA	INC1

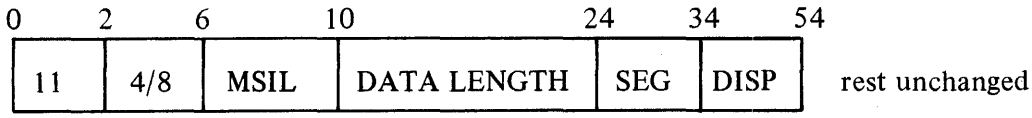
Long COP with No Segment Number

Format:



This COP can be used for any data except IPC data. A description of the fields is found in the following section on In-line Descriptors.

Long COP with Segment Number



This COP can be used for IPC data only. It cannot be used for literals. Descriptions of the fields are provided in the following section on In-line Descriptors.

COBOL74 IN-LINE DESCRIPTORS

An in-line descriptor must provide for the following capabilities:

1. Multidimensional tables with unlimited subscripting/indexing (the real limit is 48).
2. Run-time bounds-checking on each dimension of a table.
3. Variable length data items with an OCCURS DEPENDING clause.
4. Shared data addressing.
5. The specification of CODE-SET.

A variable length data item may be declared in COBOL74 by using an OCCURS DEPENDING clause.

Example:

```
01 A.
 02 B PIC X.
 02 C OC 5 TO 10 TIMES DEPENDING ON V.
  03 D PIC X OC 20 TIMES.
```

In this data structure, A is a variable length item whose length is determined as follows:

LENGTH [A] = LENGTH [B] + (V * LENGTH [C])

where 5 LEQ V LEQ 10

When A is referenced, only that part of it defined by the calculated length is used in any operation.

When C or D is referenced, the length is not variable, but V must be bound-checked to verify that the requested table element exists.

The Restrictions:

1. A data description entry that contains an OCCURS DEPENDING clause may only be followed by entries that are subordinate to it.
2. The OCCURS DEPENDING clause cannot be specified in a data description entry that describes an item whose size is variable.

These restrictions ensure that the description of the first subscript/index associated with a variable is the only one that contains an OCCURS DEPENDING clause.

Implementation Strategy

COBOL74 S-language employs a continue flag to tell the interpreter that more COP information is specified.

The same bounds-checking information is needed when referencing either a variable length item or the first dimension of a subscripted variable contained in a variable length item. Dimension information required for the subsequent dimensions is different from that which is required for the first dimension.

When a variable length item is referenced, the COP of the OCCURS DEPENDING operand is provided following the COP of data-name. The operand is bound-checked to verify that it is within the specified integer range.

MULTIPLE-ENTRY-FLAG

The MULTIPLE-ENTRY-FLAG is TRUE when indicating that multiple entry attributes are associated with this entry and FALSE when indicating otherwise. When TRUE, the next entry(s) contain(s) the necessary OCCURS DEPENDING and/or subscripting or indexing attribute information.

SHARED-DATA-FLAG

The SHARED-DATA-FLAG is TRUE when indicating that this entry was a data item passed by a CALL statement of another program and FALSE when indicating otherwise. When TRUE, IPCD-INDEX specifies the location of the IPC.DICTIONARY descriptor for this entry.

LITERAL-FLAG

If this flag is set, then a literal follows after the DATA LENGTH field.

Data Length

The data length field is 14 bits wide and contains the length of the data in digits. If this field is the length of a variable length item, the data length field contains the length of that part of the table which is not variable. The actual length is calculated at run time by the interpreter (refer to Depending Attributes in this section).

Segment Number

Segment number is expressed in binary and specifies the data segment number of the operand. It is 10 bits in length.

Displacement

Displacement is expressed in binary and specifies the digit displacement of the data from the base of the data segment. All data is stored beginning at an address which modulo 4-bit must equal zero. The container size is 20 bits.

DEPENDING-FLAG

The DEPENDING-FLAG is TRUE when indicating that an OCCURS DEPENDING operand is associated with this variable and FALSE when indicating otherwise. When TRUE, the bounds-checking and factor information follow in the entry. This is followed by any subscripting or indexing information that may be necessary.

Depending Attributes

When the DEPENDING-FLAG is TRUE, the attributes of the DEPENDING operand are encoded in LOWER-BOUND-0, UPPER-BOUND-0, and FACTOR-0, which are binary fields. LOWER-BOUND-0 and UPPER-BOUND-0 contain the integer range of the DEPENDING operand while FACTOR-0 contains the digit displacement between elements of the table.

The value of the DEPENDING operand is verified to be greater than or equal to LOWER-BOUND-0 and less than or equal to UPPER-BOUND-0. If the value is outside the range, then an error communicate is issued.

If CONTINUE-FLAG-0 is FALSE, then the value of the DEPENDING operand is multiplied by FACTOR-0 and is added to the data length specified in the primary COP entry, and decoding of the multiple entry attributes terminates. In this case, the data length specified in the COP entry is the fixed part of the table.

If CONTINUE-FLAG-0 is TRUE, then subscripting or indexing information follows. One is subtracted from the value of the DEPENDING operand and the result is multiplied by FACTOR-0 to form a new upper-bound. In this case, FACTOR-1 and UPPER-BOUND-1 are omitted from the entry because the equivalence of those factors is represented by FACTOR-0 and NEW.UPPER.BOUND.

SUBSCRIPT-FLAG

If the SUBSCRIPT-FLAG is FALSE, it specifies that indexing information follows and that the factor associated with each dimension of the table is omitted. If it is TRUE, it specifies that subscripting information follows for each dimension of the table.

Factor and upper-bound fields are binary. Factor is the digit displacement between elements of the table. Upper-bound is the maximum digit displacement allowed for this dimension of the table. UPPER-BOUND-O differs from the other upper-bound fields in that it represents the maximum value of the DEPENDING operand.

Subscripting

Subscripting requires a factor and an upper-bound for each dimension of the variable. The continue-flag is TRUE until the last dimension is reached. When the continue-flag becomes FALSE, decoding is terminated.

Each subscript value is obtained and 1 is subtracted from it. If the result is less than zero, an error communicate is issued; otherwise, the result is multiplied by the associated factor and the product is compared to the corresponding upper-bound. If the product exceeds the upper-bound, an error communicate is issued; otherwise, the product is added to the address displacement.

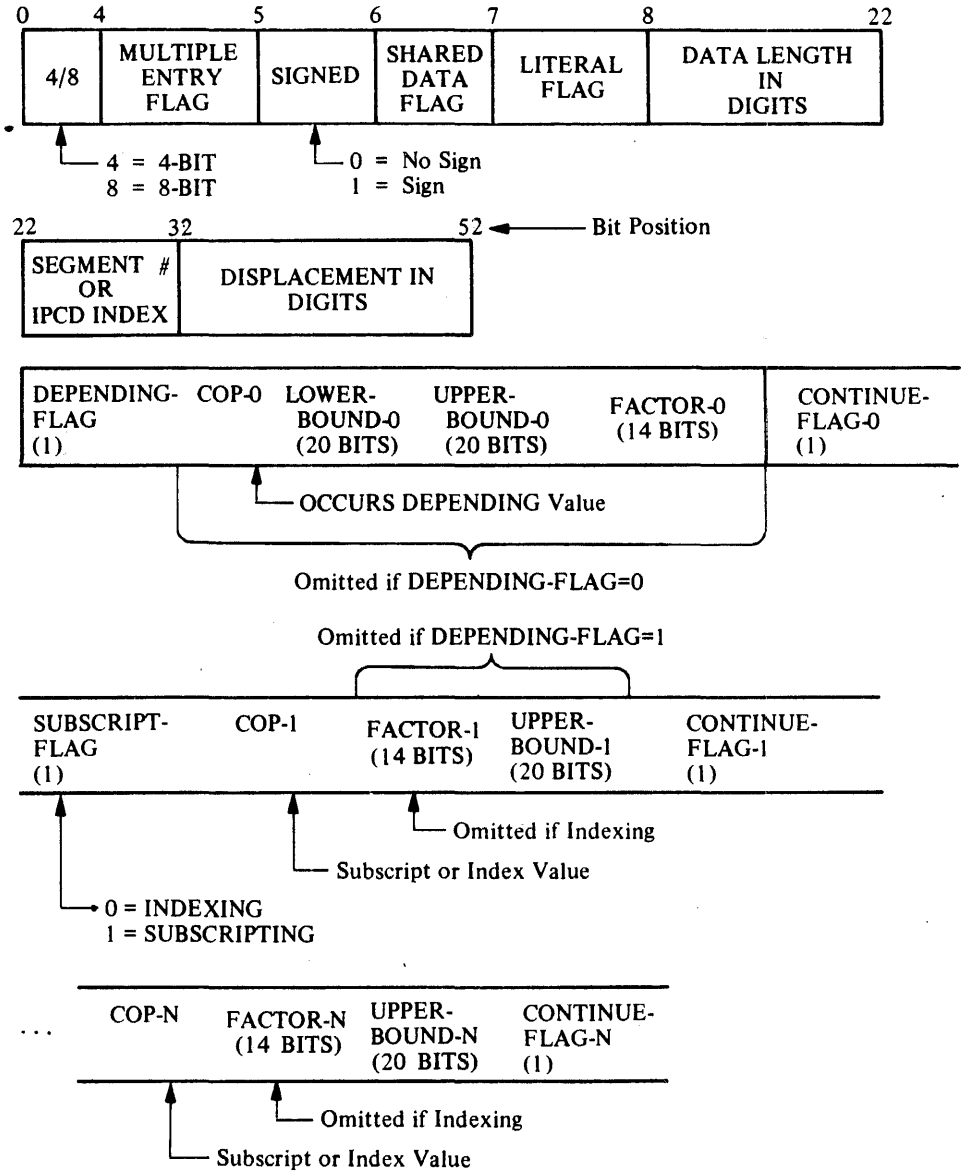
Indexing

Because an index value is premultiplied by the associated factor, the factor required for subscripting is omitted from the attributes. An upper-bound is required for each dimension of the variable. The continue-flag is TRUE until the last dimension is reached. When the continue-flag becomes FALSE, decoding is terminated.

Each index value is obtained, and if it is less than zero or greater than the upper-bound, an error communicate is issued; otherwise, the value is added to the address displacement. The format of the index consists of a 4-bit sign followed by seven 4-bit decimal digits.

In-line COP Entry Format

The format of an in-line COP entry is as follows:



INSTRUCTION SET

The following list contains the name, mnemonic, and arguments for each instruction in the COBOL74 S-Language which are discussed in more detail within this appendix.

Arithmetic

Name	Mnemonic	Arguments
INCREMENT	INC	OPND1, COP1
ADD	ADD	OPND1, COP1, COP2
DECREMENT	DEC	OPND1, COP1
SUBTRACT	SUB	OPND1, OPND2, COP1
MULTIPLY	MULT	OPND1, COP1, COP2
DIVIDE	DIV	OPND1, COP1, COP2
DIVIDE SPECIAL	DIVS	OPND1, COP1, COP2
INCREMENT BY ONE	INC1	COP1
DECREMENT BY ONE	DEC1	COP1

Data Movement

Name	Mnemonic	Arguments
MOVE ALPHANUMERIC	MVA	COP1, OPND1
MOVE SPACES	MVS	COP1
MOVE NUMERIC	MVN	COP1, OPND1
MOVE ZEROS	MVZ	COP1
CONCATENATE	CAT	N, COP1, OPND0,..., OPNDN
EDIT	EDIT	OPND1, COP1, DADDR
EDIT WITH EXPLICIT MASK	EDTE	OPND1, COP1, MASK

Branching

Name	Mnemonic	Arguments
BRANCH ON OVERFLOW	BOFL	V, BADDR
SET OVERFLOW	SOFL	V
BRANCH UNCONDITIONALLY	BUN	BADDR
PERFORM ENTER	PERF	K, BADDR
PERFORM EXIT	PXIT	K
ENTER	NTR	BADDR
EXIT	XIT	
GO TO DEPENDING	GOTD DBADDRL	COP1, L, DBADDR0,...,
ALTERED GO TO PARAGRAPH	GPAR	DADDR
ALTER	ALTR	DADDR, ACON

Conditional Branching

Name	Mnemonic	Arguments
COMPARE ALPHANUMERIC	CMPA	OPND1, OPND2, R, BADDR
COMPARE NUMERIC	CMPN	OPND1, OPND2, R, BADDR
COMPARE FOR ZEROS	CMPZ	COP1, R, BADDR
COMPARE FOR SPACES	CMPS	COP1, R, BADDR
COMPARE FOR CLASS	CMPC	COP1, C, BADDR
COMPARE REPEAT	CMPR	OPND1, COP1, R, BADDR
COMPARE COLLATE	CPC	OPND1, COP2, R, BADDR

Miscellaneous

Name	Mnemonic	Arguments
COMMUNICATE	COMM	COP1
LOAD COMMUNICATE	REPLY LDCR	DADDR
CONVERT	CONV	COP1, DADDR, N
MAKE PRESENT	MAKP	COP1, DADDR
FILE STATUS	FIST	RW, COP1

Character String Handling

Name	Mnemonic	Arguments
DESCRIPTOR SETUP	DSET	DADDR, COP1
INSPECT SETUP	ISET	V1, V2, FLAC, DADDR1, COP1, COP2, DADDR2, OPND1
INSPECT	INSP	V1, DADDR1, DADDR2
STRING	STRI	Z, OPND1, DADDR1, COP1, OPND2, BADDR
DELIMITER SETUP	DLIM	V1, V2, DADDR1, COP1
UNSTRING	UNST	F, M, Z, C, J, BADDR, DADDR1, COP1, DADDR2 .....

Interprogram Communication

Name	Mnemonic	Arguments
IPC DICTIONARY	IPCD	V, COP1, BADDR1

Optimized Operation Codes

Eight operators have been optimized for improved performance: CPA, CPN, CPZ, INC, INC1, MNA, MNV, and MVZ. The optimized OP codes include the following data as a part of the OP code:

1. The type of data item (8-bit or 4-bit).
2. Signed or unsigned data.
3. Information about the logical size of operands if the OP code has two operands.

The following notation is used to describe the OP codes.

Symbol	Meaning
8	8-bit or 4-bit data item.
S	Signed or unsigned data item.
=	TRUE if the LOGICALSIZES of two operands are equal.
>	TRUE if the LOGICALSIZE (OPND1) is greater the LOGICALSIZE (OPND2).

CPA

OP code: @(1)011100@ CAT SS =

The LOGICALSIZE of the second operand is greater than the

LOGICALSIZE of the first operand.

CPN

OP code: @(1)0110@ CAT 88SS =

The LOGICALSIZE of the second operand is always greater than the LOGICALSIZE of the first operand.

CPZ

OP code: @(1)0111010@ CAT 8S

INC

OP code: @(1)001@ CAT 88SS CA @(1)11@

INC1

OP code: @(1) 0111100@ CAT 8S

MVA

OP code: @(1)001@ CAT 88SS = >

The first operand is the destination, and the second operand is the source. h3 MVN

OP code: @(1)010@ CAT 88SS= >

The first operand is the destination, and the second operand is the source.

MVZ

OP code: @(1)0111011@ CAT 8S

ARITHMETIC OPERANDS AND INSTRUCTIONS

In general, operands can have any of the following formats:

1. Unsigned 4-BIT.
2. Unsigned 8-BIT.
3. Signed 4-BIT (sign is MSD).
4. Signed 8-BIT (sign over MSD).

Any restrictions concerning the types of operands permitted in an operation are specified under the description of the particular operation.

All fields are addressed by pointing to the most significant bit of the most significant unit which, in the case of a signed field, is the sign.

All fields are considered to be comprised of decimal integers.

The absolute value is stored if the receiving field is unsigned.

Unsigned fields are considered positive.

When a signed format is specified for the receiving field of any arithmetic operation, the sign position is set to 1100 for a positive result and to 1101 for a negative result.

4-BIT operands are interpreted in units of 4 bits. When a signed operand is specified, the sign is interpreted as a separate and leading (leftmost) 4-BIT unit which is not included in the statement of length.

8-BIT operands are interpreted in units of 8 bits. When a signed operand is specified, the sign is interpreted as being contained in the leftmost 4 bits of the leftmost 8-BIT unit.

The length of the operand field specifies the number of 4-BIT units.

When 8-BIT units are specified for the receiving field of an arithmetic operation, the leftmost 4 bits of each 8-BIT unit, except the unit carrying a sign, is set to 1111.

The value of an 8-BIT unit is carried in the rightmost 4 bits of the unit. Its value is as defined below for the 4-BIT unit. The leftmost 4 bits, except for a sign, are ignored. The value and sign interpretation of a 4-BIT unit are as follows:

4-BIT Unit	Value	Sign
0000	0	+
0001	1	+
0010	2	+
0011	3	+
0100	4	+
0101	5	+
0110	6	+
0111	7	+
1000	8	+
1001	9	+
1010	Undefined	+
1011	Undefined	+
1100	Undefined	+
1101	Undefined	−
1110	Undefined	+
1111	Undefined	+

In addition and subtraction operations, results generated when the size of the result field is not sufficient to contain the result are not specified. When the result field is longer than the length of the result, leading zero units are stored.

In three-address add, three-address subtract, and multiply operations, total or partial overlap of the first two operands is permitted. Results generated when the result field totally or partially overlaps either of the operand fields, are not specified.

In two-address add and subtract, total overlap is permitted. Results generated when the result field partially overlaps the first operand field are not specified. Total overlap implies that the two fields are identical.

No overlap of operands or result fields is permitted in divide operations. Results generated under any condition of overlap are not specified.

COBOL74 S-LANGUAGE

ADD THREE ADDRESS

ADD

OP: 08

Format:

ADD OPND1, COP1, COP2

Function:

Algebraically add an addend denoted by OPND1 to an augend denoted by COP1 and store the sum in the field denoted by COP2. OPND1, COP1, and COP2 must be 4-bit items.

SUBTRACT THREE ADDRESS

SUB

OP: 10

Format:

SUB OPND1, OPND2, COP1

Function:

Algebraically subtract a subtrahend denoted by OPND1 from a minuend denoted by OPND2 and store the difference in the field denoted by COP1. OPND1, OPND2, and COP1 must be 4-bit items.

COBOL74 S-LANGUAGE

ADD TWO ADDRESS

INC

Format:

INC OPND1, COP1

Function:

Algebraically add an addend denoted by OPND1 to an augend denoted by COP1 and store the sum in the field denoted by COP1.

INC is an optimized S-OP. OPND1 and COP1 must have the same data unit type. OPND1 and COP1 both must be either 4-bit data items or 8-bit data items. The format of the OP code is 0108855. For example, if both OPND1 and COP1 are unsigned 8-bit items, then the OP code is 010110011 or 179.

SUBTRACT TWO ADDRESS

DEC

OP: 09

DEC OPND1, COP1

Function:

Algebraically subtract a subtrahend denoted by OPND1 from a minuend denoted by COP1 and store the difference in the field denoted by COP1. OPND1 and COP1 must be 4-bit items.

COBOL74 S-LANGUAGE

MULTIPLY

MULT

OP: 11

Format:

MULT OPND1, COP1, COP2

Function:

Algebraically multiply a multiplicand denoted by COP1 by a multiplier denoted by OPND1 and store the product in the field denoted by COP2.

The result field length is the sum of the lengths of the two operands and must be denoted by COP2. OPND1 and COP1 must be 4-bit items.

