Burroughs
B 1700 SYSTEMS

COBOL
REFERENCE MANUAL

Burroughs

B 1700 Systems

COBOL

REFERENCE MANUAL

Burroughs Corporation
Detroit, Michigan 48232

$4.00

Printed in U.S. America 10-72 1057197

Burroughs Corporation believes the program described in this manual to be accurate and reliable, and much care has been taken in its preparation. However, the Corporation cannot accept any responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Sales Technical Services, Burroughs Corporation, 6071 Second Avenue, Detroit, Michigan 48232.

Table of Contents

Section Title Page

INTRODUCTION ix

1 COBOL LANGUAGE ELEMENTS 1-1
General 1-1
Character Set 1-1
Characters Used For Words 1-2
Punctuation Characters 1-2
Characters Used In Editing 1-2
Characters Used In Formulas 1-2
Characters Used In Relations 1-2
Definitions Of Words 1-3
Types of Words 1-3
Nouns 1-3
File-Name 1-3
Record-Name 1-3
Data-Name 1-3
Condition-Name 1-4
Procedure-Name 1-4
Literals 1-4
Numeric Literal 1-4
Non-Numeric Literal 1-5
Undigit Literals 1-6
Figurative Constant 1-6
Special Register Name 1-7
Tally 1-7
Todays-Date (Calendar) 1-7
Date (Julian) 1-7
Time 1-7
Special-Names 1-8
Verbs 1-8
Reserved Words 1-8
Connectives 1-8
Optional Words 1-8
Key Words 1-8
Statement And Sentence Formation 1-9
Paragraph Formation 1-9
Section Formation 1-9
Notation Used In Verbs And Entry Formats . . . 1-9
Key Words 1-9
Optional Words 1-9
Lower Case Words 1-9
Braces 1-10
Brackets 1-10
Consecutive Periods 1-10
Period 1-10

Table of Contents (cont)

Section Title Page

2 IDENTIFICATION DIVISION 2-1
General 2-1
Syntax Rules 2-1
MONITOR 2-3
Coding the IDENTIFICATION DIVISION 2-3

3 ENVIRONMENT DIVISION 3-1
General 3-1
Organization 3-1
Structure 3-1
Syntax Rules 3-1
Configuration Section 3-2
SOURCE-COMPUTER 3-2
OBJECT-COMPUTER 3-3
SPECIAL-NAMES 3-5
INPUT-OUTPUT SECTION 3-7
FILE-CONTROL 3-7
I-O-CONTROL 3-12
Coding the ENVIRONMENT DIVISION 3-14

4 DATA DIVISION 4-1
General 4-1
DATA DIVISION Organization 4-1
DATA DIVISION Structure 4-1
Record Description Structure 4-2
Level-Number Concept 4-2
Qualification 4-2
Tables 4-4
Subscripting 4-4
FILE SECTION 4-7
FILE DESCRIPTION 4-7
BLOCK 4-11
DATA RECORDS 4-13
FILE CONTAINS 4-14
LABEL 4-15
RECORD 4-18
RECORDING MODE 4-19
VALUE-OF-ID 4-20
RECORD DESCRIPTION 4-22
BLANK WHEN ZERO 4-25
Condition-Name 4-26
DATA-NAME 4-29
JUSTIFIED 4-30
Level-Number 4-31
OCCURS 4-32
PICTURE 4-35

Table of Contents (cont)

Section	Title	Page
4 (cont)	REDEFINES	4-48
	RENAMES	4-49
	USAGE	4-50
	VALUE	4-52
	WORKING-STORAGE SECTION	4-54
	Organization	4-54
	Non-Contiguous WORKING-STORAGE	4-54
	WORKING-STORAGE Records	4-55
	Initial Values	4-55
	Condition-Names	4-55
	Coding the WORKING-STORAGE SECTION	4-55
5	PROCEDURE DIVISION	5-1
	General	5-1
	Rules Of Procedure Formation	5-1
	Statements	5-1
	Imperative Statements	5-1
	Conditional Statements	5-2
	Compiler-Directing statements	5-2
	Sentences	5-2
	Imperative Sentences	5-2
	Conditional Sentences	5-2
	Compiler-Directing Sentences	5-2
	Sentence Punctuation	5-3
	Verb Formats	5-3
	Sentence Formats	5-3
	Execution of Imperative Sentences	5-3
	Execution of Conditional Sentences	5-3
	Execution of Compiler-Directing Sentences	5-4
	Control Relationship Between Procedures	5-4
	Paragraphs	5-4
	Sections	5-5
	DECLARATIVES	5-5
	USE Statement	5-5
	COPY Statement As A DECLARATIVE	5-6
	Arithmetic Expressions	5-6
	Arithmetic Operators	5-7
	Formation and Evaluation Rules	5-7
	Conditions	5-8
	Logical Operators	5-9
	Relation Condition	5-9
	Relational Operators	5-10
	Comparison of Operands	5-11
	Non-Numeric	5-11
	Numeric	5-11
	Evaluation Rules	5-11
	Simple Conditions	5-12

Table of Contents (cont)

Section	Title	Page
5 (cont)	Compound Conditions	5-12
	Abbreviated Compound Conditions	5-14
	Segmentation	5-15
	Program Segments	5-15
	Segment Classification	5-15
	Priority Numbers	5-16
	Internal Program Switches	5-18
	Verbs	5-18
	Specific Verb Formats	5-19
	ACCEPT	5-20
	ADD	5-21
	ALTER	5-24
	CLOSE	5-25
	COMPUTE	5-30
	COPY	5-31
	Library Creation	5-33
	DISPLAY	5-35
	DIVIDE	5-36
	END-OF-JOB	5-38
	EXAMINE	5-39
	EXIT	5-41
	GO TO	5-42
	IF	5-44
	Simple Conditional Tests	5-44
	Conditional Statements	5-44
	Relation Tests	5-44
	Relative Value Tests	5-45
	Class Test	5-45
	Conditional Variable Tests	5-46
	Not Logic	5-46
	MOVE	5-47
	MULTIPLY	5-51
	NOTE	5-52
	OPEN	5-53
	PERFORM	5-56
	READ	5-61
	RELEASE	5-64
	RETURN	5-65
	SEARCH	5-66
	SEEK	5-70
	SET	5-72
	SORT	5-74
	STOP	5-78
	SUBTRACT	5-79
	USE	5-81
	WRITE	5-84
	ZIP	5-87
	Coding the PROCEDURE DIVISION	5-87

Table of Contents (cont)

Section — Title — Page

6 DATA COMMUNICATIONS 6-1
General 6-1
Specific Verb Formats 6-1

7 CODING FORM 7-1
General 7-1
Coding Form Representation 7-1
Sequence Numbers (Columns 1-6) 7-1
Continuation Indicator (Column 7) 7-1
Continuation of Undigit Literals 7-3
Continuation of Non-Numeric Literals 7-3
Continuation of Words and Numeric Literals 7-3
Division Header 7-3
Section Header 7-3
Paragraph Names and Paragraphs 7-5
Data Division Entries 7-5
Declaratives 7-6
Punctuation 7-6