The result field is always either a signed 4-BIT format or an unsigned 4-BIT format.

DIVIDE

DIV

OP: 12

Format:

DIV OPND1, COP1, COP2

Function:

Algebraically divide a dividend denoted by COP1 by a divisor denoted by OPND1 and store the quotient in the field denoted by COP2. Store the remainder in the field denoted by COP1.

The result field length is the difference of the lengths of the two operands and must be denoted by COP2.

Results are not specified if the length of the dividend is not greater than the length of the divisor.

If the absolute value of the divisor is not greater than the absolute value of an equivalent number of leading digits of the dividend, the result is undefined.

Division by zero results in a fatal error communicate to the MCP.

OPND1, COP1, and COP2 must be 4-bit items.

The sign of the remainder is that of the original dividend.

The dividend field is always either signed 4-BIT format or unsigned 4-BIT format.

COBOL74 S-LANGUAGE

DIVIDE SPECIAL

DIVS

OP: 16

Format:

```
DIVS OPND1, COPX1, COPX2
```

Function:

This operation is performed in exactly the same manner as the standard divide (DIV) operator, except that when a divisor equal to zero is encountered, an overflow toggle is set and processing is allowed to continue. The overflow toggle can be manipulated by the SOFL and BOFL S-operators.

INCREMENT BY ONE

INC1

OP: 240-243

Format:

INC1 COP1

Function:

Algebraically add the positive integer 1 to an augend denoted by COP1 and store the sum in the field specified by COP1.

INCL is an optimized S-OP and the OP code includes information regarding the data type of COP1. COP1 may be a short COP: the format of the OP code is 01111008S.

OP Code	Type of Data	8S
240	4-bit unsigned	00
241	4-bit signed	01
242	8-bit unsigned	10
243	8-bit signed	11

COBOL74 S-LANGUAGE

DECREMENT BY ONE

DEC1

OP: 14

Format:

DEC1 COP1

Function:

Algebraically subtract the positive integer 1 from a minuend denoted by COP1 and store the difference in the field specified by COP1. COP1 must be a 4-bit item.

DATA MOVEMENT OPERANDS AND INSTRUCTIONS

In general, fields involved in data movement operations can have any of the following formats:

1. Unsigned 4-BIT.
2. Unsigned 8-BIT.
3. Signed 4-BIT (sign is MSD).
4. Signed 8-BIT (sign over MSD).

Any restrictions as to the type of fields permitted in an operation are specified under the description of the particular operation.

Refer to Arithmetic Operands and Instructions in this appendix for a description of the four types of fields.

Totally or partially overlapped fields are not permitted, unless specified by the description of the individual instruction.

COBOL74 S-LANGUAGE

MOVE ALPHANUMERIC

MVA

OP: 64-127 at:

Format:

MVA COP1, OPND1

Function:

Move 8-BIT or 4-BIT units from the source field denoted by OPND1 to the 8-BIT or 4-BIT destination field denoted by COP1.

If the destination field is signed, it receives either the sign of the source if the source is signed, or 1100 if the source is unsigned.

If the data type of the source field is 4-BIT and the data type of the destination field is 8-BIT, each 4-BIT unit is moved to the destination with 1111.

If the data type of the source field is 8-BIT and the data type of the destination is 4-BIT, the rightmost 4 bits are moved.

If the data type of the source field is the same as the data type of the destination field, each unit is moved unchanged to the destination.

If the destination length is greater in size than the source length, the destination field is filled in on the right with trailing spaces (0100 0000) if the destination type is 8-BIT; otherwise, it is filled in on the right with zeros (0000).

If the destination length is lesser in size than the source length, the source data is truncated on the right.

Overlapping operand fields are permitted if the data type of both fields is the same. It can be assumed that the source is moved 24 bits (six digits or three characters) at a time into the destination field and that the move is from left to right.

MVA is an optimized S-OP, and COP1 and OPND1 may be short COP descriptors. The OP code contains information regarding the data types. The format of the OP code is 00188SS = >. For example, if both source and destination fields are signed 8-bit items of equal length, then the OP code is 001111110, or 126.

MOVE SPACES

MVS

OP: 15

Format:

MVS COP1

Function:

Fill the destination field denoted by COP1 with spaces (0100 0000).

The data type of the destination field is ignored and is assumed to be unsigned 8-BIT.

COBOL74 S-LANGUAGE

MOVE NUMERIC

MVN

OP: 128-191

Format:

MVN COP1, OPND1

Function:

Move 8-BIT or 4-BIT units from the source field denoted by OPND1 to the 8-BIT or 4-BIT destination field denoted by COP1.

If the destination field is signed, it receives either the sign of the source if the source is signed, or 1100 if the source is unsigned.

If the destination field is unsigned, the sign of the source is ignored.

If the data type of the destination field is 8-BIT, the leftmost 4 bits of each 8-BIT unit, except for the sign position, if signed, are set to 1111 regardless of the data type of the source field.

If the data type of the destination field is 4-BIT, the leftmost 4 bits of each source 8-BIT unit are ignored and only the rightmost 4 bits are moved; if the source field is a 4-BIT field, each 4-BIT unit is moved unchanged.

If the destination length is greater in size than the source length, the destination field is filled in on the left with leading zeros of appropriate type (1111 0000).

If the source length is greater in size than the destination length, the source data is truncated on the left.

A sign is placed in the leftmost 4 bits of a field, whether 4-BIT or 8-BIT.

Overlapping operand fields are permitted if the data type of both fields is the same. It can be assumed that the source is moved 24 bits (six digits or three characters) at a time into the destination field and that the move is from left to right.

MVN is an optimized S-OP. Consequently, COP1 and OPND1 may be short COP descriptors. The OP code contains information regarding the data types. The format of the OP code is 01088SS =>. For example, if both source and destination fields are unsigned and the source field is 4-BIT data and the destination field is 8-BIT data and the destination length is greater in size than the source length, then the OP code is 010100001, or 161.

MOVE ZEROS

MVZ

OP: 236-239

Format:

MVZ COP1

Function:

Fill the destination field denoted by COP1 with zeros of the appropriate type (1111 0000 or 0000 if 4-BIT).

If the destination field is signed, 1100 is placed into the sign position.

MVZ is an optimized S-OP. COP1 can be a short COP. The format of the OP code is 01110118S.

Opcode	Data Type	8S
236	4-BIT unsigned	00
237	4-BIT signed	01
238	8-BIT unsigned	10
239	8-BIT signed	11

COBOL74 S-LANGUAGE

CONCATENATE

CAT

OP: 32

Format:

CAT N, COP1, OPND0, ..., OPNDN

Function:

Move each of the N+1 fields denoted by OPND0 through OPNDN, in the order specified, into an output string starting at the field denoted by COP1.

The number of source fields is specified by the 4-BIT binary value N. The value N ranging from 0000 to 1111 is used to indicate 1 to 16 source fields.

Each field is moved according to the rules specified for MOVE ALPHANUMERIC.

If the destination length is greater in size than the combined source length, the destination field is filled on the right with trailing spaces (0100 0000).

If the destination length is lesser in size than the combined source lengths, the source data is truncated on the right.

EDIT INSTRUCTIONS AND EDIT MICRO-OPERATORS

No restrictions are placed on the data type of the source field of an edit operation.

The data type of the destination field of an edit operation must be unsigned 8-BIT.

If the destination length is greater in size than the source length, the source data is assumed to have leading zero fill on the left.

If the source length is greater in size than the destination length, the source data is truncated on the left.

The operation is terminated by an edit micro-operator and not by exhaustion of either the source or destination fields.

COBOL74 S-LANGUAGE

EDIT

EDIT

OP: 17

Format:

EDIT OPND1, COP1, DADDR

Function:

Move data from the source field, denoted by OPND1, to the destination field, denoted by COP1, under the control of the micro-operator string contained at the location denoted by the DADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the micro-operator string relative to the data segment zero base. The container size of DADDR is DISPB.

EDIT WITH EXPLICIT MASK

EDTE

OP: 21

Format:

```
EDTE OPND1, COP1, MASK
```

Function:

Move data from the source field denoted by OPND1 to the destination field denoted by COP1 under the control of the micro-operator string immediately following COP1. The format of the explicit micro-operator string is the same as a literal.

COBOL74 S-LANGUAGE

EDIT MICRO-OPERATORS

The edit micro-operators used in an edit instruction are:

Operator	Mnemonic	Operation
0000 R	MVD	Move digits
0001 R	MVC	Move characters
0010 R	MVS	Move suppress
0011 R	FIL	Fill suppress
0100 N	SRD	Skip reverse destination
0101 T	INU	Insert unconditionally
0110 T	INM	Insert on minus
0111 T	INS	Insert suppress
1000 T	INF	Insert float
1001 T	EFM	End float mode
1010 0000	ENZ	End nonzero
1010 0001	EOM	End of mask
1010 0010	SZS	Start zero suppress
1010 0011	CCP	Complement check protect
OTHERS		Undefined

"R" indicates a 4-BIT binary value used as a repeat count. The value 0000 represents no repeat; do it once.

"N" indicates a 4-BIT binary value used to skip over a number of destination 8-BIT units. The value 0000 represents no skip.

"T" indicates a 4-BIT binary value which is:

1. Used to index into a table of editing constants.
2. Used to indicate a conditional selection between two table constants.
3. Used to indicate an editing constant in line with the edit-operator string.

The next edit-operator follows the constant.

The following table indicates the normal table editing constants as well as the conditional and unconditional selection of constants associated with the value of T.

EDITING CONSTANTS

T	Table Entry EBCDIC	Mnemonic	Unconditional or Conditional Constant
0000	"+"	PLU	
0001	"-"	MIN	
0010	"*"	AST	
0011	"."	DPT	
0100	","	CMA	
0101	"$"	CUR	
0110	"0"	ZRO	
0111	" "	BLK	
1000		SPM	Either entry 0 or 1
1001		SBM	Either entry 7 or 1
1010		LIT	In-line 8-BIT constant

Associated with the edit instructions are three toggles denoted as S for sign, Z for zero suppress, and P for check protect. Initially, the Z and the P toggles are assumed to be set to the zero state and are set and reset as specified by the description of the individual micro-operators. The S toggle is set to 0 if the source field sign is positive, and to 1 if the source field sign is negative. Unsigned fields are considered positive.

MOVE DIGIT

Set Z to 1, to end the zero suppress state. Move an appropriate unit (4-BIT digit or 8-BIT character) from the source field to the destination field. If a 4-BIT unit is moved, append the four bits (1111) to the left before storing in the destination. If an 8-BIT unit is moved, the four bits (1111) are substituted for the leftmost four bits of the 8-BIT unit.

MOVE CHARACTER

Set Z to 1, to end the zero suppress state. Move an appropriate unit (4-BIT digit or 8-BIT character) from the source field to the destination field. If a 4-BIT unit is moved, append the four bits 1111 to the left before storing in the destination. If an 8-BIT unit is moved, it is moved unchanged.

MOVE SUPPRESS

The micro-operator MOVE DIGIT is performed if the 4-BIT unit, or the rightmost four bits of the 8-BIT unit of the source field are not equal to 0000.

If the appropriate four bits of the source field unit are equal to 0000 the suppress toggle Z is inspected. If Z equals 1, indicating nonsuppress mode, the micro-operator MOVE DIGIT is performed. If the suppress toggle Z equals 0, the check protect toggle P is inspected. If P = 0, indicating noncheck protect mode, move the table entry containing the 8-BIT code for blank to the destination field. If P = 1, move the table entry containing the 8-BIT code for asterisk to the destination field.

```
      SOURCE NOT   = 0 Move digit
    Z=1 SOURCE     = 0 Move digit
Z=0 P=0 SOURCE     = 0 Move table entry 7 (Blank)
Z=0 P=1 SOURCE     = 0 Move table entry 2 (Asterisk)
```

FILL SUPPRESS

If P = 0, indicating noncheck protect mode, move the table entry containing the 8-BIT code for blank to the destination field. If P = 1, move the table entry containing the 8-BIT code for asterisk to the destination field.

P = 0 Move table entry 7 (Blank)

P = 1 Move table entry 2 (Asterisk)

SKIP REVERSE DESTINATION

Adjust the address pointer of the destination field to skip backward (lower address) N 8-BIT units.

INSERT UNCONDITIONALLY

Move the table entry T as indicated below to the destination field.

	T=0...7	Move table entry T
S=0	T=8	Move table entry 0 (Plus)
S=1	T=8	Move table entry 1 (Minus)
S=0	T=9	Move table entry 7 (Blank)
S=1	T=9	Move table entry 1 (Minus)
	T=10	Move in-line table entry

INSERT ON MINUS

Move the table entry T as indicated below to the destination field.

S=1	T=0...7	Move table entry T
* P=0		Move table entry 7 (Blank)
* P=1		Move table entry 2 (Asterisk)
S=1	T=8	Move table entry 1 (Minus)
S=1	T=9	Move table entry 1 (Minus)
S=1	T=10	Move in-line table entry

*: S = 0 or only source digits/characters equal to zero (minus zero) have been moved.

INSERT SUPPRESS

Move the table entry T as indicated below to the destination field.

Z=1		T=0...7	Move table entry T
Z=0 P=0			Move table entry 7 (Blank)
Z=0 P=1			Move table entry 2 (Asterisk)
Z=1	S=0	T=8	Move table entry 0 (Plus)
Z=1	S=1	T=8	Move table entry 1 (Minus)
Z=1	S=0	T=9	Move table entry 7 (Blank)
Z=1	S=1	T=9	Move table entry 1 (Minus)
Z=1		T=10	Move in-line table entry

INSERT FLOAT

Move the table entry T and/or perform the micro-operator MOVE DIGIT as indicated below.

Z=1	Move digit
Z=0 SOURCE =0 P=0	Move table entry 7 (Blank)
Z=0 SOURCE =0 P=1	Move table entry 2 (Asterisk)
Z=0 SOURCE NOT=0 T=0..7	Move table entry T, then move digit
Z=0 SOURCE NOT=0 T=8 S=0	Move table entry 0 (Plus) then move digit
Z=0 SOURCE NOT=0 T=8 S=1	Move table entry 1 (Minus) then move digit
Z=0 SOURCE NOT=0 T=9 S=0	Move table entry 7 (Blank) then move digit
Z=0 SOURCE NOT=0 T=9 S=1	Move table entry 1 (Minus) then move digit
Z=0 SOURCE NOT=0 T=10	Move in-line table entry, then move digit

END FLOAT MODE

Move the table entry T as indicated below to the destination field.

Z=0 T=0...7	Move table entry T
Z=0 S=0 T=8	Move table entry 0 (Plus)
Z=0 S=1 T=8	Move table entry 1 (Minus)
Z=0 S=0 T=9	Move table entry 7 (Blank)
Z=0 S=1 T=9	Move table entry 1 (Minus)
Z=0 T=10	Move in-line table entry
Z=1 NO OPERATION	

END NON-ZERO

Terminate the micro-operator operations if any nonzero source character/digit has been moved; otherwise, continue with the next in-line operator.

END OF MASK

Terminate the micro-operator operations.

START ZERO SUPPRESS

Set Z to the zero state.

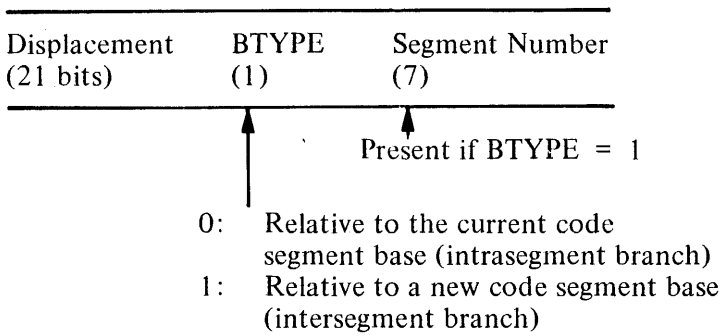
COMPLEMENT CHECK PROTECT

Complement the state of P. H1 BRANCHING OPERANDS AND INSTRUCTIONS

COBOL74 S-LANGUAGE

BRANCHING OPERANDS AND INSTRUCTIONS

A branch address argument BADDR has the following format:



Displacement is an unsigned binary value which specifies the bit displacement of an instruction relative to a segment base. The container size of the displacement and BTYPE combined is a program parameter.

BRANCH UNCONDITIONALLY

BUN

OP: 03

Format:

BUN BADDR

Function:

Obtain the next instruction from the location specified by BADDR.

COBOL74 S-LANGUAGE

BRANCH ON OVERFLOW

BOFL

OP: 23

Format:

BOFL V, BADDR

Function:

If the overflow toggle equals V, a transfer to the address (BADDR) given in the instruction occurs; otherwise, control is passed to the next sequential instruction.

The overflow toggle is unchanged. The length of V is one bit.

SET OVERFLOW TOGGLE

SOFL

OP: 07

Format:

SOFL V

Function:

Set the overflow toggle to V.

The length of V is one bit.

NOTE
The overflow toggle is set to 1 if a DIVIDE BY ZERO is encountered in the DIVIDE SPECIAL S-operator.

COBOL74 S-LANGUAGE

PERFORM ENTER

PERF

OP: 06

Format:

PERF K, BADDR

Function:

Create a stack entry with the following format:

Displacement (24)	Segment No. (7)	K (12)

Insert a displacement value, relative to the active code segment base and pointing to the next sequential S-instruction, into the stack.

Insert the current code segment number into the stack. Insert the value of K from the instruction into the stack.

Adjust the stack pointer to point to the next possible entry.

Obtain the next instruction from the location specified by BADDR.

PERFORM EXIT

PXIT

OP: 34

Format:

```
PXIT K
```

Function:

Compare the K contained in the instruction to the K in the current stack entry and, if unequal, proceed to the next in-line S-instruction. If equal, adjust the stack pointer to point to the previous entry and obtain the next S-instruction from the information contained in the removed stack entry.

COBOL74 S-LANGUAGE

ENTER

NTR

OP: 18

Format:

NTR BADDR

Function:

Same function as PERF. K is assumed to be equal to zero.

EXIT

OP: 19

Format:

XIT

Function:

Same function as PXIT. K is assumed to be equal to zero.

COBOL74 S-LANGUAGE

GO TO DEPENDING

GOTD

OP: 39

Format:

GOTD COP1, L, DBADDR0, ..., DBADDRL

Function:

Compare the 10-bit binary value L with the variable specified by COP1. The variable is first converted to a binary value, Modulo 2 to the 24th power.

If the binary value of the variable is less than zero or greater than L, the next instruction is obtained from the location specified by DBADDR0. The variable can be signed.

If the binary value of the variable is in the range zero through L, it is used as an index to select from the list of DBADDRs the appropriate DBADDR to be used to obtain the next instruction.

DBADDR and BADDR have the same format with the exception that DBADDR always contains the segment number. Although segment number is unnecessary for those DBADDRs with BTYPE equal to zero, in order to index into the list of DBADDRs, all of the DBADDRs must be of equal length. The container size of DBADDR is BDISPB1 + 7.

ALTERED GO TO PARAGRAPH

GPAR

OP: 35

Format:

GPAR DADDR

Function:

Obtain the next instruction from the location specified by the address ACON.

The address constant ACON has the same format as a BADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the ACON relative to the data segment zero base.

The container size of DADDR is DISPB.

COBOL74 S-LANGUAGE

ALTER

ALTR

OP: 36

Format:

```
ALTR DADDR, ACON
```

Function:

Copy the address constant ACON into the data area specified by the argument DADDR.

The address constant ACON has the same format as a BADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the ACON relative to the data segment zero base.

The container size of DADDR is DISPB.

CONDITIONAL BRANCH OPERANDS AND INSTRUCTIONS

If the condition A (R) B is TRUE a transfer to the address (BADDR) given in the instruction occurs; otherwise, control is passed to the next sequential instruction. The relation (R) is defined as follows:

R	Meaning
000	Undefined
001	GTR
010	LSS
011	NEQ
100	EQL
101	GEQ
110	LEQ
111	Undefined

Overlap of fields is permitted. *A* is the first operand denoted in the instruction. If an instruction has only one operand, then the assumed field is the A field.

COBOL74 S-LANGUAGE

COMPARE ALPHANUMERIC

CMPA

OP: 224-231

Format:

CMPA OPND1, OPND2, R, BADDR

Function:

Compare the two operand fields according to binary values.

The comparison is performed left to right with any shorter operand assumed to be right-filled with blank characters (0100 0000 if EBCDIC or 0010 0000 if ASCII).

The fields are considered equal when the equal size portions are equal and the longer field (if one is longer) has trailing blanks.

An 8-BIT data format is assumed for both fields with no checking to verify otherwise. Signed fields have the most significant four bits (the sign) modified to the appropriate numeric zone (1111 for EBCDIC, 0011 for ASCII) before being compared. This modification is not permanent and is done so that the sign does not affect the result of an alphanumeric comparison.

CMPA is an optimized S-OP. Consequently, OPND1 and OPND2 may be short COPs. The LOGICALSIZE of OPND2 is always greater than or equal to the LOGICALSIZE of OPND1. The format of the OP code is 011100SS =.

COMPARE NUMERIC

CMPN

OP: 192-223

Format:

CMPN OPND1, OPND2, R, BADDR

Function:

Compare the two operand fields according to the algebraic values, considering the two fields to be comprised of decimal integers.

When the field sizes are different, the longer field is tested for leading zeros (0000). There is no restriction as to data type. In comparing an 8-BIT character, only the rightmost four bits of the character are considered; the other bits are ignored.

Two fields of all zeros are equal regardless of sign.

Unsigned fields are considered positive. Sign conventions are the same as for arithmetic operands.

Results generated by invalid digit values are undefined.

CMPN is an optimized S-OP. OPND1 and OPND2 can be short COPS. The LOGICALSIZE of OPND1 is always less than the LOGICALSIZE of OPND2. The format of the OP code is 011088SS =. =.

COBOL74 S-LANGUAGE

COMPARE FOR ZEROS

CMPZ

OP: 232-235

Format:

```
CMPZ COP1, R, BADDR
```

Function:

Compare two operand fields according to algebraic values, assuming the first field to be comprised of all zeros (0000).

There is no restriction as to data type. In comparing an 8-BIT character only the rightmost four bits of the character are considered. The other bits are ignored.

Two fields of all zeros are equal regardless of sign.

Unsigned fields are considered positive. Sign conventions are the same as for arithmetic operands.

Results generated by invalid digit values are undefined.

CMPZ is an optimized S-OP. COP1 can be a short COP. The format of the OP code is 01110108S.

OP code	Data Type	8S
232	4-BIT unsigned	00
233	4-BIT signed	01
234	8-BIT unsigned	10
235	8-BIT signed	11

COMPARE FOR SPACES

CMPS

OP: 37

Format:

```
CMPS COP1, R, BADDR
```

Function:

Compare two operand fields according to binary values, assuming the first field to be comprised of all spaces (0100 0000).

The comparison is performed left to right.

Unsigned 8-BIT format is assumed with no checking to verify otherwise.

This operator is not sensitive to collate table address and is valid only for the native collating sequence.

COBOL74 S-LANGUAGE

COMPARE FOR CLASS

CMPC

OP: 38

Format:

```
CMPC COP1, C, BADDR
```

Function:

Compare the operand field and determine whether the field is:

C = 00 Completely alphabetic
01 Completely numeric
10 Not completely alphabetic
11 Not completely numeric

If the condition being tested is TRUE, a transfer to the address BADDR given in the instruction occurs; otherwise, control is passed to the next sequential instruction.

In the alphabetic test, each character is range-checked for 1100 0001 through 1100 1001, 1101 0001 through 1101 1001, 1110 0010 through 1110 1001, and for 0100 0000. Unsigned 8-BIT format is assumed with no checking to verify otherwise.

In the numeric test each character is range-checked for 1111 0000 through 1111 1001. Signed or unsigned 8-BIT format is permitted. The four bits in the sign position of a signed 8-BIT field are ignored. The sign position is the leftmost 4 bits of the most significant character.

COMPARE REPEAT

CMPR

OP: 45

Format:

CMPR OPND1, COP1, R, BADDR

Function:

Compare the two operand fields according to binary value.

If the COLLATE TABLE ADDRESS is nonzero, each character in the operands must be translated before comparison. This is accomplished by using each 8-BIT character from OPND1 and COP1, multiplied by eight, as an index into the translation table located at the address given by the COLLATE TABLE ADDRESS to obtain the character to be compared.

Comparison proceeds from left to right.

The field lengths are considered equal by repeating OPND1.

Both fields are assumed to have unsigned 8-BIT data type.

The size of COP1 must be evenly divisible by the size of OPND1; otherwise, the results of the compare may be erroneous.

COBOL74 S-LANGUAGE

COMPARE COLLATE

CPC

OP: 04

Format:

CPC OPND1, COP1, R, BADDR

Function:

If the COLLATE TABLE ADDRESS is nonzero, this program is using a non-native collating sequence and the operands must be translated into the program collating sequence before comparison. This is accomplished by using each 8-BIT character from OPND1 and COP1, multiplied by eight, as an index into the translation table located at the address given by the COLLATE TABLE ADDRESS to obtain the character to be compared.

The comparison is performed left to right with any shorter operand assumed to be right-filled with blank characters. The blanks are translated if this program is using a nonnative collating sequence.

The fields are considered equal when the equal size portions are equal and the longer field (if one is longer) has trailing blanks.

An 8-BIT data format is assumed for both fields with no checking to verify otherwise. Signed fields have the most significant four bits (the sign) modified to 1111 before any necessary translation is done. This modification is not permanent and is done so that the sign will not affect the result of an alphanumeric comparison.

MISCELLANEOUS INSTRUCTION

COMMUNICATE

COMM

OP: 33

Format:

```
COMM COP1
```

Function:

Move the length and address fields from the COP1 entry to the RS.COMMUNICATE.MSG

TR field located in the program RS.NUCLEUS, converting them enroute. The origin field is unchanged.

The length is converted from a digit or character length to a bit length. The address is stored as an absolute bit address.

COBOL74 S-LANGUAGE

LOAD COMMUNICATE REPLY

LDCR

OP: 41

Format:

LDCR DADDR

Function:

The LDCR reply does the mapping of RS.RMSG.P2 and the last logical I/O status values as follows:

RS.RMSG.P2 Value		Value Stored at DADDR
0	Good Complete	0
1	AT END	1
2	I/O Error	2
3	Incomplete I/O	3
4	Duplicate Record OK	0

	LAST LOGICAL I/O STATUS	Value Stored at DADDR
Bit 0	Exception Descriptor Follows	
	Value of 0	2
	Value of 1	Examine following bits
1	Boundary Violation	1
2	Duplicate Key	1
3	Sequence Error	1
4	Variable Length Record Error	2
5	Invalid Key	1
6	Reserved	–
7	Parity Error	2
8	Reserved	–
9	AT END / EOP	Not requested
10	Short Block	2
12	Reserved	–
13	Break On Output	2
14	Reserved	–
15	Timeout	2
16 – 23	Reserved	–

CONVERT

CONV

OP: 40

Format:

```
CONV COP1 DADDR N
```

Function:

Convert the operand denoted by COP1 from a decimal value to an unsigned 24-bit binary value, truncating or zero-filling on the left if necessary. Place the result at the location specified by DADDR. N represent the number of bits of converted value to store at DADDR.

The operand must be either unsigned 4-BIT or unsigned 8-BIT units.

Refer to MAKE PRESENT for definition of DADDR.

COBOL74 S-LANGUAGE

MAKE PRESENT

MAKP

OP: 42

Format:

MAKP COP1, DADDR

Function:

Load the data segment specified by COP1 and place the base relative address of the data area specified by COP1 into the 24-bit location specified by DADDR.

DADDR is an unsigned binary value which specifies a digit displacement from the data segment zero base.

The container size of DADDR is DISPB.

FILE STATUS

FSTA

OP: 57

Format:

```
FSTA, WRIT, COP
```

Function:

COP contains the translated value of the status of the last IO.WRIT which indicates whether the last I/O was a READ or a WRITE.

The translation from COMMUNICATE REPLY (RE.RMSG.P2) or the last logical I/O status values to File Status Codes will be according to the following table:

	Communicate Reply (Values returned by MCP)	File Status Codes
0	Good Complete	00
1	AT END / EOP	10 (READ) 34 (WRITE)
2	I/O Error	Examine I/O status
3	Incomplete I/O	–
4	Duplicate Record OK	02

	Last Logical I/O Status	File Status Codes
Bit 0	Exception Description Follows	
	value of 0	99
	value of 1	Examine following bits
1	Boundary Violation	24
2	Duplicate Key	22
3	Sequence Error	21
4	Variable Length Record Error	92
5	Invalid Key	23
6	Reserved	–
7	Parity Error	30
8	Reserved	–
9	AT END / EOP	Not requested
10	Short Block	91
11	Reserved	–
12	Reserved	–
13	Break On Output	97
14	Reserved	–
15	Timeout	96
16-23	Reserved	–

For any Communicate Reply or last logical I-O status value that does not have a File Status code, the condition is masked out with the result mask specified by the communicate. An EOP will result in a 34 instead of the expected (and correct) 00.

COBOL74 S-LANGUAGE

CHARACTER STRING S-OPS

DESCRIPTOR SETUP

DSET

OP: 50

Format:

DSET DADDR, COP

Function:

Build a descriptor of COP1 at the location specified by DADDR.

The format of a descriptor (DESC) is that of a COP entry after the OCCURS DEPENDING value has been applied and after subscripts or indices have been evaluated to derive an ultimate displacement.

The current B format for a descriptor is as follows:

DESC (64)

DATA-TYPE	MULTIPLE-ENTRY-FLAG	SIGNED	SHARED-DATA-FLAG	LITERAL-FLAG
(4)	(1)	(1)	(1)	(1)

LENGTH-IN-DIGITS	SEGNO-OR-IPCD-INDEX	DISPLACEMENT IN DIGITS	FILLER
(14)	(10)	(20)	(12)

The MULTIPLE.ENTRY.FLAG and LITERAL.FLAG are equal to zero in the above format.

INSPECT SETUP

ISET

OP: 51

Format:

ISET V1, V2, FLAC, DADDR1, COP1, COP2, DADDR2, OPND

Function:

Build an INSPECT.TABLE entry at DADDR1.

V1 specifies whether this is the last entry (V1 = 1) or not. V2 indicates whether BEFORE, V2 = 1, or AFTER, V2 = 2 or both, V2 = 3, were specified. FLAC indicates the mode of INSPECTION.

FLAC = 0 indicates FIRST
FLAC = 1 indicates LEADING
FLAC = 2 indicates ALL
FLAC = 3 indicates CHARACTERS

DADDR1 indicates the location for this INSPECT TABLE entry. COP1 indicates the number of characters to be tallied or replaced when FLAC is not equal to 3.

COP2 is the count field for tallying or the replacement string. DADDR2 is the location of the source field description if BEFORE/AFTER is specified.

OPND is the delimiter if BEFORE/AFTER is specified.

Set INELIGIBLE.FLAG = 0.

If V2 = 0, set AFTER.FLAG = 0 and examine the source field for the delimiter specified by OPND1. If a match is found, set ELIGIBLE.POSITION to the character position of the beginning of the first occurrence of the delimiter. If a match is not found, set ELIGIBLE.POSITION = 0 and AFTER.FLAG = 1. This, in effect, makes the entry eligible for all characters of the source field.

If V2 = 1, set AFTER.FLAG = 1 and examine the source field for the delimiter specified by OPND1. If a match is found, set ELIGIBLE.POSITION to the character position to the right of the first occurrence of the delimiter. If a match is not found, set INELIGIBLE.FLAG = 1.

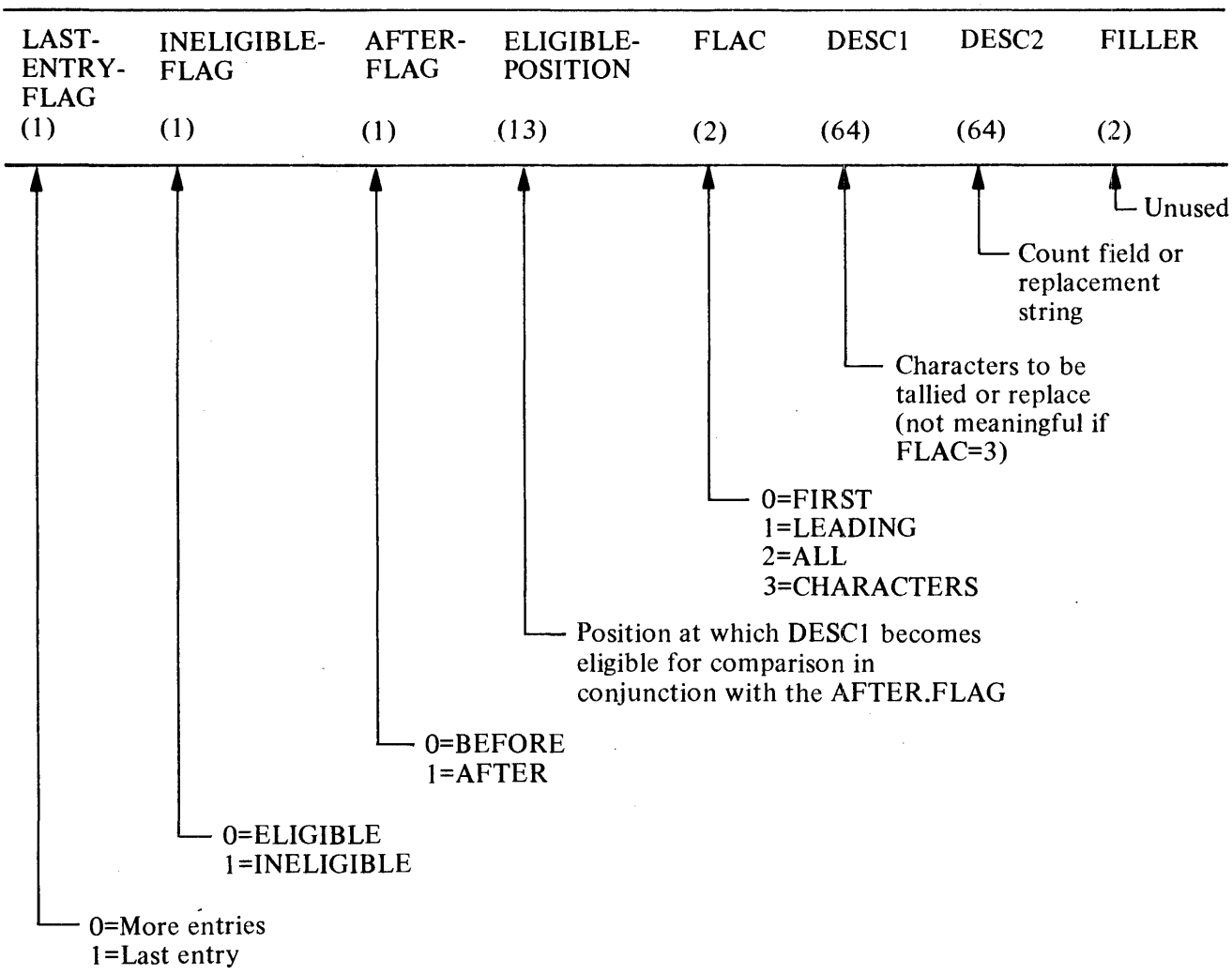
If V2 = 3, set ELIGIBLE.POSITION = 0 and AFTER.FLAG = 1. This, in effect, makes the entry eligible for all characters of the source field.

COBOL74 S-LANGUAGE

The description of the INSPECT TABLE is as follows:

INSPECT.POINTER is the first 24 bits of the Inspect Table which is used to save a pointer into the source string in case inspection is interrupted. This field is followed by a variable number of entries with the following format:

INSPECT.TABLE.ENTRY (148)



INSPECT

INSP

OP: 52

Format:

INSP V, DADDR1, DADDR2

Function:

Inspect the source field specified at DADDR2, tallying or replacing a variable number of 8-BIT character(s). The location of the character(s) to be tallied or replaced and the location of count field or replacement string is specified in the Inspect Table specified by DADDR1.

Eligibility of an entry is established as follows:

1. If INELIGIBLE.FLAG = 1, this entry is ineligible.
2. If AFTER.FLAG = 0 and the current character position is not less than ELIGIBLE.POSITION, this entry is no longer eligible and it is made ineligible; otherwise, the entry is eligible.
3. If AFTER.FLAG = 1 and the current character position is less than ELIGIBLE.POSITION, this enty is ineligible; otherwise the entry is eligible.
4. When looking for a match and the number of characters left in the source is less than the size of the data item specified by DESC1, this entry is made ineligible.

NOTE
An entry is made ineligible by setting INELIGIBLE.FLAG = 1.

The comparison operation to determine the occurrences of the operands to be tallied or replaced occurs as follows:

1. The entries in the INSPECT.TABLE are considered in the order in which they are specified.
2. If the entry is eligible, DESC1 is compared against an equal number of contiguous characters, starting with the leftmost character of the source field.
3. If no match occurs or if the entry is ineligible, the comparison is repeated for each successive entry, if any, until either a match is found or there is no next successive entry. When there is no successive entry, the character position immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins with the first table entry.
4. Whenever a match occurs, tallying or replacing takes place. The character position immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the source field and the comparison cycle starts again with the first table entry.

5. The comparison operation continues until there are no eligible entries in the table or until the rightmost character position of the source field has participated in a match or has been considered as the leftmost character position. When this occurs, the inspection is terminated.

NOTE

It is intended that this process be interruptable at the beginning of the cycle. INSPECT.POINTER is the first 24 bits of the Inspect Table, and is used to save and restore the pointer into the source field in case an interrupt occurs.

A detailed description of the inspection cycle is as follows:

1. Set current position P = INSPECT.POINTER.
2. Point to the first entry in the Inspect Table.
3. If the entry is ineligible, go to step 5.
4. Do the following according to the contents of FLAC:
   a. If FIRST is specified, look for a match.

      If a match is not found, go to step 5.

      If a match is found, make the entry ineligible, tally or replace and go to step 7.

   b. If LEADING is specified, make the entry ineligible.

      If (AFTER.FLAG = 0 AND P NOT = 1) OR (AFTER.FLAG = 1 AND P NOT = ELIGIBLE.POSITION), go to step 5.

      Look for a match.

      If a match is not found, go to step 5.

      If a match is found, tally or replace contiguous occurrences of DESC1 and go to step 7.

   c. If ALL is specified, look for a match.