8 COBOL COMPILER CONTROL 8-1
General 8-1
Compilation Card Deck 8-1
?COMPILE Card 8-1
MCP Label Card 8-2
$ Option Control Card 8-2
Source Data Card 8-5
Label Equation Card 8-5

APPENDIX A COBOL RESERVED WORDS A-1

List of Illustrations

Figure — Title — Page

2-1 IDENTIFICATION DIVISION Coding 2-4
3-1 ENVIRONMENT DIVISION Coding 3-15
4-1 Coding of Level-Number 4-3
4-2 Coding of Multi-Dimensioned Table 4-5
4-3 Coding of FD and DATA RECORDS 4-10

List of Illustrations (cont)

Figure	Title	Page
4-4	Coding of Condition-Name	4-28
4-5	WORKING-STORAGE SECTION Coding	4-56
5-1	Example of SEARCH Operation Relating To Option 1	5-69
5-2	Coding of PROCEDURE DIVISION	5-88
7-1	Coding Format for a Source Line	7-1
7-2	COBOL Coding Form	7-2
7-3	Sample Coding Showing Continuation of Lines, Special Remarks and Actions	7-4
8-1	Compilation Card Deck	8-1

List of Tables

Table	Title	Page
4-1	Maximum Value of Integers	4-12
4-2	Recording Modes for Peripheral Devices	4-19
4-3	Editing Sign Control Symbol Results	4-39
4-4	Order of Precedence When Using Characters As Symbols	4-44
4-5	Numeric and Alphabetic Items	4-45
4-6	Alphanumeric Items	4-46
4-7	Editing Application of the Picture Clause	4-47
5-1	Combination of Symbols in Arithmetic Expressions	5-7
5-2	Relationship of Conditions, Logical Operators, and Truth Values	5-9
5-3	Combinations of Conditions and Logical Operators	5-10

INTRODUCTION

This manual provides a complete description of COBOL (COMMON BUSINESS ORIENTED LANGUAGE) as implemented for use on this system. This concept of COBOL embraces the adoption of proposed American National Standards Institute (ANSI) COBOL-68.

COBOL's long list of advantages is derived chiefly from its intrinsic quality of permitting the programmer to state the problem solution in English. The programming language reads much like ordinary English prose, and can 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, they obtain maximum documentational advantages of the language described herein.

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

a. Expeditious means of program implementation.

b. Accelerated programmer training and simplified retraining requirements.

c. Reduced conversion costs when changing from a computer of one manufacturer to that of another.

d. Significant ease of program modification.

e. Standardized documentation.

f. Documentation which facilitates non-technical management participation in data processing activities.

g. Efficient object program code.

h. Segmentation capability which sets the maximum allowable program size well in excess of any practical requirement.

i. Due to the incorporation of debugging language statements, a high degree of sophistication in program design is achieved.

j. A comprehensive source program diagnostic capability.

A program written in COBOL, called a source program, is accepted as input by the COBOL Compiler. The compiler verifies that all rules outlined in this manual are satisfied, and translates the source program language into an object program language capable of communicating with the computer and directing it to operate on the desired data. Should source corrections become necessary, appropriate changes can be made and the program recompiled. Thus, the source deck always reflects the object program being operationally executed.

A COBOL source program is always divided into four parts or DIVISIONS in the following order:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

The purpose of the IDENTIFICATION DIVISION is to identify the program and to include an overall description of the program.

The ENVIRONMENT DIVISION consists of two sections. The Configuration Section specifies the equipment being used. The Input-Output Section associates files with the hardware devices that will be used for their operation. This section also furnishes the compiler with information about mass storage parameters.

The DATA DIVISION is used to describe data elements which the object program is to manipulate or create. These data elements may be items within files, records or program work areas, and constants.

The PROCEDURE DIVISION defines the necessary steps which will accomplish the desired task when operating on the data as defined in the DATA DIVISION.

SECTION 1

COBOL LANGUAGE ELEMENTS

GENERAL.

It has been stated that COBOL is a language based on English and that the language is composed of words, statements, sentences, paragraphs, etc. 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.

CHARACTER SET.

The COBOL character set for this system consists of the following 53 characters:

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

CHARACTERS USED FOR WORDS.
The character set for words consists of the following 37 characters:

0 - 9
A - 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 note below)
)	right parenthesis	;	semicolon (see note below)

NOTE

Commas and semicolons may be used between statements, at the programmer's discretion, for enhanced readability of the source program. Use of these characters implies that a following statement is to be included as a portion of an entire statement.

CHARACTERS USED IN EDITING.
The COBOL Compiler accepts the following characters in editing:

$	dollar sign	+	plus
*	asterisk (check protect)	-	minus
,	comma	CR	credit
.	actual decimal point	DB	debit
B	space	Z	zero suppress
0	zero		

CHARACTERS USED IN FORMULAS.
The COBOL Compiler accepts the following characters in arithmetic expressions:

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

CHARACTERS USED IN RELATIONS.
The COBOL Compiler accepts the following characters in conditional relations:

=	equal
<	less than
>	greater than

DEFINITIONS OF WORDS.
A word is created from a combination of not more than 30 characters, selected from the following:

A through Z
0 through 9
- (the hyphen)

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

TYPES OF WORDS.
COBOL (like English) contains types of words. These word types are:

a. Nouns.
b. Verbs.
c. Reserved words.

NOUNS.
Nouns are divided into nine special categories:

a. File-names.
b. Record-names.
c. Data-names.
d. Condition-names.
e. Procedure-names.
f. Literals.
g. Figurative constants.
h. Special register names.
i. Special names.

Since the noun is a word, its length may not exceed 30 characters (exception: literals may not exceed 160 characters). For purposes of readability, a noun may contain a hyphen. However, the hyphen may neither begin nor end the noun (this does not apply to literals).

FILE-NAME. A file-name is a collective name or word assigned to designate a set of data items. The contents of a file are divided into logical records that in turn are made up of any consecutive set of data items.

RECORD-NAME. A record-name is a noun assigned to identify a logical record. A record can be sub-divided into several data items, each of which is 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 COBOL to refer to an element of data, or to a defined data area containing data elements. Each data-name must be composed of at least one alphabetical character.

CONDITION-NAME. A condition-name is a special data-name which is assigned to a specific value within a set of values. For illustrating a condition-name, consider this example. If THIS-YEAR identifies the 12 months of a year, whereas its subordinate data items are defined as JANUARY, FEBRUARY, etc., and the values assigned to each month range from 01 to 12, then it follows that JUNE would have the assigned value of 06. Using the condition-name JUNE, the programmer can utilize it in conditional statements as follows:

```
IF JUNE GO TO . . . .
```

which is logically equivalent to the statement:

```
IF THIS-YEAR IS EQUAL TO 06 GO TO . . . .
```

As a conditional-name, the special data-name itself is called a conditional-variable. The value that it may assume is referred to by condition-names. The condition-name is formatted according to noun rules and may be used only in conditional statements.