      If a match is not found, go to step 5.

      If a match is found, tally or replace and go to step 7.

   d. If CHARACTERS is specified, tally or replace and go to step 6.
5. Get the next table entry. If there is a next entry, go to step 3.

If INELIGIBLE.FLAG = 1 for all the table entries, terminate the process.

6. Increment P by 1.
7. If P > size of the source string, terminate the process; otherwise, go to step 2.

STRING

STRI

OP: 53

Format:

STRI Z, OPND1, DADDR1, COP1, OPND2, BADDR

Function:

Move a variable number of 8-BIT characters from the source field denoted by OPND1 to the destination field specified at DADDR1 using the value specified in COP1 as a character offset into the destination field. Z indicates how OPND2 is to be used:

0 = OPND2 is the number of characters to move

1 = OPND2 is the delimiting character(s)

2 = OPND2 is omitted – the number of characters to move is the size of OPND1

BADDR specifies the next statement if overflow occurs.

An overflow condition is caused if at anytime during execution the value in the pointer field is less than 1 or greater than the size of the destination string.

If an overflow condition exists, the operation is terminated and the next instruction address is obtained from BADDR.

The transfer of data is governed by the following rules:

1. Characters are moved from the source field to the destination field in accordance with the rules for MOVE ALPHANUMERIC (MVA), except that no space fill is provided. The contents of COP1 are incremented by 1 for each character transferred.

2. If V = 0, characters are transferred until the end of the source field is reached, or until the number of characters specified by OPND2 are transferred.

   If V = 1, characters are transferred until the end of the source field is reached, or until the delimiting character(s) specified by OPND2 are encountered. The delimiting character(s) specified are not transferred.

COBOL74 S-LANGUAGE

DELIMITER SETUP

DLIM

OP: 54

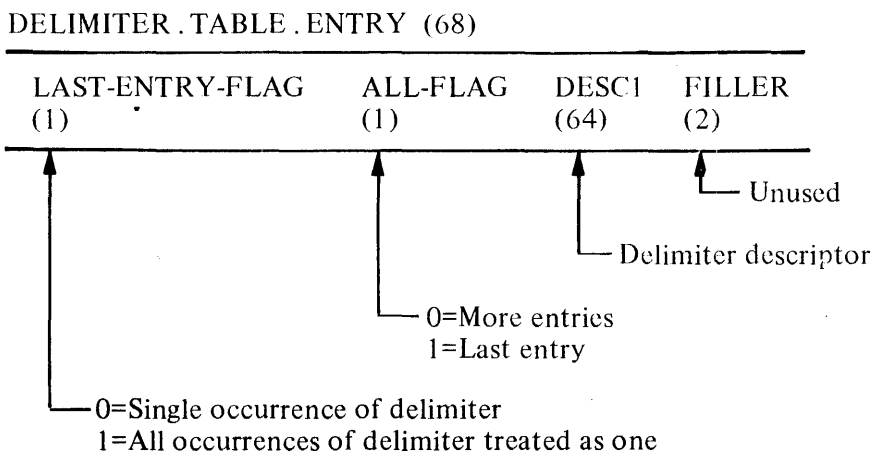
Format:

DLIM V1, V2, DADDR1, COP1

Function:

Build a delimiter table entry at the location specified by DADDR1. V1 = 1 specifies that this is the last entry. V2 = 1 indicates that ALL was specified. COP1 is the delimiter.

The table used by the Unstring S-operator contains a variable number of entries with the following format:



When Z = 1, DADDR3 specifies the location of the Delimiter Table and the following comparison cycle occurs:

1. The entries in the Delimiter Table are processed in the order specified.
2. The delimiter is compared against an equal number of characters of the source field, starting with the leftmost character.
3. If no match occurs, the comparison is repeated for any successive entry until a match is found or until there is no next successive entry. When there is no successive entry, the character position to the immediate right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins with the first table entry.
4. Whenever a match occurs, examination stops and the comparison cycle is discontinued.
5. The comparison operation continues until there are no more entries in the table or until the rightmost character position of the source field has participated in a match or has been considered as the leftmost character position. When this occurs, examination is terminated.

UNSTRING

UNST

OP: 55

Format:

**UNST F, M, Z, C, J, BADDR, DADDR1, COP1,
DADDR2, OPND1, DADDR3, COP2, COP3, DADDR4**

Function:

Move a variable number of 8-BIT characters from the source field specified at DADDR1 to the destination field denoted by COP1 using the value specified at DADDR2 as a character offset into the source field.

Any of the following situations cause an overflow condition:

1. F=2 or F=3 and the value in the pointer field is less than 1 or greater than the size of the source string.
2. Z=0 and the value in OPND1 is less than 1 or greater than the size of the source string.
3. Upon completion of the operation, F=1 or F=3 and either of the following occurs:
   a. The source field contains characters that have not been examined.
   b. Z=0 and the number of characters transferred is less than the number of characters specified by OPND1.

If an overflow condition exists, the operation is terminated and the next instruction address is obtained from BADDR.

The transfer of data is governed by the following rules:

1. The string of source characters is examined beginning with the relative character position in the pointer field.
2. The number of characters examined is determined as follows:
   a. If Z=1, the examination proceeds from left to right until either a delimiter in the Delimiter Table specified by DADDR3 is encountered or until the end of the string is encountered.

      If the end of the string is encountered before the delimiting condition is met, examination terminates with the last character to be examined.

      If two contiguous delimiters are encountered, the number of characters examined is zero.

   b. If Z=2, the number of characters is the size specified by COP1.

3. The characters, thus examined, are moved to the destination field as follows:

   a. If C = 0, the characters are moved in accordance with the rules of the MOVE NUMERIC (MVN) S-operator.

      If the number of characters to be moved is zero, the destination field is zero-filled.

   b. If C = 1, the characters are moved according to the rules of the MOVE ALPHANUMERIC (MVA) S-operator.

      If J = 1, the characters are moved right-justified.

      If the number of characters to be moved is zero, the destination field is space-filled.

4. If the delimiter receiving field is present, the delimiting character(s) are moved to it according to the rules of the MOVE ALPHANUMERIC (MVA) S-operator. If the delimiting condition is the end of the string, then it is space-filled.

5. If the count field is present, the number of characters examined is moved to it according to the rules of the MOVE NUMERIC (MVN) S-operator.

6. If the tallying field is present, 1 is added to the contents according to the rules of the Increment By One (INC1) S-operator.

7. If Z = 0, Z = 2 or the delimiting condition is the end of the string, the pointer field is set to point to the character to the right of the last character transferred in step 3.

   If Z = 1, the pointer field is set as follows:

   a. If the ALL.FLAG = 0 for the matched delimiter, it points to the character to the right of the delimiter.

   b. If the ALL.FLAG = 1 for the matched delimiter, the string is examined for contiguous occurrences of the delimiter and it points to the right of the last such occurrence.

INTER-PROGRAM COMMUNICATION

IPC DICTIONARY

IPCD

Format:

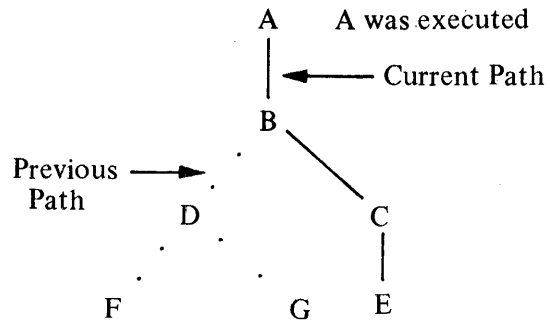
IPCD V COP BADDR

Functions:

The execution of the IPCD S-OP results in the construction and insertion of an 80-bit system descriptor for the COP in the IPC.DICTIONARY at the displacement address specified by V. The base address of the IPC.DICTIONARY is adjacent to the program RSN and can be found in RS.IPC.DICTIONARY.SPACE as an absolute address. The BADDR contains the address of the first IPCD S-op which references an operand that lies in an overlayable data segment. This address is required by the interpreter when checkerboarding has occurred and a necessary data segment does not fit into dynamic memory without overlaying another required data segment.

The Inter-Program Communication (IPC) Module provides a facility to transfer control from one program to another and the ability for both programs to have access to the same data items. Language is not a barrier in IPC. The names of the programs to which control is to be passed may or may not be known at compile time. Additionally, this module provides the ability to determine the availability of memory for the program to which control is being passed.

The definition of a 'run unit' is critical to the implementation of the CALL/CANCEL mechanism described in the ANSI Standard. The beginning-of-job (BOJ) of any program by an execute instruction does not establish a run unit. A run unit is established only when an executed program initiates another program through the CALL communicate. That called program is now a member of the run unit associated with that program that was originally executed. Likewise, any program called by a program within the run unit becomes part of that run unit and remains in that run unit until terminated or cancelled (CANCEL). A job cannot be a member of more than one run unit. The following figure represents seven programs (A through G) which have been called within a run unit.

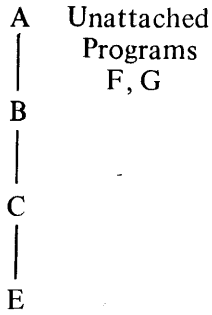


COBOL74 S-LANGUAGE

The connecting links are generated by and represent the last used path, and the link exists until a return (EXIT PROGRAM) is accomplished. Once a called program has been exited (D, F, G), it remains suspended in the current state. The only path that is of interest is the path last traversed.

The current path is important in order to check the validity of a CALL or CANCEL statement; a program may not call or cancel itself or any of its predecessors. The other links are unimportant, as any program in the run unit can call or delete other existing programs (with the exceptions previously mentioned) or can call new programs.

If, for example, program E cancels program D, then the run unit consists of all of the following programs and appears as:



A call to any of these programs results in a transfer of control to the existing state, whereas a call to any other program (including D) causes an initial state copy to be invoked before control is transferred. The termination (by STOP RUN or ABORT) of any program in a run unit results in the removal of all programs in that run unit from memory.

The calling program may specify one or more data items to which the called program has access. The shared data may be any 01 or 77 level item (including items whose addresses have been received through a CALL) described in the calling program. The data items may be named and defined differently in each program. Additionally, storage for the shared data is never allocated in the called program; the address is always passed to the called program.

The IPC.DICTIONARY is a list of SYSTEM.DESCRIPTORS built by the program to describe the parameters to be passed with a CALL. This dictionary is within the space defined by RS.IPC.DICT in the RS.NUCLEUS of the calling program. The length of this dictionary is passed in the CALL communicate. The MCP verifies that the number and length of parameters passed match the IPC.PARAMETER.LIST of the CALLed program.

The IPC.PARAMETER.LIST is a series of 24-bit fields that contain the length in bits of the parameters required for a given program. The original copy is generated by the compiler and resides at the end of the code file. The MCP can locate this list through a 24-bit field in the Program Parameter Block (the PROG.IPC.PTR contains the relative disk address in the code file of the IPC-.PARAMETER.LIST). The number of entries in the list is obtained from a 16-bit field in the PPB (PROG.IPC.SIZE). After the program has been successfully called, the IPC.PARAMETER.LIST is appended to the RS.NUCLEUS (and the first element contains the number of entries) in order to facilitate future calls on this program.

After the MCP verifies that the number and length of parameters are correct, it updates this field to allow access to the shared data. The interpreter uses this field just as it would use the address of a data dictionary, that is, to obtain the absolute address of the IPC.DICTIONARY.

Figure E-2 illustrates the manner in which data is linked between the calling program and the called program.

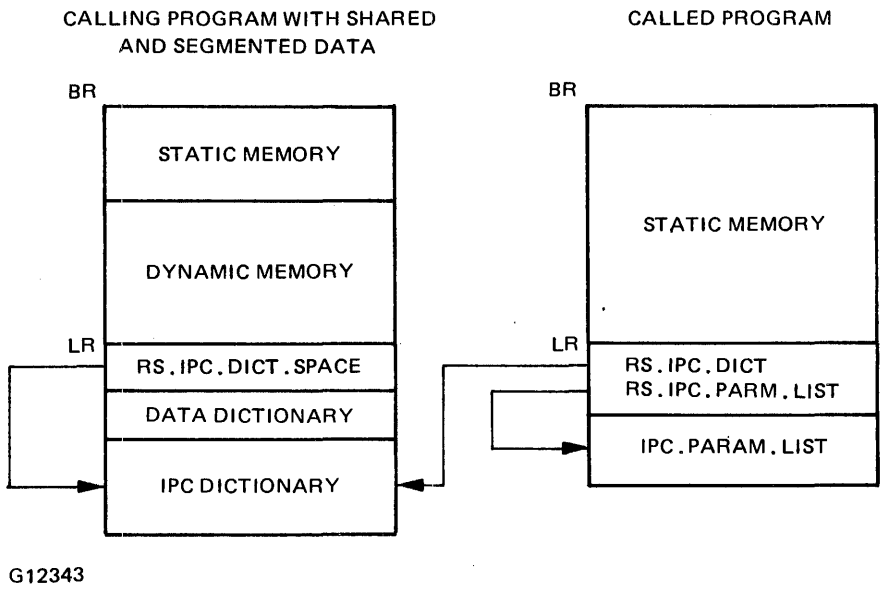


Figure E-2. Memory Layout

Accessing shared data should be almost identical to accessing segmented data. One difference is that the reading and writing take place outside the program base-to-limit area and, therefore, the base-limit check must be inhibited. This implies that any COP for a parameter must have a flag (SHARED.DATA.FLAG) which suppresses the base-limit test and indicates that there is a following 10-bit field (IPCD.INDEX) which supplies an index into the IPC.DICTIONARY. With the absolute address obtained from the IPC.DICTIONARY, the interpreter can proceed as if it has an absolute address for a normal data item.

APPENDIX F
COMMUNICATION CONCEPTS AND EXAMPLES

COBOL74 QUEUE FILES

Queue files permit message transfer between one or more programs. A queue is established when the program opens a queue file. This program may be one of many readers or writers of the queue file. Message transfer to and from the queue is handled by the operating system when a WRITE or a READ operation is specified for the queue file. Messages are written to the back of the queue and read from the front of the queue. Therefore, if one program opens a queue file I/O, there is a good chance that no message transfer between two programs will actually occur. Instead, the single program could be reading messages that it just wrote. Figure F-1 is a diagram of a unidirectional message transfer with two writers and three readers.

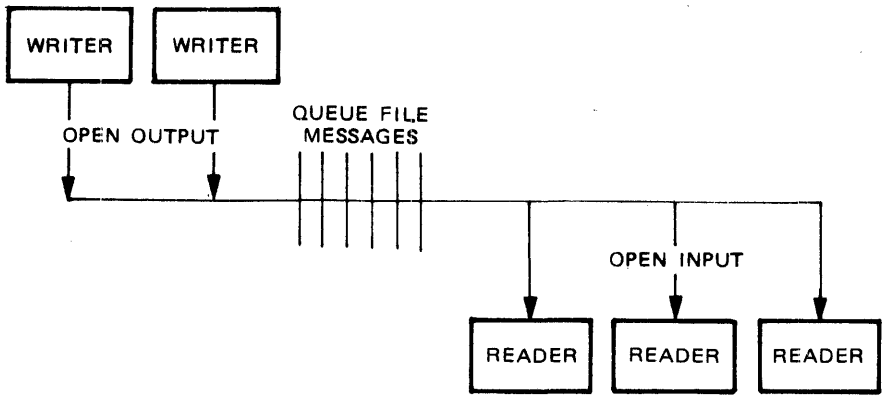


Figure F-1. Unidirectional Queue File Message Transfer

When message transfer in two directions is desired, two queue files should be used. This can be seen in figure F-2.

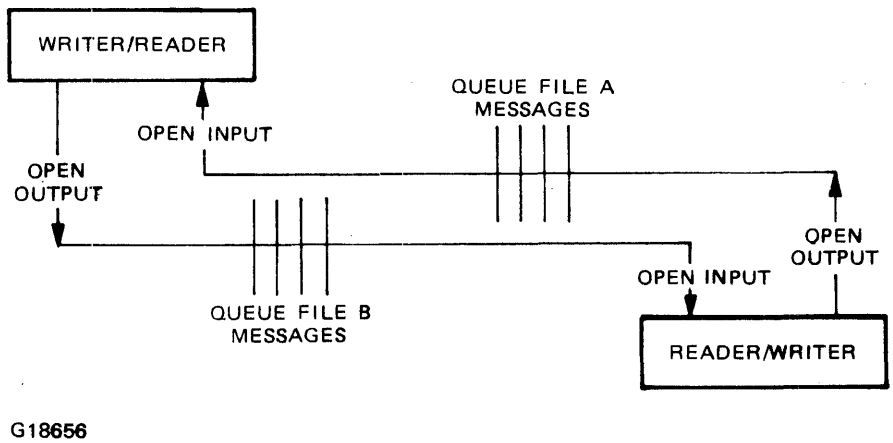


Figure F-2. Bidirectional Queue File Message Transfer

Messages may be written into a queue (space permitting) even though a reader of that queue does not exist. The messages are stored until read out of the queue or the queue file is closed by the last program that has the queue file open. The oldest messages are read from the queue first and are then removed (FIFO). When the queue file is closed by the last user of the queue file, any unread messages are discarded by the operating system.