PROCEDURE-NAME. A procedure-name is either a paragraph-name or section name, and is formulated according to noun rules. The exception is that a procedure-name may be composed entirely of numeric characters. Two procedure-names are identical only if they both consist of the same character strings. For example: procedure-names 007 and 7 are not equivalent.

LITERALS. A literal is an item of data which contains a value identical to the characters being described. There are three classes of a literal: numeric, non-numeric, and undigit.

Numeric Literal.

A numeric literal is defined as an item composed of characters chosen from the digits 0 through 9, the plus sign (+) or minus sign (-), and the decimal point. The rules for the formation of a numeric literal are:

a. Only one sign character and/or more than one decimal point may be contained in a numeric literal for use with Sterling. Left-most decimal determines the scale.

NOTES

A comma must be substituted for the decimal point if the DECIMAL-POINT IS COMMA option is used (see SPECIAL-NAMES in the ENVIRONMENT DIVISION).

The implied USAGE of numeric literals is COMPUTATIONAL except when used with the verbs DISPLAY or STOP.

b. There must be at least one digit in a numeric literal.

c. The sign of a numeric literal must appear as the left-most character. If no sign is present, the literal is defined as a positive value.

d. The decimal point may appear anywhere within the literal except for the right-most character of a numeric literal. A decimal point within a numeric literal is treated as an implied decimal point. Absence of a decimal point denotes an integer quantity. (An integer is a numeric literal which contains no decimal point.)

e. A numeric literal used for arithmetic manipulations cannot exceed 125 signed digits, otherwise, the maximum is 160 digits. The following are examples of numeric literals.

```
  13247
   .005
 +1.808
 -.0968
7894.54
```

Non-Numeric Literal.

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

A non-numeric literal cannot itself exceed 160 characters. Examples of non-numeric literals are:

Literal on Source Program Level	Literal Stored by Compiler
"ACTUAL SALES FIGURE"	ACTUAL SALES FIGURE
"-1234.567"	-1234.567
"""LIMITATIONS"""	"LIMITATIONS"
"ANNUAL DUES"	ANNUAL DUES
""""	"
"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 non-numeric literals. For example, "-7.7" and -7.7 are not equivalent. The compiler stores the non-numeric literal as -7.7, whereas the numeric literal would be stored as 0077 if the PICTURE were S999V9 DISPLAY with the assumed decimal point located between the two sevens.

Undigit Literals.

Binary 10 through 15 are represented as A through F and must be bounded by @ signs. For example, binary 11 would be literalized by @B@. An undigit literal cannot exceed 160 digits. Refer to section 7 for the correct declaration.

FIGURATIVE CONSTANT. A figurative constant is a particular value that has been assigned a fixed data-name and must never be enclosed in quotation marks except when the word, rather than the value, is desired. The figurative constant names and their meanings are:

ZERO
ZEROS
ZEROES — Represents the value of 0.

ZERO
ZEROS
ZEROES
: Represents the value of 0.

SPACE
SPACES
: Represents one or more spaces (blanks).

HIGH-VALUE
HIGH-VALUES
: Represents the highest internal coding sequence (i.e., 999) value. When HIGH-VALUES are moved to a signed numeric computational field, the sign will not be changed.

LOW-VALUE
LOW-VALUES
: Represents the lowest internal coding sequence (blanks) value. When LOW-VALUES are moved to a signed numeric computational field, the sign will not be changed.

QUOTE
QUOTES
: Represents one or more of the single character " (quotation mark). The word QUOTE or QUOTES does not have the same meaning in COBOL as the symbol ". For example, if "STANDARDS" appears as part of the COBOL source program, the word STANDARDS is stored in the object program. If however, the full "STANDARDS" is desired in a DISPLAY statement, it can be achieved by writing QUOTE "STANDARDS" QUOTE, in which case the object program will print "STANDARDS". The same result can be obtained by writing """STANDARDS""" in the source program. Only the latter method can be used in MOVE statements and conditionals.

ALL
: When followed by a non-numeric literal or a figurative constant, the word ALL represents a series of that literal. For example, if the COBOL 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	Resulting value of ERROR-CODE
ALL "ABC"	7 characters	ABCABCA
ALL "3" or ALL 3	5 characters	33333
ALL "HI-LO"	12 characters	HI-LOHI-LOHI
ALL QUOTE	3 characters	"""
ALL SPACES	9 characters	(nine spaces)

NOTE

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

SPECIAL REGISTER NAME. The Burroughs COBOL Compiler provides four special PROCEDURE DIVISION register names which are:

a. TALLY.
b. TODAYS-DATE (Calendar).
c. DATE (Julian).
d. TIME.

Tally.
The special register TALLY is automatically provided by the COBOL Compiler and has a defined length of five COMPUTATIONAL digits. The primary use of TALLY is in conjunction with the EXAMINE statement, however, TALLY may be used as temporary storage or an accumulative area during the interim when EXAMINE...TALLYING...is not being executed in a program.

Todays-Date (Calendar).
This special register contains the current date and is maintained by the Master Control Program (MCP). Its format is made of three character pairs, each representing the month, day and year. For example, if the current date is Dec. 13th, 1971, the TODAYS-DATE register contains 121371. The function of TODAYS-DATE is to provide the programmer with a means of referring to the current date during program execution. TODAYS-DATE is maintained in COMPUTATIONAL form.

Date (Julian).
This special register contains the current Julian date and is maintained by the MCP. Its format is YYDDD. For example, if the current date were January 1, 1971, the DATE register would contain 71001. The function of DATE is to save programmatic evaluation of TODAYS-DATE when Julian dates are required. DATE is maintained in COMPUTATIONAL form.

Time.
Access to an internal clocking register reflecting the time of day is programmatically available whenever TIME is requested. This register is maintained in milliseconds by the MCP as a 10-digit COMPUTATIONAL field. The contents of the TIME register will be

maintained in hours, minutes, seconds and 60th of seconds when TIME 60 is declared in the OBJECT-COMPUTER paragraph.

SPECIAL-NAMES.
The SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION allows the programmer to assign a significant character for a CURRENCY SIGN, and to declare DECIMAL-POINT as being a COMMA and to provide a means of relating implementor hardware-names to mnemonic-names as desired by the programmer.

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

RESERVED WORDS.
The third type of COBOL word is a reserved word. Reserved words have a specific function in the COBOL language and cannot be used out of context, or for any other purpose than the one for which they were intended. Reserved words are for syntactical purposes and can be divided into three categories:

a. Connectives.
b. Optional words.
c. Key words.