In the following program, there is one queue file that is read from and one queue file that is written to. This is done for example only, as normal use of queue files is for communication between separate programs. The VALUE OF TITLE clause for each of the queue files declared in the program has the same value. This is the only statement in the source code that instructs the operating system to use the same location in memory (queue) for each of the declared queue files.

```
000100
000200 IDENTIFICATION DIVISION.
000300
000400 PROGRAM-ID.                          QUEUE-ECHO.
000500 AUTHOR.                              ORWELL84.
000600 DATE-WRITTEN.                        12/02/83.
000700 SECURITY.                            NONE.
000800 ENVIRONMENT DIVISION.
000900
001000 CONFIGURATION SECTION.
001100
001200 SOURCE-COMPUTER.                      B1000.
001300 SOURCE-COMPUTER.                      B1000.
001400
001500 INPUT-OUTPUT SECTION.
001600
001700 FILE-CONTROL.
001800    SELECT QUEUE-IN              ASSIGN TO QUEUE
001900               FILE STATUS IS IN-STAT
002000               ACTUAL KEY IS Q-IN-KEY
002100               RESERVE 2 AREAS.
002200    SELECT QUEUE-OUT             ASSIGN TO QUEUE
002300               FILE STATUS IS OUT-STAT
002400               ACTUAL KEY IS Q-OUT-KEY
002500               RESERVE 2 AREAS.
002600
002700 I-O-CONTROL.
002800    SAME RECORD AREA FOR QUEUE-IN QUEUE-OUT.
002900
003000 DATA DIVISION.
003100
003200 FILE SECTION.
003300
003400    FD  QUEUE-IN
003500           VALUE OF MAXCENSUS IS 2
003600           VALUE OF MAXSUBFILES IS 4
003700           VALUE OF TITLE IS "ORWELL84".
003800        01  QUEUE-REC-IN.
003900            03  BYE-BYE        PIC X(3).
004000            03  FILLER         PIC X(77).
004100    FD  QUEUE-OUT
004200           VALUE OF MAXSUBFILES IS 4
004300           VALUE OF TITLE IS "ORWELL84".
004400        01  QUEUE-REC-OUT  PIC X(80).
004500
```

```
004600 WORKING-STORAGE SECTION.
004700    77 Q-IN-KEY              PIC 9(8) COMP VALUE 1.
004800    77 Q-OUT-KEY             PIC 9(8) COMP VALUE 0.
004900    77 FLAG                  PIC 9 COMP VALUE ZERO.
005000       88 EOF                VALUE 1.
005100    01 Q-MESS                       PIC X(80).
005200    01 OUT-STAT               PIC XX VALUE "00".
005300       88 Q-FULL                     VALUE "95".
005400    01 IN-STAT                PIC XX VALUE "00".
005500       88 Q-EMPTY                    VALUE "94".
005600 PROCEDURE DIVISION.
005700
005800 MAIN-DRIVER.
005900     OPEN INPUT QUEUE-IN.
006000     OPEN OUTPUT QUEUE-OUT.
006100     PERFORM ECHO THRU ECHO-EXIT UNTIL
006200        FLAG > 0.
006300     CLOSE QUEUE-IN.
006400     IF FLAG NOT = 2
006500       CLOSE QUEUE-OUT.
006600     STOP RUN.
006700
006800 ECHO.
006900     DISPLAY "ENTER ONE LINE MESSAGE, AND I WILL ECHO IT BACK".
007000     DISPLAY "TO EXIT PROGRAM, ENTER BYE"
007100     ACCEPT Q-MESS.
007200     MOVE Q-MESS TO QUEUE-REC-OUT
007300     MOVE Q-IN-KEY TO Q-OUT-KEY.
007400     WRITE QUEUE-REC-OUT.
007500     MOVE Q-OUT-KEY TO Q-IN-KEY.
007600     IF Q-FULL
007700        READ QUEUE-IN
007800        GO TO ECHO-EXIT.
007900     DISPLAY QUEUE-REC-OUT.
008000     WAIT UNTIL READ-OK ON QUEUE-IN (1)
008100                READ-OK ON QUEUE-IN (2)
008200                READ-OK ON QUEUE-IN (3)
008300                READ-OK ON QUEUE-IN (4)
008400     GIVING Q-IN-KEY.
008500     READ QUEUE-IN AT END
008600        MOVE 1 TO FLAG.
008700     IF BYE-BYE = "BYE"
008800        DISPLAY "BYE-BYE"
008900        MOVE 2 TO FLAG
009000        CLOSE QUEUE-OUT
009100        GO TO ECHO-EXIT.
009200 ECHO-EXIT.
009300     EXIT.
009400 END-OF-JOB.
```

COBOL74 REMOTE FILES

Each COBOL74 program can declare a maximum of one remote file. A remote file, as declared in a COBOL74 program, is a Burroughs extension to ANSI 74 COBOL. A remote file allows the COBOL74 program a high degree of control for input, output or I/O with a set of data communication devices.

With COBOL74 remote files, all message transfer and control of the remote files is handled by the network controller in conjunction with a Message Control System (MCS), if present. The presence of an MCS allows dynamic, run-time modification of the station list in the network controller, such as transforming the second or subsequent open request to an attach of the station to an existing remote file instead of an open of a new remote file. Without an MCS, there can be no flexibility of the station list and attachment of a remote file to a station is rigidly dependent on the file statement as pre-defined in the network controller.

The controlling MCS is responsible for approving or denying the second and any subsequent open of the same remote file name. The approval or denial of the open is determined by a variety of factors, such as authentication of the station list as defined in the network controller.

If there is no MCS, the second and any subsequent open of the same remote file name results in a denial of the open from the network controller, with a message indicating the file is locked.

In figure F-3, program A opens the remote file named XYZ, and the network controller generated by the the Network Definition Language (NDL) compiler assigns remote file number 1 to that file, which is the data communication system's internal method of identification. Program B opens the remote file named XYZ, and the network controller assigns remote file number 2 (or the next lowest available file number) to that file. Thus, it is possible (only under control of an MCS) to run multiple copies of the same program from different stations, or declare the same remote file name in different programs, with a guarantee from the system that messages to and from individual programs will be correctly directed to the appropriate program. This is handled by assigning a unique remote file number to each remote file opened even if the remote file names are the same. (If program A or program B is an MCS, the above sequence of events may be different.)

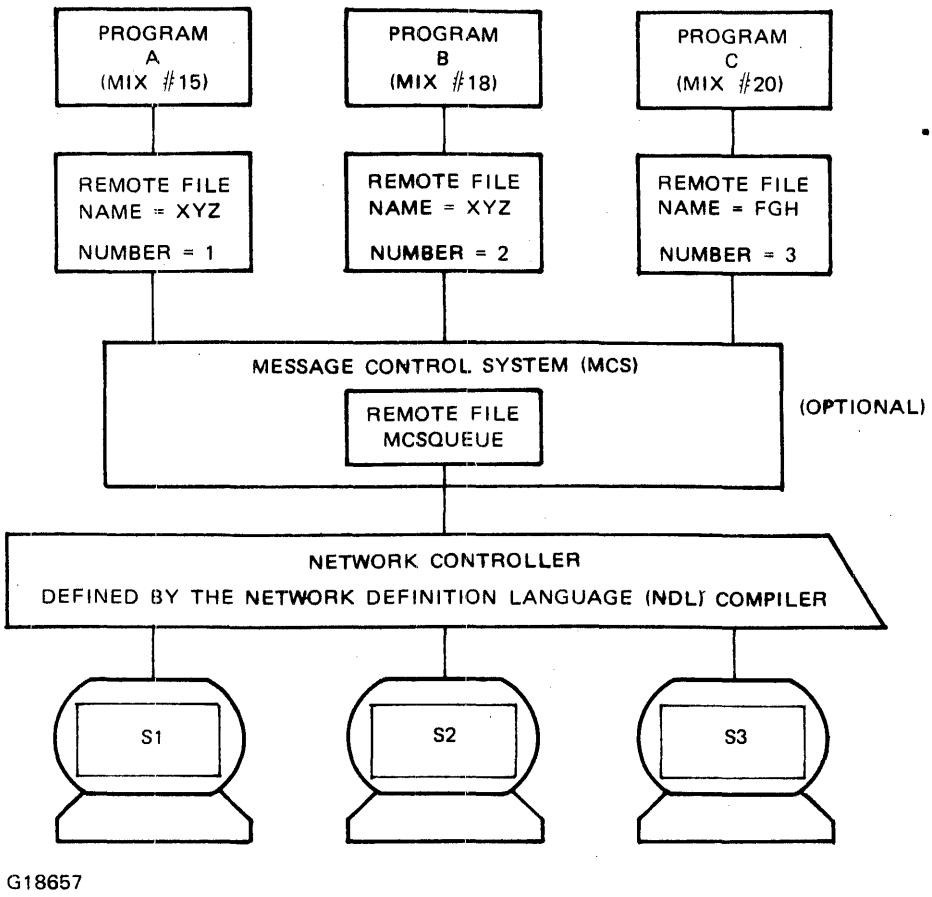


Figure F-3. COBOL74 Remote Files

Enhancements to the Mark 11.0 COBOL74 compiler expand the capabilities of remote file usage. These include the abilities to selectively access multiple stations of a remote file and to construct variable length records for a remote file.

Multiple Stations of a Remote File

A COBOL74 program can open a remote file with more than one station if that configuration has been defined in the station list of the network controller. More than one station can also be established for a remote file of a COBOL74 program when running under an MCS by means of the dynamic attachment mechanism. The MAXSTATIONS file attribute must be declared greater than one in the COBOL74 program (see programming example below) or the NUMBER.STATIONS file attribute must be modified to the number of stations expected.

Example:

MODIFY <program name> FILE <file name> NUMBER.STATIONS = <integer>

In COBOL74, a program may initiate communication to a specific terminal, which was attached in its remote file open, and that terminal remains assigned to the program for the duration of the remote file open. This is done by association of a Relative Station Number (RSN) within a station list for that remote file, as defined in the network controller.

The LASTSTATION file attribute indicates the relative station number of the source of a message following a READ operation from a remote file. By changing the value of the LASTSTATION attribute prior to a WRITE operation to a remote file, the message can be directed to another station. Use of the LASTSTATION attribute is optional. It is only required if the program needs to know what station was accessed on the last READ operation, or if the program wants to direct the next WRITE operation to a station other than the one that was the source of the previous READ operation. The LASTSTATION attribute can be read or changed when the file is open.

The following example shows how the LASTSTATION and MAXSTATIONS file attributes can be used in a program using remote files. When the number of stations has been declared to be greater than 1 and the program has been entered into the JOBS file of the SMCS program, two or more users can sign on to this program and receive the messages from each other.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.                    MULTI-STATIONS.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500 SOURCE-COMPUTER.               B1000.
000600 OBJECT-COMPUTER.               B1000.
000700 INPUT-OUTPUT SECTION.
000800
000900 FILE-CONTROL.
001000     SELECT   RMTE             ASSIGN TO REMOTE
001100                               RESERVE 3 AREAS.
001200 DATA DIVISION.
001300
001400 FILE SECTION.
001500 FD  RMTE
001600        VALUE OF MAXSTATIONS IS 2
001700        RECORD CONTAINS 1950 CHARACTERS.
001800
001900 01  RMTE-RECORD               PIC X(1950).
002000
002100 WORKING-STORAGE SECTION.
002200 01  SCREEN-OUTPUT.
002300     05  TOP-LINE                PIC 9(10) COMP
002400           VALUE @0C000000000@.
002500     05  SCREEN                  PIC X(1920).
002600     05  BOTTOM-LINE             PIC 9(12) COMP
002700           VALUE @27E600000003@.
002800 01  ENTRY-SCREEN-OUT.
002900     05  ES-FIRST-LINE.
003000         10  FILLER              PIC X(31) VALUE SPACES.
003100         10  ES-TITLE            PIC X(19)
003200            VALUE "REMOTE FILE EXAMPLE".
003300         10  FILLER             PIC X(30) VALUE SPACES.
003400     05  FILLER                  PIC X(160) VALUE SPACES.
003500     05  ES-NAME-LINE.
003600         10  FILLER             PIC X(20)  VALUE SPACES.
003700         10  NAME-FILL          PIC X(16)
003800            VALUE "ENTER YOUR NAME ".
003900         10  LEFT-DELIMITER     PIC 9(02) COMP VALUE @1F@.
004000         10  FILLER             PIC X(20) VALUE SPACES.
004100         10  RIGHT-DELIMITER    PIC 9(02) COMP VALUE @1E@.
004200         10  FILLER             PIC X(22) VALUE SPACES.
004300     05  FILLER                 PIC X(80) VALUE SPACES.
004400     05  ES-MESSAGE-LINE.
004500         10  FILLER             PIC X(8)
004600            VALUE "MESSAGE ".
004700         10  LEFT-DELIMITER     PIC 9(02) COMP VALUE @1F@.
004800         10  FILLER             PIC X(70) VALUE SPACES.
004900         10  RIGHT-DELIMITER    PIC 9(02) COMP VALUE @1E@.
005000     05  FILLER                 PIC X(1440)  VALUE SPACES.
005100 01  INPUT-SCREEN.
005200     05  YOUR-NAME              PIC X(20).
005300     05  YOUR-MESSAGE           PIC X(70).
005400 01  SECOND-SCREEN-OUT.
```

```
005500      05  SS-FIRST-LINE.
005600          10  FILLER              PIC X(31) VALUE SPACES.
005700          10  SS-TITLE            PIC X(19)
005800              VALUE "REMOTE FILE EXAMPLE".
005900          10  FILLER              PIC X(30) VALUE SPACES.
006000      05  FILLER                  PIC X(160) VALUE SPACES.
006100      05  FILLER                  PIC X(46)
006200          VALUE "WELCOME TO THE WORLD OF COBOL74 REMOTE FILES, ".
006300      05  NAME-OUT1               PIC X(20).
006400      05  FILLER                  PIC X(94) VALUE SPACES.
006500      05  FILLER                  PIC X(48)
006600          VALUE "I WILL NOW SEND YOUR MESSAGE TO ANOTHER STATION." .
006700      05  FILLER                  PIC X(132)  VALUE SPACES.
006800      05  FILLER                  PIC X(9)
006900          VALUE "GOOD-BYE ".
007000      05  NAME-OUT2               PIC X(20).
007100      05  FILLER                  PIC X(1311)  VALUE SPACES.
007200
007300  01  LAST-SCREEN-OUT.
007400      05  LS-FIRST-LINE.
007500          10  FILLER              PIC X(31) VALUE SPACES.
007600          10  LS-TITLE            PIC X(19)
007700              VALUE "REMOTE FILE EXAMPLE".
007800          10  FILLER              PIC X(30) VALUE SPACES.
007900      05  FILLER                  PIC X(172) VALUE SPACES.
008000      05  FILLER                  PIC X(21)
008100          VALUE "THIS MESSAGE IS FROM ".
008200      05  NAME-OUT3               PIC X(20).
008300      05  FILLER                  PIC X(112) VALUE SPACES.
008400      05  SEND-MESSAGE            PIC X(70).
008500      05  FILLER                  PIC X(120)  VALUE SPACES.
008600      05  FILLER                  PIC X(9)
008700          VALUE "GOOD-BYE ".
008800      05  FILLER                  PIC X(1316)  VALUE SPACES.
008900
```

```
009000 PROCEDURE DIVISION.
009100
009200 OPEN-FILE.
009300     OPEN I-O RMTE.
009400
009500 READ-AND-WRITE-DEFAULT.
009600****  THE DEFAULT FOR THE RELATIVE STATION NUMBER IS 1****
009700****     THIS MESSAGE GOES TO STATION WITH RSN=1.     ****
009800     MOVE ENTRY-SCREEN-OUT TO SCREEN.
009900     WRITE RMTE-RECORD FROM SCREEN-OUTPUT.
010000****  THIS READ ACCEPTS THE FIRST MESSAGE SENT TO THE ****
010100****     REMOTE FILE FROM ANY STATION ATTACHED TO IT. ****
010200     READ RMTE INTO INPUT-SCREEN   AT END
010300        GO TO CLOSE-FILE.
010400 WRITE-TO-LAST-READ.
010500****  THIS WRITE IS DIRECTED TO THE RELATIVE STATION  ****
010600****    NUMBER OF THE SOURCE OF THE LAST READ.        ****
010700     MOVE YOUR-NAME TO NAME-OUT1, NAME-OUT2, NAME-OUT3.
010800     MOVE YOUR-MESSAGE TO SEND-MESSAGE.
010900     MOVE SECOND-SCREEN-OUT TO SCREEN.
011000     WRITE RMTE-RECORD FROM SCREEN-OUTPUT.
011100     WAIT 6.
011200 WRITE-TO-DIFFERENT-STATION.
011300****  THIS WRITE IS DIRECTED TO A DIFFERENT STATION   ****
011400****     BY CHANGING THE VALUE OF LASTSTATION.        ****
011500     IF ATTRIBUTE LASTSTATION OF RMTE = 1
011600        CHANGE ATTRIBUTE LASTSTATION OF RMTE TO 2.
011700     IF ATTRIBUTE LASTSTATION OF RMTE = 2
011800        CHANGE ATTRIBUTE LASTSTATION OF RMTE TO 1.
011900     MOVE LAST-SCREEN-OUT TO SCREEN.
012000     WRITE RMTE-RECORD FROM SCREEN-OUTPUT.
012100 CLOSE-FILE.
012200     CLOSE RMTE.
012300     STOP RUN.
012400 END-OF-JOB.
```

Variable Length Records

Variable length records for a remote file are declared by the use of the RECORD CONTAINS and the RECORDS CONTAINS...DEPENDING ON phrases in the FD statement for the remote file in the source code. Run-time code is generated by the compiler to handle the variable length records. Other than specifying one of the above phrases, the programmer need not be concerned with the processing of the variable length records except as noted in the three examples below.

Example 1:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.                          VARI-METHOD1.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                     B1000.
000800 OBJECT-COMPUTER.                     B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT   RMTE                   ASSIGN TO REMOTE.
001400
001500 DATA DIVISION.
001600
001700 FILE SECTION.
001800 FD  RMTE
001900        RECORD CONTAINS 125 TO 1940 CHARACTERS.
002000
002100 01  SHORT-RECORD                   PIC X(125).
002200 01  LONG-RECORD                    PIC X(1940).
002300
002400 WORKING-STORAGE SECTION.
002500
002600 01  REMOTE-MESSAGE.
002700     05  SPACING-AREA               PIC X(115)  VALUE SPACES.
002800     05  MESSAGE-AREA               PIC X(1825) VALUE SPACES.
002900
003000 PROCEDURE DIVISION.
003100
003200 OPEN-REMOTE..
003300     OPEN I-O RMTE.
003400 WRITE-SHORT-RECORD.
003500***   ONLY THE FIRST 125 CHARACTERS ARE WRITTEN TO FILE RMTE. ***
003600     MOVE "HELLO USER, GO AWAY. I DO NOT LIKE YOU. "
003700        TO MESSAGE-AREA.
003800     WRITE SHORT-RECORD FROM REMOTE-MESSAGE.
003900     MOVE SPACES TO MESSAGE-AREA.
004000     WAIT 4.
004100 WRITE-LONG-RECORD.
004200***   UP TO 1940 CHARACTERS ARE WRITTEN TO REMOTE FILE RMTE. ***
004300     MOVE "HELLO USER.             WELCOME TO THE WORLD OF VARIABLE
004400-        " LENGTH REMOTE FILE RECORDS.  WE ARE VERY HAPPY YOU ARE
004500-      "HERE."  TO MESSAGE-AREA.
004600     WRITE LONG-RECORD FROM REMOTE-MESSAGE.
004700     WAIT 4.
004800 CLOSE-FILE.
004900     CLOSE RMTE.
005000     STOP RUN.
005100 END-OF-JOB.
```

Example 2:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.                          VARI-METHOD2.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                     B1000.
000800 OBJECT-COMPUTER.                     B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT   RMTE                   ASSIGN TO REMOTE.
001400
001500 DATA DIVISION.
001600
001700 FILE SECTION.
001800
001900 FD  RMTE
002000         RECORD CONTAINS 4 TO 1944 CHARACTERS
002100         DEPENDING ON LENGTH-IN-BYTES.
002200
002300 01  DEPENDING-RECORD.
002400         05  LENGTH-IN-BYTES        PIC 9(4).
002500         05  ACTUAL-RECORD-AREA     PIC X(1940).
002600
002700 WORKING-STORAGE SECTION.
002800 01  MESSAGE-IN.
002900     05  HELLO                      PIC X(6).
003000     05  MESSAGE-AREA.
003100         10  USER-CODE              PIC X(7).
003200         10  FILLER                 PIC X VALUE SPACES.
003300         10  REST-OF-MESSAGE        PIC X(1932).
003400 01  MESSAGE-OUT.
003500     05  GOOD-MORNING               PIC X(14)
003600             VALUE "GOOD MORNING, ".
003700     05  USER-NAME                  PIC X(10).
003800     05  INSTRUCT                   PIC X(36)
003900          VALUE ".  CLEAR HOME AND ENTER ANY MESSAGE.".
004000     05  FILLER                     PIC X(1880)  VALUE SPACES.
004100
004200 PROCEDURE DIVISION.
004300 OPEN-REMOTE.
004400     OPEN I-O RMTE.
004500 READ-AND-WRITE.
004600** A USER KEYS IN "HELLO " THEN HIS USER CODE AND PASSWORD TO  **
004700**    GET ON THE SYSTEM.                                      **
004800     WAIT UNTIL READ-OK ON RMTE.
004900     READ RMTE AT END GO TO CLOSE-FILE.
005000     MOVE ACTUAL-RECORD-AREA TO MESSAGE-IN.
005100     MOVE SPACES TO ACTUAL-RECORD-AREA.
005200     MOVE 1940 TO LENGTH-IN-BYTES.
```