A complete list of reserved words in COBOL used by the compiler is included in appendix A.

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. On the other hand, AND, AND NOT, OR, or NOT are used as logical connectives in conditional statements.

OPTIONAL WORDS. Optional words are included in the COBOL language to improve the readability of the statement formats. These optional words may be included or omitted, as the programmer wishes. For example, IF A IS GREATER THAN B... is equivalent to IF A GREATER B..... Therefore, the inclusion or omission of the words IS and THAN does not influence the logic of the statement.

KEY WORDS. The third kind of reserved words is referred to as being a key word. The category of key words includes the verbs and required words needed to complete the meaning of statements and entries. The category also includes words that have a specific functional meaning. In the example shown in the above paragraph, the words IF and GREATER are key words.

STATEMENT AND SENTENCE FORMATION.

Statements are formed by the completion of the various entry and verb constructs discussed in the later sections of this manual. A statement may be terminated by a period and thus become a sentence. A group of statements, terminated by a period, forms a sentence. An example of a sentence made up of a group of statements would be MOVE A TO B, ADD 01 TO COUNTER WRITE SUMMARY. Note that the word THEN can be used interchangeably with the semi-colon or comma.

PARAGRAPH FORMATION.

One or more sentences may comprise a paragraph. A paragraph begins with a paragraph name and is terminated by the paragraph name of the next paragraph.

SECTION FORMATION.

One or more paragraphs may formulate a section. A section includes all paragraphs between one section name and a following section name, or the end of the source program. Each section must begin with a paragraph-name. The method of referring to procedures within sections and transferring of operational control to these procedures is discussed in the PROCEDURE DIVISION.

NOTATION USED IN VERBS AND ENTRY FORMATS.

The notation conventions that follow enable the reader to interpret the COBOL syntax presented in this manual.

KEY WORDS.

All underlined upper case words are key words and are required when the functions of which they are a part are utilized. Their omission will cause error conditions at compilation time. An example of key words is as follows:

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

The keys words are: IF, NOT, NUMERIC, and ALPHABETIC.

OPTIONAL WORDS.

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

LOWER CASE WORDS.

All lower case words represent generic terms which must be supplied in that format position by the programmer. Integer-1 and integer-2 are generic terms in the following example:

```
    FILE-LIMIT IS integer-1 THRU integer-2
```

BRACES.
When words or phrases are enclosed in braces { }, a choice of one of the entries must be made. In reference to the key words example above, one or the other of the words NUMERIC or ALPHABETIC must be included in the statement.

BRACKETS.
Words and phrases enclosed in brackets [] represent optional portions of a statement. If the programmer wishes to include the optional feature, he may do so by including the entry shown between brackets. Otherwise it may be omitted. In terms of the example above, the word enclosed in brackets is optional. However, if the programmer wishes to distinguish between NUMERIC and ALPHABETIC, he must choose one of the words enclosed in braces.

CONSECUTIVE PERIODS.
The presence of ellipsis (...) within any format indicates that the data immediately preceding the notation may be successively repeated, depending upon the requirements of problem solving.

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. A space after a period is not required, however, such a practice will enhance readability of the source program.

SECTION 2

IDENTIFICATION DIVISION

GENERAL.

The first part or division of the source program is the IDENTIFICATION DIVISION. Its function is to identify the source program and the resultant output of its compilation. In addition, the date the program was written, the date the compilation was accomplished, plus other pertinent information may be included in the IDENTIFICATION DIVISION.

The structure of this division is as follows:

[MONITOR...]

IDENTIFICATION DIVISION.

[PROGRAM-ID. Any COBOL word.] CS4/e S700

[AUTHOR. Any entry.]

[INSTALLATION. Any entry.]

[DATE-WRITTEN. Any entry.]

[DATE-COMPILED. Any entry - replaced by the current date and time as maintained by the MCP.]

[SECURITY. Any entry.]

[REMARKS. Any entry. Continuation lines must be coded in Area B of the coding form.]

SYNTAX RULES.

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

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

b. All paragraph-names within this division must begin under Area A of the coding form.

c. An entry following a paragraph-name cannot contain periods, except that one must be present to denote the end of that entry.

NOTES

When DATE-COMPILED is included, the compiler automatically inserts the time of compilation in the form of HH:MM and the date of compilation in the form of MM/DD/YY.

With the exception of the DATE-COMPILED paragraph, the entire division is copied from the input source program by the compiler and listed on the output listing for documentational purposes only.

MONITOR.
This statement provides a debugging trace of specified data-names and/or paragraph names.

Construct of this statement is:

```
 ┌                                                          ┐
 │ MONITOR [DEPENDING] file-name  ( [data-name] ...    :
 └
    ┌ ┌ ALL               ┐ ┐ )   ┐
    │ │                   │ │ )   │
    └ └ paragraph-name... ┘ ┘ ) . ┘
```

This statement must begin under Area A of the coding form. The parentheses and colon are required as part of the source program statement. MONITOR is active only while the file-name is in OPEN status.

Only one MONITOR statement per program is allowed and must precede the IDENTIFICATION DIVISION header card in the source program.

The file-name must be ASSIGNed to a line printer and is recognized by the compiler as being the output media for the MONITORed data-names.

The data-name(s) may be any name(s) appearing in the DATA DIVISION except for those which require subscripting or indexing.

Whenever a MONITORed elementary data-name is encountered as the receiving field in a MOVE or arithmetic statement, data-name and its current value are listed.

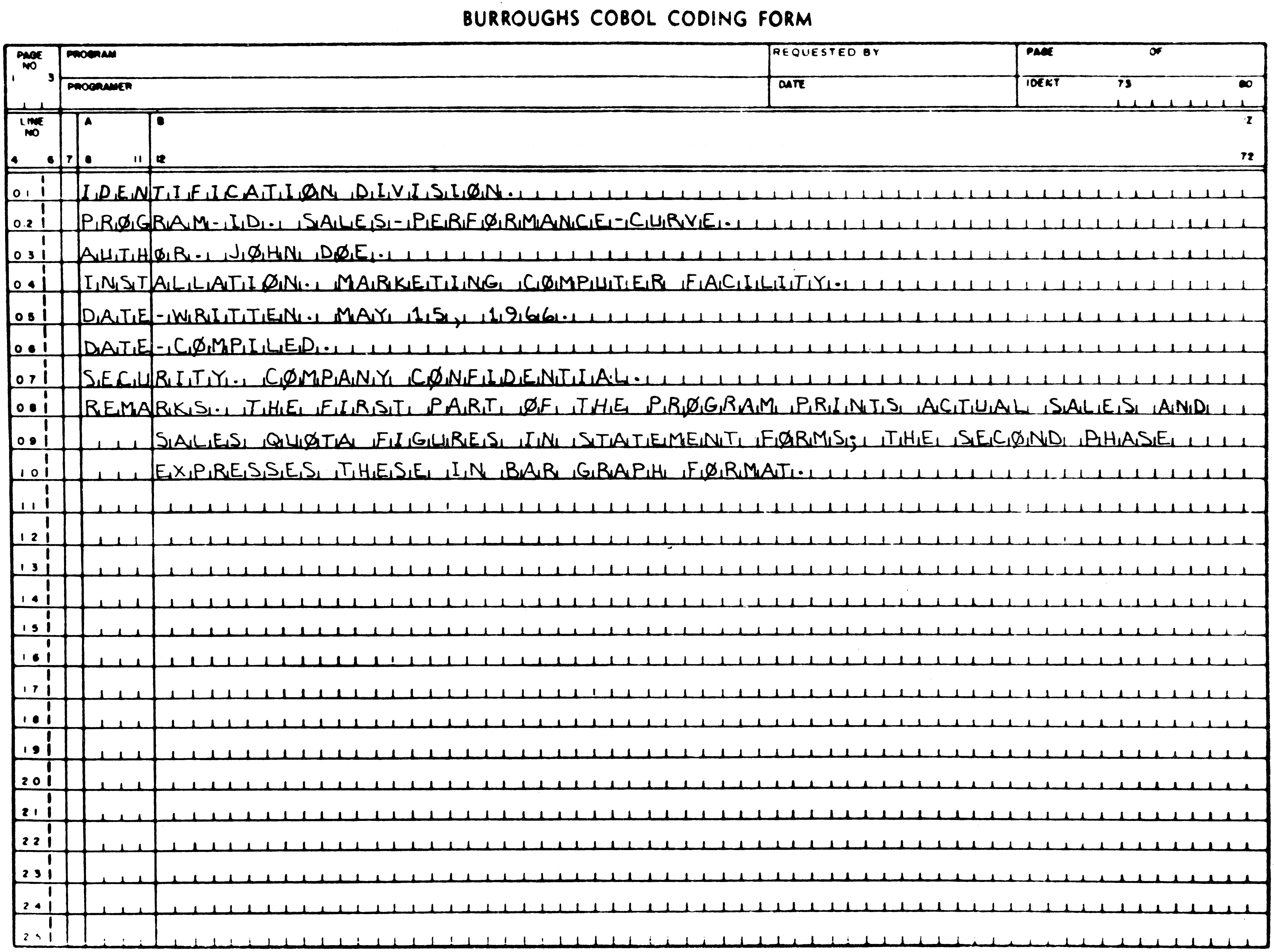
If a group item appears in the data-name-list, it will be MONITORed only when explicitly used as a receiving field.

If the DEPENDING option is present, SW6 will be tested for an ON-OFF condition. Print of MONITORed items will depend upon the setting as being "ON".

All paragraph-names listed will be printed each time they are encountered, along with a total indicating the number of times that a paragraph-name has been passed.

The use of the ALL option, instead of the paragraph-name list, will cause all section and paragraph-names to be MONITORed, thus providing a trace of the programs control path during operation.

CODING THE IDENTIFICATION DIVISION.
Figure 2-1 provides an illustrative example of how the IDENTIFICATION DIVISION may be coded in the source program. Note that continued lines must be indented to the B position of the form, or beyond.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

```
LINE   A    B
NO
4  6 7 8  11 12                                                          72
01     IDENTIFICATIØN DIVISIØN.
02     PRØGRAM-ID. SALES-PERFØRMANCE-CURVE.
03     AUTHØR. JØHN DØE.
04     INSTALLATIØN. MARKETING CØMPUTER FACILITY.
05     DATE-WRITTEN. MAY 15, 1966.
06     DATE-CØMPILED.
07     SECURITY. CØMPANY CØNFIDENTIAL.
08     REMARKS. THE FIRST PART ØF THE PRØGRAM PRINTS ACTUAL SALES AND
09         SALES QUØTA FIGURES IN STATEMENT FØRMS; THE SECØND PHASE
10         EXPRESSES THESE IN BAR GRAPH FØRMAT.
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Figure 2-1. IDENTIFICATION DIVISION Coding

SECTION 3

ENVIRONMENT DIVISION

GENERAL.

The ENVIRONMENT DIVISION is the second division of a COBOL source program. Its function is to specify the computer being used for the program compilation, to specify the computer to be used for object program execution, to associate files with the computer hardware devices, and to 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.

ORGANIZATION.

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

STRUCTURE.

The structure of this division is as follows:

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

SYNTAX RULES.

The following syntax rules must be observed in the formulation of the ENVIRONMENT DIVISION:

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

b. All entries other than the ENVIRONMENT DIVISION source line are optional, but when used they must begin under Area A of the coding form.

Specific definitions for the ENVIRONMENT DIVISION paragraphs are given on the following pages.

CONFIGURATION SECTION.
The CONFIGURATION SECTION contains information concerning the system to be used for program compilation (SOURCE-COMPUTER) and the system to be used for program execution (OBJECT-COMPUTER).

SOURCE-COMPUTER.
The function of this paragraph is to allow documentation of the configuration used to perform the COBOL compilation.

The construct of this paragraph is:

Option 1:

```
[  SOURCE-COMPUTER.   COPY    library-name

   [             { word-1      }    { word-2      }
   [ REPLACING   { data-name-1 } BY { data-name-2 }
   [                                { literal-1   }

           [,    { word-3      }    { word-4      } ]    ]  . ]
           [     { data-name-3 } BY { data-name-4 } ] ...]    ]
           [                        { literal-2   } ]    ]    ]
```

Option 2:

```
[ SOURCE-COMPUTER.   { B-1700    } .  ]
[                    { any entry }    ]
```

This paragraph is for documentation only.

OBJECT-COMPUTER.
The function of this paragraph is to allow a description of the configuration used for the object program.

The construct of this paragraph is as follows:

Option 1:

```
[ OBJECT-COMPUTER.     COPY     library-name
[ [                                                                ]
  [ REPLACING  { word-1      }  BY  { word-2      }
  [            { data-name-1 }      { data-name-2 }
                                    { literal-1   }

   [ ,         { word-3      }  BY  { word-4      } ]   ... ] ]
   [           { data-name-3 }      { data-name-4 } ]
                                    { literal-2   }
```

Option 2:

```
 [ OBJECT-COMPUTER.  [ { B-1700    } ] ]
 [                   [ { any entry } ] ]

  [ MEMORY-SIZE   integer-1 [ { WORDS      } ] ]
  [                         [ { CHARACTERS } ] ]
  [                         [ { MODULES    } ] ]