```
005300      WRITE DEPENDING-RECORD.
005400      MOVE USER-CODE TO USER-NAME.
005500      MOVE MESSAGE-OUT TO ACTUAL-RECORD-AREA.
005600      MOVE 60 TO LENGTH-IN-BYTES.
005700      WRITE DEPENDING-RECORD.
005800      WAIT UNTIL READ-OK ON RMTE.
005900*ENTER ANY MESSAGE AT THE REMOTE TERMINAL AND TRANSMIT**
006000      READ RMTE AT END GO TO CLOSE-FILE.
006100      MOVE SPACES TO MESSAGE-IN.
006200      MOVE LENGTH-IN-BYTES TO USER-CODE.
006300      MOVE "CHARACTERS = THE LENGTH OF YOUR MESSAGE"
006400           TO REST-OF-MESSAGE.
006500      MOVE MESSAGE-AREA TO ACTUAL-RECORD-AREA.
006600      MOVE 47 TO LENGTH-IN-BYTES.
006700      WRITE DEPENDING-RECORD.
006800 CLOSE-FILE.
006900      CLOSE RMTE.
007000      STOP RUN.
007100 END-OF-JOB.
```

Example 3:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.                          VARI-METHOD3.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                     B1000.
000800 OBJECT-COMPUTER.                     B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT   RMTE                   ASSIGN TO REMOTE.
001400
001500 DATA DIVISION.
001600
001700 FILE SECTION.
001800
001900 FD  RMTE
002000     RECORD CONTAINS 280 TO 680 CHARACTERS.
002100
002200 01  OCCURS-RECORD.
002300     05  FILLER                       PIC X(120).
002400     05  VARIABLE-PART.
002500         10  FIRST-PART               PIC X(80).
002600         10  SECOND-PART              PIC X(80)
002700            OCCURS 1 TO 6 TIMES DEPENDING ON Z.
002800
002900 WORKING-STORAGE SECTION.
003000 77  Z                                PIC 9 COMP VALUE 1.
003100 01  ENTREES.
003200     05  FILLER                   PIC X(15) VALUE "ONION SOUP     ".
003300     05  FILLER                   PIC X(65) VALUE SPACES.
003400     05  FILLER                   PIC X(15) VALUE "FRUIT SALAD    ".
003500     05  FILLER                   PIC X(65) VALUE SPACES.
003600     05  FILLER                   PIC X(15) VALUE "HAMBURGER      ".
003700     05  FILLER                   PIC X(65) VALUE SPACES.
003800     05  FILLER                   PIC X(15) VALUE "FRENCH FRIES   ".
003900     05  FILLER                   PIC X(65) VALUE SPACES.
004000     05  FILLER                   PIC X(15) VALUE "ICED TEA       ".
004100     05  FILLER                   PIC X(65) VALUE SPACES.
004200     05  FILLER                   PIC X(15) VALUE "ICE CREAM      ".
004300     05  FILLER                   PIC X(65) VALUE SPACES.
004400 01  ENTREE-TABLE REDEFINES ENTREES.
004500     05  ENTREE                   PIC X(80) OCCURS 6.
004600
004700 PROCEDURE DIVISION.
004800 OPEN-REMOTE.
004900     OPEN I-O RMTE.
005000     MOVE SPACES TO OCCURS-RECORD.
005100     MOVE "   LUNCH MENU: " TO FIRST-PART.
005200     PERFORM WRITE-REMOTE VARYING Z FROM 1 BY 1 UNTIL Z > 6.
005300     CLOSE RMTE.
005400     STOP RUN.
005500 WRITE-REMOTE.
005600     MOVE ENTREE(Z) TO SECOND-PART(Z).
005700     WRITE OCCURS-RECORD.
005800 END-OF-JOB.
```

COBOL74 CD (COMMUNICATION DESCRIPTION) FILES

In addition to the use of queue files and remote files, another method exists in the COBOL74 language of implementing communication between a user program and remote terminals. This is described in the COMMUNICATION SECTION and refers to a CD (Communication Description) file. The programmer specifies a symbolic queue file name, and, by means of the associated CD statement parameters, messages are passed to and from an actual physical file. The actual file is identified by the symbolic queue name. The symbolic queue name must be the name of a remote file as defined in the network controller if executing under the SMCS program. When the COBOL74 program does the first RECEIVE or ENABLE INPUT operation, the operating system looks for a COBOL74 queue file already open, with the same name as the symbolic queue name specified in the appropriate CD of the COBOL74 program. If no such queue file is open, the operating system opens the file, with the name as specified in the symbolic queue name. If such a queue file is already open, messages to and from this program are directed to the queue file of the same name. Figure F-4 is a diagram of the concept of COBOL74 communication description (CD) files.

For more information and a sample program using COBOL74 CD files, refer to COMMUNICATION SECTION in section 6 of this manual.

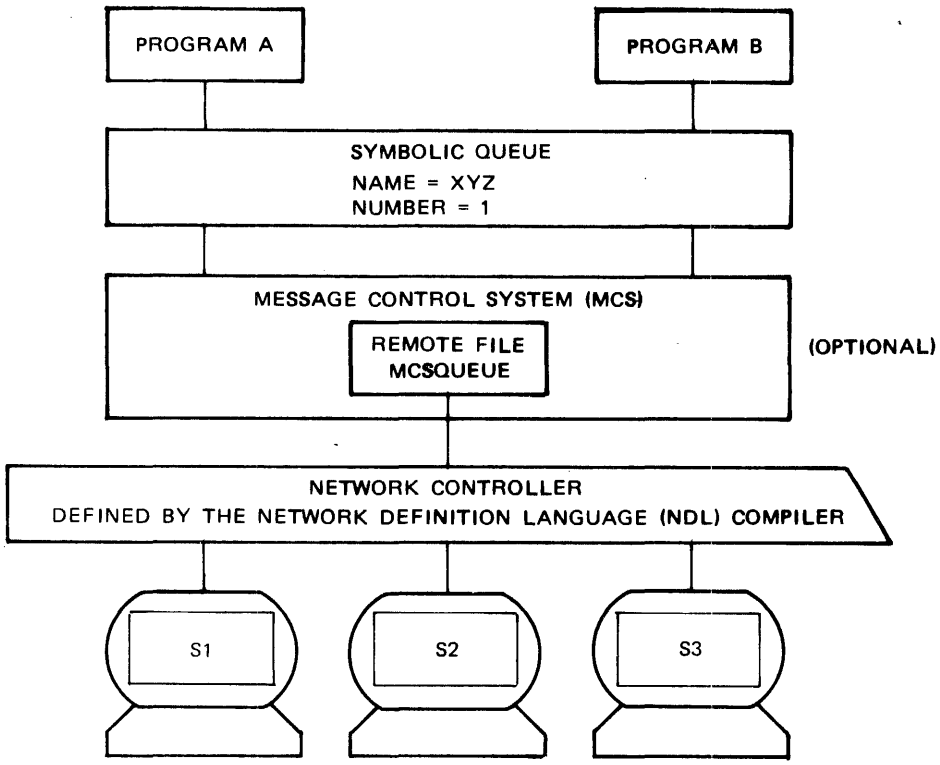


Figure F-4. COBOL74 Communication Description (CD) Files

PORT FILES

Port files permit communication between two processes across a Burroughs Network Architecture (BNA) network. The communication path must be established between both processes before any messages may be read or written. A port file may contain subfiles that enable a single process to communicate with several other processes (one per subfile). For bidirectional message traffic, a single subfile is used. The BNA network keeps track of the two paths.

Port subfiles may communicate across machine boundaries, enabling a program to communicate with other Burroughs host computers. Figure F-5 is a diagram of a possible port/subport communication path.

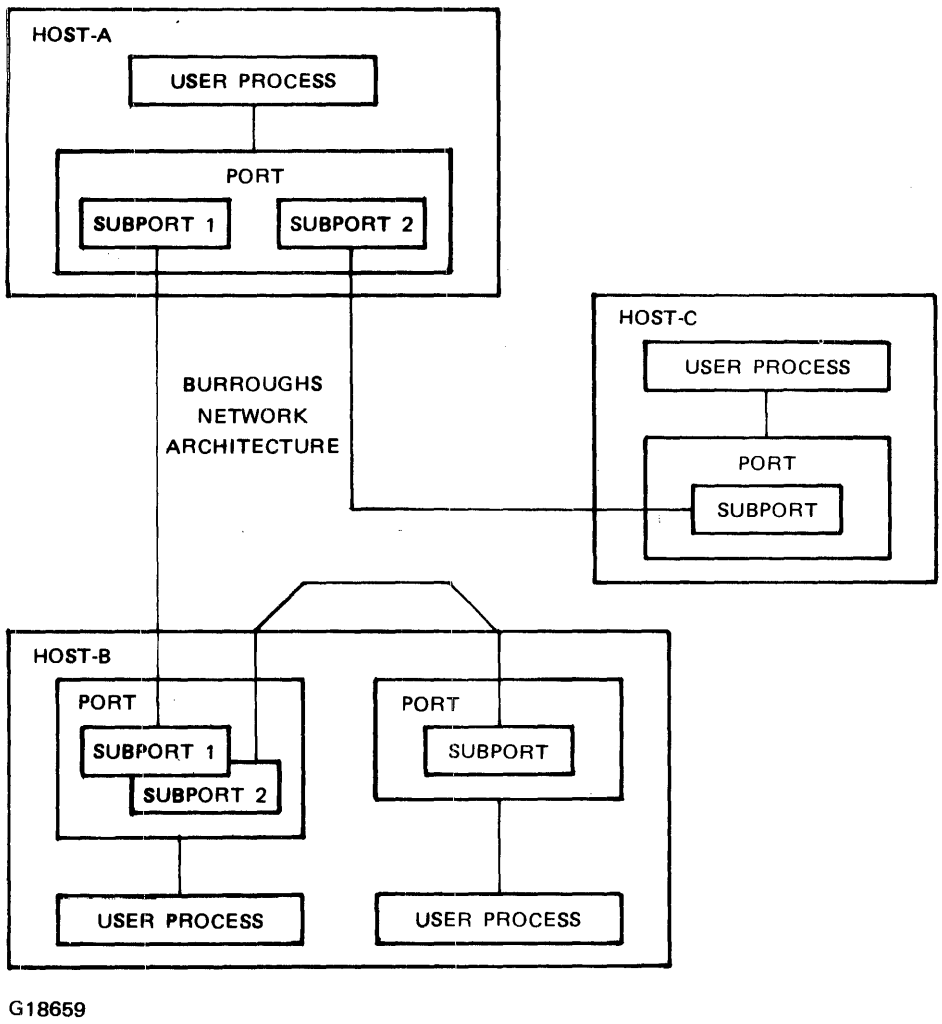


Figure F-5. Port/Subport Communication Path across BNA Network

For more information on PORT files, how to use them, and the file attributes to be used with them, refer to sections 5, 7, and 8 in this manual and the B 1000 Systems Burroughs Network Architecture (BNA) Installation and Operation Manual.

INTER-PROGRAM COMMUNICATION (IPC)

The Inter-Program Communication (IPC) module allows communication and data sharing from one program to one or more other programs within the same run unit. This concept can be referenced in section 6 under LINKAGE SECTION in this manual.

ROLE OF THE MESSAGE CONTROL SYSTEM (MCS)

Any program that opens a remote file with the HEADERS option is a Message Control System (MCS) and can control the stations associated with that file as defined in the network controller. In addition, an MCS can attach and detach unassigned stations from the station list. When there is no MCS present, message transfer and control of the data communications files is handled by the operating system with strict adherence to the remote file station list declarations in the network controller. Two of the more commonly used Burroughs MCS products are briefly discussed below along with the COBOL74MCS program.

Supervisory Message Control System (SMCS)

When multiple COBOL74 programs are executed (EX) under the SMCS program, they can open remote files with the same remote file name. Individual remote files are created with the same name, unique remote file numbers and the file attribute NUMBER.STATIONS equal to 1. When multiple users sign on (ON <program name>) and the NUMBER.STATIONS is greater than 1, then the SMCS program attempts to attach subsequent stations to the original remote file within the limits of the network controller. Refer to the B 1000 Systems SMCS Installation, Operation and Functional Description Manual for additional information.

Likewise, under the SMCS program, multiple COBOL74 programs can reference the same CD queue file name, but in this case, the programs physically share the same CD queue file (both name and number are the same). The SMCS program approves the second and any subsequent attachment of COBOL74 messages to an already established queue file or CD file.

Generalized Message Control System (GEMCOS)

Under the GEMCOS program (as under the SMCS program), multiple COBOL74 programs can open the same remote file name, creating individual remote files with the same name but unique numbers. The GEMCOS program can attach stations to an existing open remote file or deny the second or any subsequent open of the same remote file name.

GEMCOS allows the COBOL74 CD queue file sharing function; however, the receiver of any response is unpredictable since GEMCOS cannot control the distribution of messages from the queue to the remote files. If control is necessary, remote files should be used instead of CD files.

COBOL74MCS

The COBOL74MCS supports all capabilities of the ANSI 74 COBOL data communications module. This MCS program was designed to be run under the control of the SMCS program.

APPENDIX G
COBOL74 ISAM FILE CONCEPTS

INTRODUCTION

The Indexed I-O Module provides the capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one primary key within that record. Additionally, each record can contain up to 98 alternate keys, for which duplicates are allowed. The alternate keys allow versatility in the access path for the indexed file. The B 1000 COBOL74 compiler utilizes all features of the ANSI 74 syntax for the Indexed I-O module, including, but not limited to deletion of records, duplicates on the alternate keys, input-output error monitoring through the FILE STATUS clause and the USE procedure, and versatility in access method and/or path.

Enhancements to Burroughs B 1000 COBOL74 include 1) the AUDITED file attribute, which insures that each update to the file is completed and written to disk before the user program continues processing, 2) file handling procedures, which are resident within the operating system, as opposed to being compiled into the codefile of each program, 3) file management procedures that automatically recycle record slots, making it unnecessary to reorganize the file after deletions and additions, and 4) block control information (BCI) located within each block of the data file that enables re-creation of a corrupted ISAM file or creation of a new ISAM file with the added, deleted, or altered keys from an existing ISAM file.

A COBOL74 Indexed file consists of a minimum of three files (maximum of 101 files) referred to as the associated files in this appendix. The associated files can only be created or accessed as an Index Sequential Access Method (ISAM) file while all of the associated files are resident on disk.

ORGANIZATION

An ISAM file consists of a minimum of three separate but related files. For each Indexed file, there exists one global file, one data file, and one primary index file. If the file includes alternate keys (98 possible), there is one index file for each of the alternate keys as well. Figure G-1 demonstrates the relationship of the associated ISAM file structures.

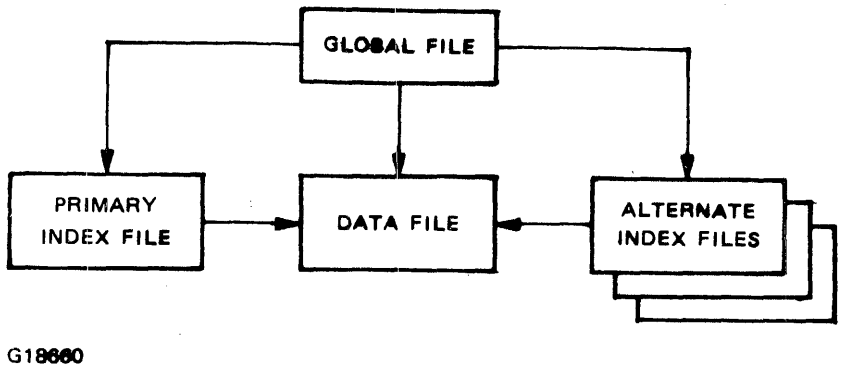


Figure G-1. Relationship of the ISAM File Structures

Global File Concepts

Every ISAM file has one global file with a file type of IS.G. This file, sometimes referred to as the cluster file, is created along with the data file, primary index file and alternate index files (if present), when a file with indexed organization is opened output. The external file identifier of the global file is also the name of the associated files collectively referred to "the ISAM file." Physical attributes of the global file are set by the operating system and cannot be modified in any way. The global file contains all the information needed by the operating system for management of the ISAM file. It must be present on disk whenever the ISAM file is opened. If corruption of the global file occurs, the ISAM file must be rebuilt as described in this appendix.

Some of the information fields in the global file are listed below:

Name and location of all subfiles
Number of users
Number of users with file open for updates
Structure information
Version number
Disk file header extension
File audited or not audited

The disk file header extension keeps track of the space available in the data file where new records can be added. This results in automatic recycling of record slots when records are deleted.

Data File Concepts

Every ISAM file has one file containing the data records. IS.D is the B file type for this file. It tells the operating system that the physical structure of the file is exactly like any other relative file with the exception that this file is one of the associated files of an ISAM file as well.

Physical Attributes

An ISAM file may be created using operating system default values. Alternatively, file attribute values describing physical size and maximum population of the ISAM data file may be declared by the user in the file description (FD) statement of the program creating the ISAM file. These values are also used by the operating system to establish physical sizes of the index files when created. Size-related file attributes include maximum number of areas, record size, records per block, blocks per area, and area length.

The relationship of the size-related attributes of a relative file or an ISAM data file is given in the following formula:

Formula 1:

$$(\text{blocksize}) * (\text{blocks per area}) = (\text{area length})$$

For an ISAM data file or a relative file, block control information (BCI), must be included when calculating blocksize as shown below:

Formula 2:

$$(\text{blocksize}) = (\text{record size} * \text{records per block}) + \text{BCI}$$

When an ISAM file is created, or records are added to the file, the operating system opens new areas for the data and index files as required. When the file is first created, the default value of 25 maximum areas is used by the operating system unless overridden by the VALUE OF AREAS clause in the FD statement of the program. The value of maximum number of areas of the ISAM data file is the only size-related file attribute that can be changed after the file has been created without re-creating the ISAM file and changing the programs that access that file. The maximum number of areas may increase to the operating system maximum of 105 when the FLEXIBLE attribute is set to TRUE within the program.

NOTE

Use of the FLEXIBLE attribute to increase the maximum number of areas for an existing ISAM file can cause one or more of the subfiles to exceed their allocated space before the data file reaches the limit of 105 areas. In this case, the file must be re-created with a user program.

The value of record size may be specified by the RECORD CONTAINS integer CHARACTERS clause of the FD statement. If this clause is not included, the operating system obtains the record size from the largest record description entry for the file. The minimum record size for an ISAM data record is 4 bytes.

The default value for records per block is 1 unless overridden by the BLOCK CONTAINS integer RECORDS clause of the FD statement in the source program. Unlike other file types in the B 1000 system, blocking an ISAM file with the default value will not insure that as each record is updated it will be written to disk. When the AUDITED file attribute is set to TRUE for the ISAM file, each update is written to disk before control returns to the user program.

The operating system default value for blocks per area is 500, but this can be overridden by the user in two separate ways. The value of blocks per area cannot be explicitly specified within the B 1000 Systems COBOL74 syntax, but it can be implicitly specified by including the VALUE OF AREALENGTH clause in the FD statement as seen in the above formulas. The desired number of blocks per area may also be controlled by the user by modifying the BLOCKS.PER.AREA file attribute of the code file for the program creating the ISAM file.

Example:

MODIFY <program name> FILE <file name> BLOCKS.PER.AREA = 100

This must be done prior to the creation of the ISAM file. If the VALUE OF AREALENGTH clause has not been specified and BLOCKS.PER.AREA has not been modified, the operating system uses the default value for the ISAM data file.

The largest value that can be specified for AREALENGTH is 2097151. However, as with the previously listed size-related file attributes, declaring a VALUE OF AREALENGTH is optional. If the value of blocksize exceeds 4194 bytes, user specification of a VALUE OF AREALENGTH or modification of BLOCKS.PER.AREA to a value less than 500 is required. This can be seen by substituting 4194 for the blocksize and the default of 500 blocks per area into formula 1:

(4194) * 500 = 2097000

An understanding of block control information (BCI) is necessary when calculating blocksize for an ISAM data file and also when choosing optimum values for record size and records per block.

Block Control Information (BCI)

Data storage on disk media includes the concepts of bits, bytes and segments. A byte consists of eight bits and every 180 bytes on disk represents a new segment. A bit can be on (1) or off (0). A digit of information consists of four bits and can be represented by the bit pattern (example: 1011) or by the hexadecimal values, 0-9 and A-F. The hexadecimal representation of a byte with all bits on is @FF@. The (@) character delimits a hexadecimal value. A RECORD is defined as the logical unit of data made available to a program in an INPUT or OUTPUT command. A BLOCK is the physical unit of data that is read from disk (INPUT) or written to disk (OUTPUT). A SEGMENT is the smallest unit of disk that can be located and read from or written to. All space from the end of the block to the end of the segment or segment multiple is wasted. A knowledge of these concepts is assumed in the following discussion.

The structure of an ISAM data file is exactly the same as that of any other B relative file. In relative file organization, each record is uniquely identified by an integer value greater than zero. This integer specifies the logical ordinal position of the record in the file. As each record is added to a relative file, a "presence" bit is turned on (a value of 1 is on, 0 is off). Block control information (BCI) consists of one presence bit for each record in the block plus up to seven extra bits to force the BCI to end on a byte boundary. An ISAM file has the added capability of record deletion. When a record is deleted, the presence bit for that record is turned off and information in the disk file header extension is updated to reflect a free record slot.

Each physical block of an ISAM data file (or relative file) starts on a segment boundary with the block control information. All disk space from the end of the physical block to the start of the next block (at the segment boundary) is allocated to the file but is wasted. The BCI for all files with blocking factors of 1 through 8 records per block is the same; one byte. In each case, however, only the presence bits for the block are significant. Unused bits in the BCI of an ISAM data file are not initialized. Similarly, the BCI of an ISAM file blocked 10 records per block consists of two bytes; 10 bits significant and 6 bits insignificant as shown. Table G-1 shows the BCI statistics for several sample blocks.

Table G-1. BCI Statistics For Several Blocking Factors

Records per Block	BCI (bytes)	Records Deleted	Bit Pattern
2	1	none	{11xxxxxx}
4	1	#2	{1011xxxx}
8	1	none	{11111111}
10	2	#5, #8	{1111011011xxxxxx}
13	2	#13	{1111111111110xxx}

x = unused bits of the BCI byte

An ISAM data file may be opened independently as a relative file. At this time the user need not be concerned with eliminating invalid records. The operating system handles this by using the BCI and the records per block value in the disk file header.

Efficient Blocking of the Data File

Careful calculation of ISAM blocksize from formula 2 above and a knowledge of disk segmentation may result in optimum disk utilization. Table G-2 lists the percent of utilization of disk space for various record sizes and blocking factors for an ISAM file.

Table G-2. ISAM Disk Utilization

Record size (bytes)	Records per block	BCI (bytes)	Blocksize (bytes)	Wasted (bytes)	Percent Utilization
32	255	32	8192	808	91.00
89	1	1	90	90	50.00
89	2	1	179	1	99.44
90	1	1	181	179	50.27
90	9	2	812	88	90.22
440	16	2	7042	158	97.80
440	18	3	7923	177	97.81

The maximum blocksize allowable for an ISAM data file is 8192 bytes.

Index File Concepts

For every ISAM file there is one primary index file and one index file for each of the alternate keys. There can be up to 98 alternate index files. All keys are alphanumeric and may overlap, but the left-most character of overlapping keys may not start in the same location. Primary keys are unique, whereas alternate keys may have duplicates if declared that way when file is originally created. Duplicate keys are returned on INPUT in their order of introduction to the file.

An index file consists of coarse tables and/or fine tables, collectively referred to as index tables. The index table entries each contain a key and an address. For a fine table, the address points to the location of the data record in the ISAM data file for that key. Fine tables are linked in one direction for efficient sequential access to the data records. Coarse tables exist when there is more than one fine table. An address entry in the coarse table points to a location in a fine table. The "coarsest" coarse table is called a root table. This exists when there is more than one coarse table. Figure G-2 is a simplified diagram of an index file with pointers into the data file.

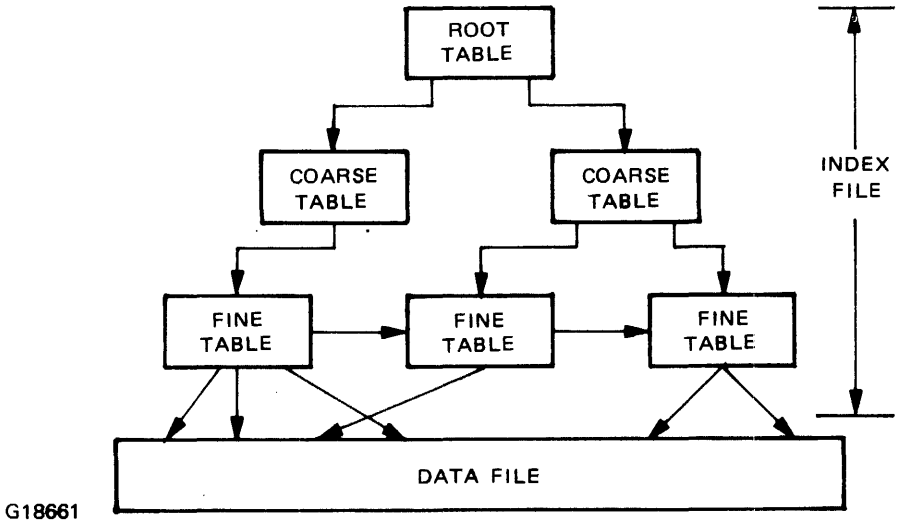


Figure G-2. The ISAM Index and Data Files

Naming Convention for the ISAM File Structures

B 1000 ISAM files have been designed so that internal file management is handled completely by the operating system. The naming convention for ISAM files is as follows:

The internal-file-name is the name assigned to a file by the programmer in the SELECT statement of the source program. The external-file-name is the name assigned to a file on disk by the operating system during creation of the file. The external-file-name has the following form:

<family-name>/<first-name>/<second-name>.

Family-name is the pack identifier of the file. First-name is always present. When file security is utilized, second-name is the file identifier and first-name is the user code. Second-name is optional when there is no file security. The B operating system limits each part of the external-file-name to ten characters. This nomenclature is explained further in the B 1000 Systems System Software Operation Guide, Volume 1. It is possible to change the external-file-name after creation, but for purposes of this discussion, the external-file-name is assigned at creation time and is dependent on how the creation program is executed and the VALUE OF TITLE clause (if present) in the source program. An ISAM file is referred to by the name of its global file.

File Creation Without a User Code

When an ISAM file is not created under a user code and the VALUE OF TITLE clause has not been specified, the associated files reside on the system disk and do not have a second-name. Their first-names are assigned using the ISAM naming convention: in this case a 2-digit numeric prefix is attached to the internal-file-name forcing truncation of the internal-file-name if it has more than eight characters. The first-names of the associated files are shown below.

File Type	First Name
global	= <internal-file-name>
data	= <00internal-file-name>
primary index	= <01internal-file-name>
alternate index	= <02internal-file-name> through <99internal-file-name>

When the VALUE OF TITLE syntax is included in the source program, specification of a family-name overrides the system disk default. If both first-name and second-name are specified through the VALUE OF TITLE syntax, it is necessary for the second-names to be unique in the first eight digits. The external file identifiers are as follows:

File Type	External File Identifier
global	= <family-name>/<first-name>/<second-name>
data	= <family-name>/<first-name>/<00second-name>
primary index	= <family-name>/<first-name>/<01second-name>
alternate index	= <family-name>/<first-name>/<02second-name> through <99second-name>

When a second-name is not specified, the ISAM naming convention applies to the first-name as follows:

File Type	External File Identifier
global	= <family-name>/<first-name>/
data	= <family-name>/<00first-name>/
primary index	= <family-name>/<01first-name>/
alternate index	= <family-name>/<02first-name>/ through <99first-name>/

File Creation Under a User Code

When an ISAM file is created under a user code and a VALUE OF TITLE clause is not specified, the user code becomes the first-name and the internal-file-name becomes the stem for attaching the numeric prefix as above, but now this becomes the second-name of the external-file-name. The default pack for the user code is the family-name.

File Type	External File Identifier
global	= <default pack>/<(user code)>/<internal-file-name>
data	= <default pack>/<(user code)>/<00internal-file-name>
primary index	= <default pack>/<(user code)>/<01internal-file-name>
alternate index	= <default pack>/<(user code)>/<02internal-file-name> through <99internal-file-name>

When an ISAM file is created under a user code, the VALUE OF TITLE clause can override the default pack; otherwise, the family-name is the default pack for the user code. An ISAM file created under a user code will always have a first-name and second-name. The VALUE OF TITLE clause can override the user code as the first-name, but the ISAM naming convention for the associated files is always applied to the second-name of the file.

When an ISAM file is opened output, all of the associated files are created and entered into the disk directory at that time, automatically removing any existing files with the same names. For this reason, care should be taken to insure that names to which the ISAM naming convention is applied are unique to the first eight characters. For example, if the ISAM file User1/(SBPC)/Inventory exists and the file User1/(SBPC)/Inventors is opened output, creation of the data file for the Inventors file, User1/(SBPC)/00inventor, will remove the data file for the Inventory file. The same thing occurs for the primary index file, and alternate index files, if they exist.

Changing the Name of an ISAM File

The B 1000 Systems operating system command CHANGE can be used to change the name of any of the associated files of an ISAM file individually, but this renders the files unusable as an ISAM file. The global file contains information needed by the operating system to process the associated files as an ISAM file. This information can only be changed using the system utility program SYSTEM/ISMAINT as described later in this appendix.

ISAM ACCESS METHODS

The access method for an ISAM file can be SEQUENTIAL, RANDOM, or DYNAMIC. Programming examples are located at the end of this appendix.

Sequential access refers to access via ascending order of the key of reference. The primary key is the default key of reference. In order to change the access path, the START verb must be specified, with the data-name referring to the name of the key desired. For the necessary syntax for the START statement, refer to section 7 in this manual.

Random access requires specification of a key value. The key of choice is declared in the syntax of the READ, WRITE and DELETE statements (refer to section 7).

Dynamic access allows the user program to use either sequential access or access via key value.

MULTIPLE USERS OF AN ISAM FILE

An ISAM file may have more than one user at any one time and still retain its integrity. A user is loosely defined as an "opener" of the file. A program may open one ISAM file with more than one access method at the same time, and/or more than one program can open the file at the same time. The latter represents multiple users of the file. The internal-file-names within one program must be unique in the SELECT statements, while the VALUE OF TITLE clauses in the FD statements relate the internal-file-names to the same physical file for all users.

As each user opens the file, the operating system creates a File Information Block (FIB) connecting that user to the only copy of the global file in memory for the ISAM file. The FIB contains information specific to that user and remains in memory until the user closes the file.

The global file contains all the information needed for system management of the file. It points to and contains information on all of the structures of the ISAM file. When in memory, each block of the data file is one buffer, and each table of the index files is also one buffer. As many buffers as needed are in memory at any one time, but there is never more than one copy (the latest) of any one block. There is a current-pointer for each user that associated with each index file in memory. This current maintains the pointer into that index file for the user.

Since there is only one copy of any of the ISAM structures in memory at any time, each user has access to the most recent data for the ISAM file. Simultaneous updating of an ISAM file does not occur since a buffer being updated is marked by the operating system and cannot be used by any other process until that update is completed. Figure G-3 is a simplified diagram relating two users and one ISAM file in memory. STRUCTURE 0 represents the data file and STRUCTURE 1 represents the primary index file.

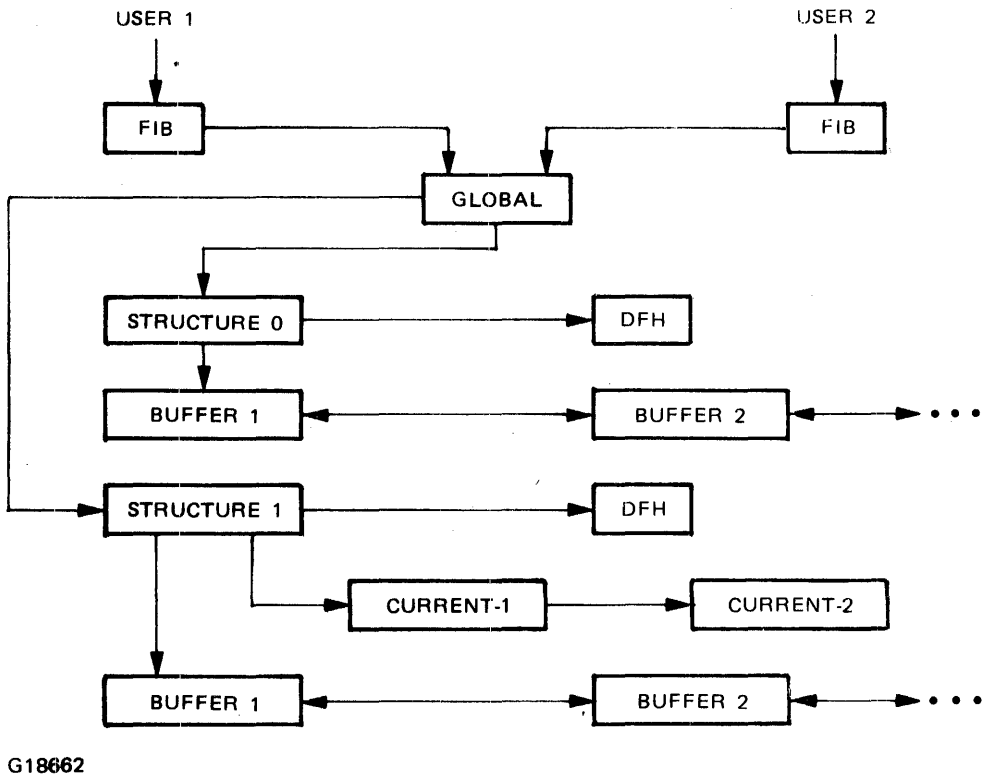


Figure G-3. Relationship of Two Users to the ISAM Structures

When an update to a record in an ISAM file occurs, each buffer that has been changed is marked by the operating system as needing to be written to disk. As long as the operating system has no demand on the memory space, those buffers remain in memory. The actual physical file on disk is not updated until the I/O occurs and the buffers are written to disk. When an update user closes the file, the ISAM file is written from memory to disk even if other users remain attached to the ISAM file.

The AUDITED file attribute was introduced for ISAM files as a patch to the B 1000 System Software in the Mark 10.0 release. Use of this attribute reduces the possibility for data loss in case of a system halt when an ISAM file is open for update.

The AUDITED File Attribute

The AUDITED file attribute is especially designed for maintaining the integrity of an ISAM file in an online environment where the actual I/O time is small compared to the elapsed time. When VALUE OF AUDITED IS TRUE is specified in the FD description, or the codefile is modified to true for this attribute of the ISAM file, each update to the file is written to disk before control is returned to the program. If a system halt occurs when an audited ISAM file is open for update, recovery procedures for that file are simplified considerably, as seen in table G-3.

When an ISAM file is open for update, it is possible that the operating system is 1) updating the global file or one of the index files, 2) updating the data file, or 3) not doing an update at all. Recovery of an ISAM file after a CLEAR/START depends on which of these three possibilities was happening and whether or not the file was AUDITED. Table G-3 lists the recovery procedures for all cases. The entries in this table reflect operating system messages that are issued the first time the file is opened as an ISAM file after a CLEAR/START operation.

Table G-3. ISAM File Recovery

	AUDITED	NOT AUDITED
Global or index file update in process	Must rebuild or OK to continue	Must rebuild or run SYSTEM/ISVERIFY to verify validity of file
Data file update in process	Message indicates possible loss of one data record, continuation is automatic	Must rebuild or run SYSTEM/ISVERIFY to verify validity of file
No updates in process	Message indicates file open for update during system halt, continuation is automatic	Must rebuild or run SYSTEM/ISVERIFY to verify validity of file

When there are multiple users of an ISAM file, the file remains in an audited state until all users specifying AUDITED have closed the file. Since only one copy of the file is in memory, while one user is auditing, all users are auditing.

Rebuilding an ISAM file can be accomplished by a user program similar to the example at the end of this appendix or by use of the B 1000 system utility program, CREATE/ISAM.

SYSTEM UTILITY PROGRAMS FOR ISAM FILE MAINTENANCE

Three B 1000 system utility programs exist for specific use with ISAM files: CREATE/ISAM, SYSTEM/ISVERIFY, and SYSTEM/IS-MAINT. They are explained briefly below. Refer to the B 1000 Systems System Software Operation Guide, Volume 2, for more detailed information on each program.

CREATE/ISAM

The purpose of this utility program is to generate an ISAM file from an existing data file. The existing data file can be an ISAM data file or a sequential data file.

The CREATE/ISAM utility program may be executed by means of three interfaces; a screen interface, an ODT interface and a card interface.

This utility program can be used to create the associated files on different disk packs or on a default pack. It can be used to build an ISAM file with added, deleted or altered keys for special purposes, while maintaining the original file. In the case of file recovery, the utility program can be used to re-create an ISAM file whose global or index files have become corrupt.

The CREATE/ISAM utility program should not be used to rebuild an ISAM file if new blocking is desired. The blocking factors of the input data file are used to create the blocking factors of the new ISAM data file. The user has control over this information only with a user program.

The CREATE/ISAM utility program does not allow creation of an ISAM file whose output data file has the same file identifier as the input data file.

SYSTEM/ISVERIFY

The purpose of this utility program is to verify the integrity of an ISAM file. The program will:

List information contained in the global file
List structure information for data and index files
Verify mapping of index file(s) onto the data file
Verify global file information
Verify the overall structure of all index files
Notify the user about the condition of the ISAM file

When the SYSTEM/ISVERIFY utility program finds that there is no problem with the integrity of the file, it changes the integrity bit in the cluster file allowing user access to the ISAM file.

It is not possible for any system utility program to verify the integrity of the data within the ISAM data file. This can only be accomplished with a user program, because only the user knows what constitutes valid data in his file.