    [DATA SEGMENT-LIMIT IS integer-2 CHARACTERS] ]
                                                 ]
    [SEGMENT-LIMIT IS priority number]        .  ]
```

If section priority numbers are used in the PROCEDURE DIVISION, they must be positive integers with a value from zero through 99. The SEGMENT-LIMIT clause signifies the limit for non-overlayable program segmentation of sections numbered from 00 through 49. See SEGMENT CLASSIFICATION, PROGRAM SEGMENTS, and PRIORITY NUMBERS on pages 5-15 through 5-19.

The MEMORY-SIZE clause is used for documentation only.

The DATA SEGMENT-LIMIT clause may be used to specify the size of the data segments in the WORKING-STORAGE section. Integer-2 will reflect the number of characters desired in each data segment. When the value of integer-2 is zero, the WORKING-STORAGE section will not be segmented, and will reside in memory as a contiguous block.

If the DATA SEGMENT-LIMIT clause is omitted, no data segmentation will take place.

A record (01 level) that is greater in length than the DATA SEGMENT-LIMIT will be placed in a segment by itself, and will not be split between segments. If DATA SEGMENT-LIMIT has been declared larger than the defined record size, the record will reside in the declared amount of memory, plus the next other record if it will fit into the defined segment.

SPECIAL-NAMES.
The function of this paragraph is to allow the programmer to assign a significant character for all currency signs, to declare decimal points as being commas and to provide a means of relating implementor hardware-names to user specified mnemonic-names.

The construct of this paragraph is:

Option 1:

```
 [ SPECIAL-NAMES.    COPY    library-name

   [ REPLACING    { word-1      }  BY  { word-2      }
                  { data-name-1 }      { data-name-2 }
                                       { literal-1   }

   [ ,    word-3         BY  { word-4      } ]      ... ] ]
          data-name-3        { data-name-4 }
                             { literal-2   }
```

Option 2:

```
 [ SPECIAL-NAMES.    [CURRENCY sign IS literal]

        [implementor-names IS mnemonic-name ...]

        [DECIMAL-POINT IS COMMA]      . ]
```

This paragraph is required if all decimal points are to be interchanged with commas and/or if all currency signs are to be represented by a character other than a dollar sign ($).

This literal is limited to a single character and must not be one of the following:

a. Numeric digits 0 through 9.

b. Alphabetic characters A, B, C, D, J, K, P, R, S, V, X, Z, or blank.

c. Special characters * + - , . ; ( ) ".

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

SPECIAL-NAMES
cont

The implementor-name clause must be one of the allowable COBOL hardware-names are listed on page 3-8. For example:

```
PUNCH IS CARD-PUNCH-EBCDIC
```

The mnemonic named device can be directly referred to in the ASSIGN clause.

The SPECIAL-NAMES paragraph statement ends with a period as a delimiter. Periods between clauses are not allowed.

INPUT-OUTPUT SECTION.

The INPUT-OUTPUT section contains information concerning files to be used by the object program.

FILE-CONTROL.
The function of this paragraph is to name each file, to identify the file medium, and to specify a particular hardware assignment. The paragraph also specifies alternative input-output areas.

The construct of this paragraph has three options which are:

Option 1:

```
[ FILE-CONTROL.    COPY    library-name

  [ REPLACING    { word-1      }  BY  { word-2      }
                 { data-name-1 }      { data-name-2 }
                                      { literal-1   }

    [ , { word-3      }  BY  { word-4      } ] ... ] ]
        { data-name-3 }      { data-name-4 }
                             { literal-2   }
```

Option 2:

```
[ FILE-CONTROL.

        SELECT  [OPTIONAL]  file-name-1 ASSIGN TO hardware-name-1

   [{ NO BACKUP }] [FORM]    [FOR MULTIPLE REEL]
   [{ BACKUP    }]

    [ RESERVE { NO        }  [ ALTERNATE { AREA  } ] ]
    [         { integer-1 }  [           { AREAS } ] ]

 [{ FILE-LIMIT IS   }{ literal-1   }{ THRU    }     { END         }
 [{ FILE-LIMITS ARE }{ data-name-1 }{ THROUGH }     { literal-2   } ...
                                                    { data-name-2 }


    { literal-m   }  { THRU    }   { literal-n   } ] ]
    { data-name-m }  { THROUGH }   { data-name-n } ] ]

  [ ACCESS MODE IS  { RANDOM     } ]   [ACTUAL KEY IS data-name-3]
                    { SEQUENTIAL }
```

WENN RANDOM
fehlt

```
[PROCESSING MODE IS SEQUENTAIL]   .
```

Option 3:

```
[ FILE-CONTROL.
  SELECT sort-file-name ASSIGN TO SORT DISK. ]
```

Option 1 may be used when the systems library contains the LIBRARY name entry. See COPY, section 5.

The files used in a program must be the subject of only one SELECT statement. If it is to be OPENed INPUT-OUTPUT or I-O, it must be present in the MCP Disk Directory.

The word OPTIONAL must be used in the SELECT statement whenever an input file can be omitted during certain operational circumstances.

The ASSIGN clause must be used in order for the MCP to associate the file with a hardware peripheral component. The allowable hardware-name entries are:

B-1712
B-1714
B-1726
B-2500
B-3500
B-4700
B-9350
B-9352
B-9353

DISK (or DISC)
DISK-PACK
DISPLAY-UNIT
IBM-1030
IBM-1050
LISTER
O-L-BANKING
PRINTER
PT-PUNCH

READER
SORTER
SPO
TAPE (7 or 9 channel MCP to assign)
TAPE-7 (7 channel only)
TAPE-9 (9 channel only)
TC-500
TC-700
TOUCH-TONE

CARD96

DC-1000

DCT-2000

PT-READER

PUNCH

TT-28

TWX

The BACKUP option will cause printer output files to be placed on a printer backup tape or disk file for subsequent printing. The BACKUP option will cause punch output files to be placed on punch backup disk files for subsequent punching.

The NO BACKUP option will prevent the file from going to printer backup automatically when the MCP's printer backup option is set "ON" and a Line Printer is not available. This file may be manually assigned to printer backup by the operator with an "OU" or "OUDK" message.

Use of the FORM option with printer or punch files, will cause the program to halt and a MCP message to be printed declaring the need for special forms to be loaded in the Line Printer.

It is recommended that a STOP literal be executed just prior to a STOP RUN if the FORM option is used. This will allow the operator sufficient time to remove the special forms before the printer is released back to the MCP. Without a temporary halt, there is a possibility that another job in the mix may start printing on that same printer.

The MULTIPLE REEL clause is for documentation only. This function is performed by the MCP.

The RESERVE clause allows a variation of the number of input or output physical record buffers to be supplied by the MCP at the time the file is opened. Each ALTERNATE AREA reserved requires additional memory to be utilized, and will be the size of a physical record as defined in the FD statement of the DATA DIVISION for that specific file.

No alternate areas are reserved when the NO option is specified or if the entire option is omitted.

The MCP will keep track of passing record data to or from the buffer and record work area.

The programmer can use the READ or WRITE statements without regard to the buffering action taking place.

The FILE-LIMIT clause is invalid if specified for a sort file description (SD) entry. The FILE-LIMIT clause for input and output files associated with the SORT verb will not be effective when executing the SORT unless there is an INPUT/OUTPUT PROCEDURE declared.

The FILE-LIMIT clause specifies the following:

a. For SEQUENTIAL access, logical records are obtained from, or placed sequentially in, the disk storage file by the implicit progression from segment to segment. The AT END imperative statement of a READ statement is executed when the logical end of the last segment of the file is reached and an attempt is made to READ another record. The INVALID KEY clause of a WRITE statement is executed when the end of the last segment is reached and an attempt is made to WRITE another record. The END option specifies that the compiler is to determine the upper limit of an existing file.

b. For RANDOM access, logical records are obtained from, or placed randomly in, the disk storage file within the specified FILE LIMIT. The contents of ACTUAL KEY not within the specified limit will cause the execution of the INVALID KEY branch in the READ and the WRITE statements.

In the FILE-LIMIT clause, each pair of operands associated with the key word THRU represents a logical segment of a file. The logical beginning of a disk storage file is considered to be that address represented by the first operand of the FILE-LIMIT clause; the logical end is considered to be that address as specified by the last operand of the FILE-LIMIT clause.

In a FILE-LIMIT series, SEQUENTIAL records are accessed in the order in which they are specified. For example:

```
FILE-LIMITS 1 THRU 5, 10 THRU 12, 3 THRU 7
```

This example will result in the sequential access of records 1, 2, 3, 4, 5, 10, 11, 12, 3, 4, 5, 6 and 7 in that order.

For the ACCESS MODE SEQUENTIAL clause, the disk storage records are obtained or placed sequentially. That is, the next logical record is made available from the file on a READ statement execution, or a specific logical record is placed into the file on a WRITE statement execution. The ACCESS MODE SEQUENTIAL clause is assumed if ACCESS MODE RANDOM is not specified.

If the ACCESS MODE RANDOM clause is specified, the ACTUAL KEY entry must be used.

Values of the ACTUAL KEY data-name-3 are controlled by the programmer, including any execution of the USE FOR KEY CONVERSION statement. The value may range from 1 to n, where n equals the number of records in the file or as reflected by the FILE-LIMITS clause. The ACTUAL KEY signifies the relative position of a record within the file and is equated to a data-name at any level which is defined with a PICTURE of 9(8) COMPUTATIONAL. ACTUAL KEY is not used for ACCESS MODE SEQUENTIAL files.

The PROCESSING MODE IS SEQUENTIAL clause is for documentation only.

All integers must be of positive values.

File-name-1 must be unique in the first ten characters if the use of an MCP Label Equation Card is anticipated.

The sort-file-name in Option 2 is the SD level file-name to be used by the SORT verb.

I-O-CONTROL

I-O-CONTROL.
The function of this paragraph is to specify memory area, to be shared by different files during object program execution and the point in time that a rerun procedure is to be established.

The construct of this paragraph is:

Option 1:

```
[ I-O-CONTROL.   COPY   library-name

  [ REPLACING  { word-1      }   BY  { word-2       }
               { data-name-1 }       { data-name-2  }
                                     { literal-1    }

   [ ,  { word-3      }   BY   { word-4      } ] ... ] ]
        { data-name-3 }        { data-name-4 }
                               { literal-2   }
```

Option 2:

```
[ I-O-CONTROL.

  [ RERUN EVERY       { [END OF] REEL.    }   OF file-name-1 ]   ...
                      { integer-1 RECORDS }
  [ SAME   [RECORD] AREA FOR file-name-2 file-name-3

          [file-name-4] ... ]      [ MULTIPLE FILE TAPE "multi-file-id"