SYSTEM/IS–MAINT

This system utility program performs library maintenance functions on ISAM files. A few of the more important commands and their functions are listed below.

The CHANGE command is used to change the names of an ISAM file in the disk directory. At the same time, it modifies the data in the global file to reflect the new names. This command can only be used when the pack identifier remains the same.

The COPY command may be used to copy all of the associated files of an ISAM file to another disk or to tape. Only the name of the global file need be entered, no matter how many index files exist. If the file is copied to another disk, the update command must be used to modify the information within the global file.

The LIST ODT command lists the names of all the associated files of an ISAM file. It is good practice to use this command after any change to verify the accuracy of the changes.

The UPDATE command is the only way to correct the information within the global file when it becomes necessary to move an ISAM file or subfiles of the ISAM file to another disk pack.

RPG COMPATIBILITY

COBOL74 ISAM files can be accessed or created by RPG programs as well as COBOL74 programs. If an ISAM file is to be accessed by RPG programs, the key length must be limited to 23 alphanumeric bytes. For further information refer to the B 1000 Systems Report Program Generator (RPG) Reference Manual.

PROGRAMMING EXAMPLES

Three COBOL74 source programs are listed below. Each program demonstrates a variety of features and constructs described earlier in this appendix.

Creating an ISAM File

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CREATE-ISAM.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                  B1000.
000800 OBJECT-COMPUTER.                  B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT  MYISAMFILE  ASSIGN TO DISK
001400             ORGANIZATION IS INDEXED
001500             ACCESS IS DYNAMIC
001600             RECORD KEY IS            EMPL-NUM
001700             ALTERNATE RECORD KEY IS  EMPL-NAME
001800             ALTERNATE RECORD KEY IS  DEPARTMENT
001900                       WITH DUPLICATES
002000             FILE STATUS IS           ISAM-STAT.
002100
002200 DATA DIVISION.
002300
002400 FILE SECTION.
002500 FD  MYISAMFILE
002600        RECORD CONTAINS 89 CHARACTERS
002700        BLOCK CONTAINS 16 RECORDS
002800        VALUE OF TITLE IS ISAM-NAME
002900        VALUE OF AREAS IS 105
003000        VALUE OF AUDITED IS FALSE.
003100
003200 01  ISAM-REC.
003300     05  EMPL-NUM.
003400         10  SOC-SEC                PIC X(11).
003500         10  DEPARTMENT             PIC X(4).
003600     05  EMPL-NAME                  PIC X(20).
003700     05  FILLER                     PIC X(54).
003800
003900 WORKING-STORAGE SECTION.
004000 01  ADD-COUNT                      PIC 9(4)  VALUE ZERO.
004100 01  REJECT-COUNT                   PIC 9(4)  VALUE ZERO.
004200 01  ADD-MORE-FLAG                  PIC X(3)  VALUE "YES".
004300     88  THATS-ALL      VALUE "NO".
004400 01  ISAM-STAT                      PIC XX    VALUE SPACES.
004600     88  WRITEOK        VALUE "00", "02".
004700 01  ISAM-NAME                      PIC X(35) VALUE SPACES.
004800
```

```
004900 PROCEDURE DIVISION.
005000
005100 PUT-FILE-IN-DIRECTORY.
005200     DISPLAY "THIS PROGRAM IS GOING TO REMOVE OLD FILE ".
005300     MOVE "(KARIN)/C74ISAM ON S" TO ISAM-NAME.
005400     OPEN OUTPUT MYISAMFILE.
005500     CLOSE MYISAMFILE.
005600**  ALL OF THE ASSOCIATED FILES ARE IN THE DISK          **
005700**  DIRECTORY, AND CAN BE ACCESSED BY OTHER USERS AT     **
005800**  THIS TIME.                                           **
005900
006000 LOAD-DYNAMIC.
006100     CHANGE ATTRIBUTE AUDITED OF MYISAMFILE TO TRUE.
006200     OPEN I-O MYISAMFILE.
006205     IF ISAM-STAT NOT = "00"
006210        DISPLAY "STATUS IS " ISAM-STAT.
006300     PERFORM GET-INFORMATION UNTIL THATS-ALL.
006400     PERFORM DISPLAY-TOTALS.
006500     CLOSE MYISAMFILE SAVE.
006600     STOP RUN.
006700
006800 GET-INFORMATION.
006900     DISPLAY "ENTER EMPLOYEE NAME: LAST NAME, FIRST NAME".
007000     ACCEPT EMPL-NAME.
007100     DISPLAY "ENTER SOCIAL SECURITY NUMBER: XXX-XX-XXXX".
007200     ACCEPT SOC-SEC.
007300     DISPLAY "ENTER 4-DIGIT DEPARTMENT NUMBER".
007400     ACCEPT DEPARTMENT.
007500**  VALIDATION PROCEDURES FOR EACH ENTRY GO HERE.        **
007600     PERFORM RANDOM-WRITE.
007700     IF WRITEOK
007800         DISPLAY "ENTRY ADDED"
007900     ELSE
008000         DISPLAY "ENTRY NOT ADDED, STATUS = " ISAM-STAT.
008100     DISPLAY "MORE RECORDS TO ADD,  YES OR NO? "
008200     ACCEPT ADD-MORE-FLAG.
008300
008400 RANDOM-WRITE.
008600     WRITE ISAM-REC   INVALID KEY
008800         ADD 1 TO REJECT-COUNT.
008900     IF WRITEOK
009000         ADD 1 TO ADD-COUNT.
009100
009200 DISPLAY-TOTALS.
009300     DISPLAY "RECORDS ADDED = " ADD-COUNT.
009400     DISPLAY "RECORDS REJECTED = " REJECT-COUNT.
009500     DISPLAY " ****** GOOD-BYE ******".
```

Updating an ISAM File

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. UPDATE-ISAM.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                    B1000.
000800 OBJECT-COMPUTER.                    B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT  MYISAMFILE  ASSIGN TO DISK
001400             ORGANIZATION IS INDEXED
001500             ACCESS IS DYNAMIC
001600             RECORD KEY IS              EMPL-NUM
001700             ALTERNATE RECORD KEY IS    EMPL-NAME
001800             ALTERNATE RECORD KEY IS    DEPARTMENT
001900                       WITH DUPLICATES
002000             FILE STATUS IS             ISAM-STAT.
002100
002200 DATA DIVISION.
002300
002400 FILE SECTION.
002500 FD  MYISAMFILE
002600        VALUE OF TITLE IS ISAM-NAME
002700        VALUE OF AUDITED IS TRUE.
002800
002900 01  ISAM-REC.
003000     05  EMPL-NUM.
003100         10  SOC-SEC                  PIC X(11).
003200         10  DEPARTMENT               PIC X(4).
003300     05  EMPL-NAME                    PIC X(20).
003400     05  EVENT                        PIC X(10).
003500     05  EVENT-DATE                   PIC 9(6).
003600     05  FILLER                       PIC X(38).
003700
003800 WORKING-STORAGE SECTION.
003900 01  ISAM-STAT                        PIC XX    VALUE SPACES.
004000     88 DELETEOK        VALUE "00".
004100     88 READOK          VALUE "00", "02".
004200     88 WRITEOK         VALUE "00", "02".
004300     88 END-OF-FILE     VALUE "10".
004400     88 INVALID-KEY     VALUE "21", "22", "23", "24".
004500     88 END-DUP-KEY     VALUE "00".
004600 01  ISAM-NAME                        PIC X(35) VALUE SPACES.
004700
004800 PROCEDURE DIVISION.
004900
```

```
005000 MAIN-DRIVER.
005100     MOVE "(KARIN)/C74ISAM ON S" TO ISAM-NAME.
005200     OPEN I-O MYISAMFILE.
005300     PERFORM READ-SEQUENTIAL-DEFAULT.
005400     PERFORM READ-SEQUENTIAL-ALTERNATE.
005500     PERFORM ADD-RANDOM-WRITE.
005600     PERFORM UPDATE-RANDOM-REWRITE.
005700     PERFORM DELETE-RANDOM.
005800     CLOSE MYISAMFILE.
005900     STOP RUN.
006000
006100 READ-SEQUENTIAL-DEFAULT.
006200** AFTER OPEN, CURRENT RECORD POINTER FOR THIS USER IS **
006300** POSITIONED AT FIRST RECORD OF PRIMARY KEY.          **
006400     PERFORM READ-NEXT UNTIL END-OF-FILE.
006500     DISPLAY "THIS IS THE END OF THE FILE.".
006600
006700 READ-SEQUENTIAL-ALTERNATE.
006800**  IT IS NECESSARY TO CHANGE THE KEY OF REFERENCE     **
006900**  WHEN READING SEQUENTIALLY VIA AN ALTERNATE KEY.    **
007000**  THE START STATEMENT CHANGES THE KEY HERE.          **
007100     MOVE "PA&S" TO DEPARTMENT.
007200     START MYISAMFILE    KEY IS EQUAL TO DEPARTMENT
007300       INVALID KEY
007400         DISPLAY "NO RECORDS FOR DEPARTMENT PA&S".
007500
007600**  THE FILE STATUS IS "00" WHEN THE START IS          **
007700**  SUCCESSFUL.  IT BECOMES "02" AFTER A SUCCESSFUL    **
007800**  READ.....NEXT VIA AN ALTERNATE KEY PATH WITH       **
007900**  DUPLICATES.  ON THE LAST READ OF THE ALTERNATE KEY,**
008000**  THE FILE STATUS WILL CHANGE TO "00".  IN ORDER TO  **
008100**  USE THIS CHANGE OF STATUS TO MARK THE END OF THE   **
008200**  ALTERNATE KEY PATH, IT IS NECESSARY TO SET THE FILE**
008300**  STATUS TO A VALUE OTHER THAN "00" BEFORE THE FIRST **
008400**  READ.....NEXT OPERATION IS PERFORMED.              **
008500     IF ISAM-STAT  = "00"
008600        MOVE "XX" TO ISAM-STAT
008700        PERFORM READ-NEXT UNTIL END-DUP-KEY
008800        DISPLAY "END OF DEPARTMENT " DEPARTMENT.
008900
009000 READ-NEXT.
009100     READ MYISAMFILE NEXT AT END
009200        NEXT SENTENCE.
009300     IF READOK
009400        DISPLAY "SEQ READ ON " EMPL-NAME " " DEPARTMENT.
009500
```

```
009600 ADD-RANDOM-WRITE.
009700     MOVE "KO" TO EMPL-NAME.
009800     MOVE "571-92-2758UCSC" TO EMPL-NUM.
009900     MOVE "NEW HIRE  " TO EVENT.
010000     MOVE "850623" TO EVENT-DATE.
010100     WRITE ISAM-REC   INVALID KEY
010200        DISPLAY "INV. KEY ON WRITE " EMPL-NAME EMPL-NUM.
010300     IF WRITEOK
010400        DISPLAY "RECORD ADDED " EMPL-NAME EMPL-NUM.
010500
010600 UPDATE-RANDOM-REWRITE.
010700**  A REWRITE OPERATION MUST FOLLOW A SUCCESSFUL READ.  **
010800**  THEREFORE, IT IS IMPERATIVE TO CHECK FILE STATUS     **
010900**  AFTER THE READ, BEFORE THE REWRITE.                  **
011000     MOVE "565-50-7656PA&S" TO EMPL-NUM.
011100     READ MYISAMFILE INVALID KEY
011200        DISPLAY "INVALID READ ON " EMPL-NUM.
011300     IF READOK
011400        MOVE "PROMOTION " TO EVENT
011500        MOVE "840103" TO EVENT-DATE
011600        REWRITE ISAM-REC INVALID KEY
011700           DISPLAY "INVALID REWRITE ON " EMPL-NUM.
011800     IF WRITEOK
011900        DISPLAY "UPDATE OK ON " EMPL-NAME " " EMPL-NUM.
012000
012100 DELETE-RANDOM.
012200     MOVE "103-20-4373PA&S" TO EMPL-NUM.
012300     DELETE MYISAMFILE INVALID KEY
012400        DISPLAY "INVALID DELETE ON " EMPL-NUM.
012500     IF DELETEOK
012600        DISPLAY "DELETE OK ON " EMPL-NUM.
```

Rebuilding an ISAM File from an ISAM DATA File

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. REBUILD-ISAM.
000300 ENVIRONMENT DIVISION.
000400
000500 CONFIGURATION SECTION.
000600
000700 SOURCE-COMPUTER.                    B1000.
000800 OBJECT-COMPUTER.                    B1000.
000900
001000 INPUT-OUTPUT SECTION.
001100
001200 FILE-CONTROL.
001300     SELECT  OLDDATAFILE  ASSIGN TO DISK
001400             ORGANIZATION IS RELATIVE
001500             ACCESS IS SEQUENTIAL
001600             FILE STATUS IS            RELA-STAT.
001700
001800     SELECT  MYISAMFILE  ASSIGN TO DISK
001900             ORGANIZATION IS INDEXED
002000             ACCESS IS DYNAMIC
002100             RECORD KEY IS             EMPL-NUM
002200             ALTERNATE RECORD KEY IS   EMPL-NAME
002300             ALTERNATE RECORD KEY IS   DEPARTMENT
002400                       WITH DUPLICATES
002500             FILE STATUS IS            ISAM-STAT.
002600
```

```
002700 DATA DIVISION.
002800
002900 FILE SECTION.
003000 FD  OLDDATAFILE
003100         VALUE OF TITLE IS RELA-NAME.
003200**  THERE ARE 89 CHARACTERS PER RECORD,                    **
003300**             16 RECORDS PER BLOCK  AND                   **
003400**           500 BLOCKS PER AREA FOR THIS FILE, AS         **
003500**  CREATED BY THE USER PROGRAM, CREATE-ISAM. SINCE        **
003600**  THIS FILE IS OPENED INPUT, THESE VALUES DO NOT         **
003700**  HAVE TO BE INCLUDED IN THIS PROGRAM.                   **
003800
003900 01  RELA-REC                          PIC X(89).
004000
004100 FD  MYISAMFILE
004200         BLOCK CONTAINS 16 RECORDS
004300         VALUE OF TITLE IS ISAM-NAME
004400         VALUE OF AUDITED IS FALSE
004500         VALUE OF AREALENGTH IS 1426000.
004600**  ONE REASON FOR REBUILDING AN ISAM FILE WITH A USER     **
004700**  PROGRAM IS TO CHANGE THE BLOCKING FACTORS.  IN THIS    **
004800**  CASE, IT IS DESIRED TO MAKE THE CAPACITY OF THE NEW    **
004900**  ISAM FILE GREATER.  SPECIFYING A VALUE OF 1426000      **
005000**  FOR THE AREALENGTH WILL FORCE THE NUMBER OF BLOCKS     **
005100**  PER AREA TO 1000, WITH THE SAME INPUT VALUES OF 16     **
005200**  RECORDS PER BLOCK, AND 105 MAXIMUM AREAS.  THIS        **
005300**  DOUBLES THE CAPACITY OF THE NEW ISAM FILE BUT NONE     **
005400**  OF THE OLD UPDATE PROGRAMS NEED BE RECOMPILED.         **
005500**                                                         **
005600**  AUDITING A REBUILD PROGRAM SUCH AS THIS IS NOT         **
005700**  PRACTICAL SINCE IT WOULD BE LESS TIME CONSUMING TO     **
005800**  RE-EXECUTE THE PROGRAM IN CASE OF A FAILURE.           **
005900 01  ISAM-REC.
006000     05  EMPL-NUM.
006100         10  SOC-SEC                   PIC X(11).
006200         10  DEPARTMENT                PIC X(4).
006300     05  EMPL-NAME                     PIC X(20).
006400     05  EVENT                         PIC X(10).
006500     05  EVENT-DATE                    PIC 9(6).
006600     05  FILLER                        PIC X(38).
006700
006800 WORKING-STORAGE SECTION.
006900 01  RECORD-COUNTS.
007000     05  READ-COUNT                    PIC 9(6)  VALUE ZERO.
007100     05  GOOD-WRITE-COUNT              PIC 9(6)  VALUE ZERO.
007200     05  BAD-WRITE-COUNT               PIC 9(6)  VALUE ZERO.
007300 01  ISAM-STAT                         PIC XX    VALUE SPACES.
007400     88 WRITEOK          VALUE "00", "02".
007500     88 INVALID-KEY      VALUE "21", "22", "23", "24".
007600 01  RELA-STAT                         PIC XX    VALUE SPACES.
007700     88 READOK           VALUE "00", "02".
007800     88 END-OF-FILE      VALUE "10".
007900 01  ISAM-NAME                         PIC X(35) VALUE SPACES.
008000 01  RELA-NAME                         PIC X(35) VALUE SPACES.
008100
```

```
008200 PROCEDURE DIVISION.
008300
008400 REMOVE-OLD-ISAM.
008500** THE OPEN OUTPUT STATEMENT WILL REMOVE ALL OF THE     **
008600** ASSOCIATED FILES OF THE OLD ISAM FILE SO IT IS       **
008700** ABSOLUTELY NECESSARY TO CHANGE THE NAME OF THE OLD   **
008800** DATA FILE BEFORE THIS PROGRAM IS EXECUTED.           **
008900** EXAMPLE:                                            **
009000** *Z USER KARIN/BESTFRIEND CHANGE 00C74ISAM TO OLDDATA**
009100     MOVE "(KARIN)/C74ISAM ON S" TO ISAM-NAME.
009200     OPEN OUTPUT MYISAMFILE.
009300     CLOSE MYISAMFILE.
009400
009500 PROCESS-TRANSFER-OF-DATA.
009600     MOVE "(KARIN)/OLDDATA ON S" TO RELA-NAME.
009700     OPEN I-O MYISAMFILE.
009800     OPEN INPUT OLDDATAFILE.
009900     PERFORM READ-AND-WRITE-LOOP UNTIL END-OF-FILE.
010000     CLOSE MYISAMFILE
010100           OLDDATAFILE.
010200     PERFORM TOTALS.
010300     STOP RUN.
010400
010500 READ-AND-WRITE-LOOP.
010600     READ OLDDATAFILE AT END
010700        DISPLAY "END OF OLD DATA FILE".
010800     IF READOK
010900        ADD 1 TO READ-COUNT
011000        WRITE ISAM-REC FROM RELA-REC INVALID KEY
011100           ADD 1 TO BAD-WRITE-COUNT
011200           DISPLAY "INVALID WRITE ON " EMPL-NUM.
011300     IF READOK AND WRITEOK
011400        ADD 1 TO GOOD-WRITE-COUNT.
011500
011600 TOTALS.
011700     DISPLAY "READS ON OLD DATA FILE  = " READ-COUNT.
011800     DISPLAY "WRITES TO NEW ISAM FILE = " GOOD-WRITE-COUNT.
011900     DISPLAY "BAD WRITES TO NEW FILE  = " BAD-WRITE-COUNT.
012000     DISPLAY "****** GOOD-BYE ******".
```

INDEX

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

INDEX (CONT)

Documentation Evaluation Form

Title: B 1000 COBOL74 Language Manual

Form No: 1168622

Date: August 1984

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

From:

Name _____

Title _____

Company _____

Address _____

_____

Phone Number _____ Date _____

Remove form and mail to:

Burroughs Corporation
Corporate Documentation – West
1300 John Reed Court
City of Industry, CA 91745
U.S.A.