          CONTAINS file-name-list [POSITION integer-2 ...]    ... ]  . ]
```

The I-O-CONTROL paragraph name may be omitted from the program if the paragraph does not contain any of the clause entries.

The RERUN clause sets up a communication with the MCP to create control procedures whereby an operational program encountering a malfunction can be restarted at the last RERUN control point instead of restarting from the beginning of the program. Integer-1 records cannot exceed 99999.

The SAME AREA clause in this COBOL compiler is used to assign the same sector and displacement addresses to the record work areas of all files named in the clause. This area will be in the overlayable data section of the program. This capability is due to the VIRTUAL MEMORY concept employed in the design of the system. For example, a given file's File Information Block (FIB), Buffer and Alternate areas will not exist in memory until an OPEN statement in the PROCEDURE DIVISION has been executed. At this time the MCP will allocate sufficient memory outside of the Base and Limit register limits to contain these areas. The file's Record Work area will be called into the overlayable data section of the program whenever it is referenced by the program. When the file is programmatically CLOSED, the memory being used to contain the files FIB, BUFFER and ALTERNATE AREAS will be returned to the MCP.

COBOL restricts the OPENing of files defined as residing in the SAME AREA of memory to one file at a time. This system ignores that logic and the result saves memory over the conventional intent by not using memory to contain FIB record area, buffers, or ALTERNATE AREAs until a file is actually OPENed by the program.

When the RECORD option of the SAME AREA clause is used, only the record area is shared and the associated alternate areas for each file remain independent. In this case, any number of the files sharing the same record area may be OPEN at one time, but only one of the records can be processed at a time.

The use of the RECORD option may decrease the physical size of a program as well as increase the speed of the object program. To illustrate this point, consider file maintenance. If the SAME RECORD AREA is assigned to both the old and new files, a MOVE will be eliminated which transfers each record from the input to the output area. The records do not have to be defined in detail for both files. Definition of a record within one file and the simple inclusion of an 01 level entry for the other file will suffice.

Because these are record areas in fact in the same memory location, one set of data names is sufficient for all processing requirements without requiring qualification.

The MULTIPLE FILE clause specifies that two or more files are resident on one magnetic tape. All files resident on a multi-file tape (that are required in a program) must be represented in the source program by a SELECT statement and a FD entry for each file. The file-name-list entries do not have to be defined in the program sequence in which the files appear on the multi-file tape. However, the MCP will read the label of the next file on tape, check the label against the file request, and, if the next file is not the one requested, it will rewind the multi-file tape and will start searching for it from the beginning of tape.

The "multi-file-id" is the file-name contained in the physical tape label of a magnetic tape containing multi-files, when file-name-list is a series of FD file-names in the program indicated as residing on the multi-file-tape.

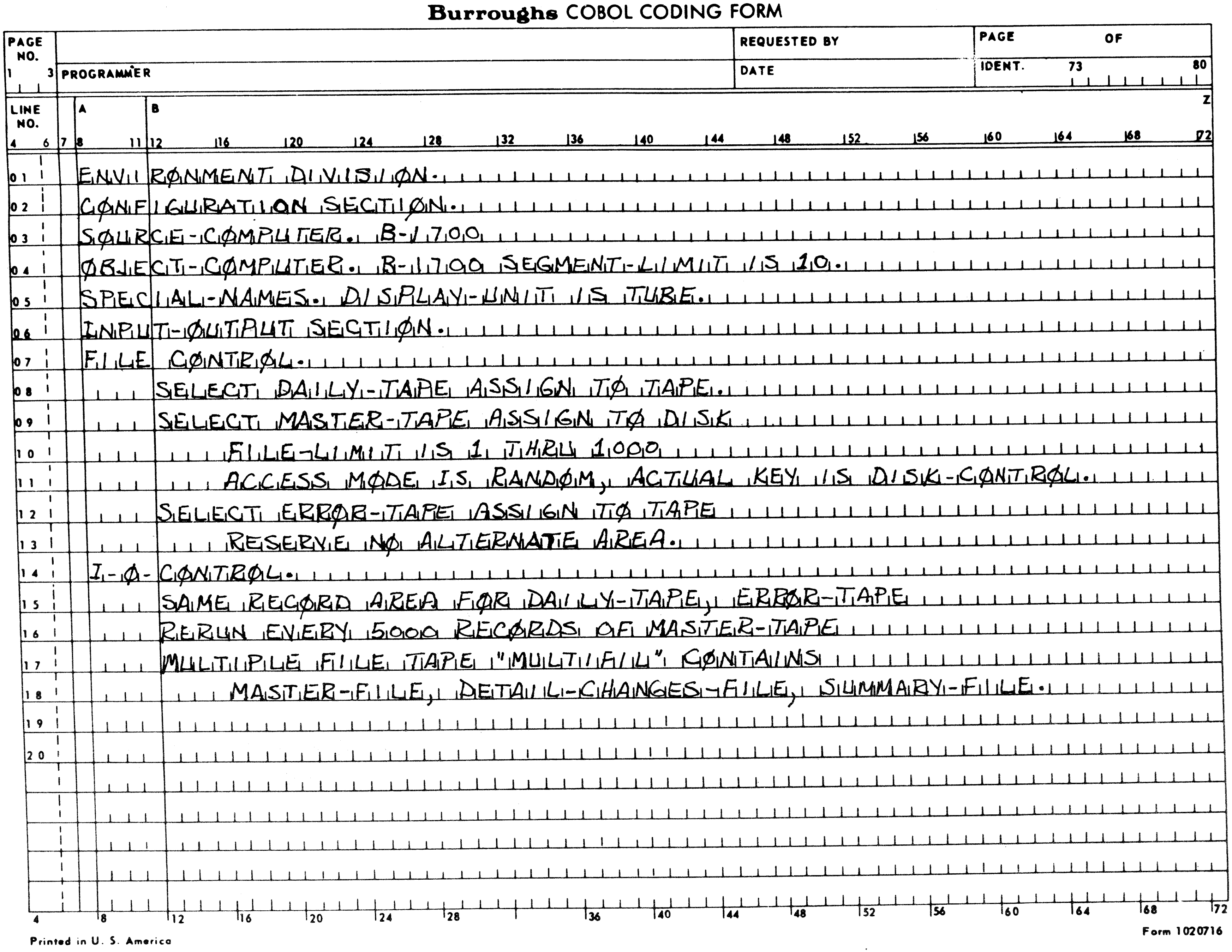
All files named in the MULTIPLE FILE clause will have an implied SAME AREA clause.

Multi-files, or any file contained within the file may be OPTIONAL.

The POSITION clause is for documentation only.

CODING THE ENVIRONMENT DIVISION.

An example of ENVIRONMENT coding is provided in figure 3-1.



Burroughs COBOL CODING FORM

PAGE NO. | PROGRAMMER | REQUESTED BY | DATE | PAGE OF | IDENT. 73 80

LINE NO. | A | B | Z

```
ENVIRØNMENT DIVISIØN.
CØNFIGURATION SECTIØN.
SØURCE-CØMPUTER. B-1700
ØBJECT-CØMPUTER. B-1700 SEGMENT-LIMIT IS 10.
SPECIAL-NAMES. DISPLAY-UNIT IS TUBE.
INPUT-ØUTPUT SECTIØN.
FILE CØNTRØL.
    SELECT DAILY-TAPE ASSIGN TØ TAPE.
    SELECT MASTER-TAPE ASSIGN TØ DISK
        FILE-LIMIT IS 1 THRU 1000
        ACCESS MØDE IS RANDØM, ACTUAL KEY IS DISK-CØNTRØL.
    SELECT ERRØR-TAPE ASSIGN TØ TAPE
        RESERVE NØ ALTERNATE AREA.
I-Ø-CØNTRØL.
    SAME RECØRD AREA FØR DAILY-TAPE, ERRØR-TAPE
    RERUN EVERY 5000 RECØRDS OF MASTER-TAPE
    MULTIPLE FILE TAPE "MULTIFIL" CØNTAINS
        MASTER-FILE, DETAIL-CHANGES-FILE, SUMMARY-FILE.
```

Printed in U. S. America

Form 1020716

Figure 3-1. ENVIRONMENT DIVISION Coding

SECTION 4

DATA DIVISION

GENERAL.

The third part of a COBOL source program is the DATA DIVISION which describes all data that the object program is to accept as input, and to manipulate, create, or produce as output. The data to be processed falls into three categories:

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

b. Data which is developed internally and placed into intermediate storage, or into a specific format for output reporting purposes.

c. Constants which are defined by the programmer.

DATA DIVISION ORGANIZATION.

The DATA DIVISION is subdivided into two sections:

a. The FILE SECTION defines the contents of data files which are to be created or used by an external medium. Each file is defined by a file description, followed by a record description or a series of file-related record descriptions.

b. The WORKING-STORAGE SECTION describes records, constants, and non-contiguous data items which are not part of an external data field, but are developed and processed internally.

DATA DIVISION STRUCTURE.

The general structure of the DATA DIVISION is as follows:

```
DATA DIVISION.
[FILE SECTION.]
[FD file-name-1 . . . .]
   [01 record-name-1 .]
   [02 data-name-1 . . .].
   [02 . . .].
   [03 data-name-2 . . .] .
   [01 record-name-2 .]
[SD file-name-2 .]
[WORKING-STORAGE SECTION.]
   [77 data-name-3 . . . ] .
   [77 data-name-4 . . . ] .
   [01 record-name-3 .]
   [02 data-name-5 . . .] .
   [02 data-name-6 . . .] .
   etc.
   [01 record-name-4 .]
   etc.
```

RECORD DESCRIPTION STRUCTURE.

A Record Description consists of a set of data description entries which describe the elements within a particular record. Each data element consists of a level-number followed by a data-name, followed by a series of independent clauses, as required. A Record Description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate elementary entries.

LEVEL-NUMBER CONCEPT.

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for Condition-Names, non-contiguous constants, Working-Storage items, and the RENAMES clause.

Each record of a file begins with the level-number 01 (which may also be shown as 1). This number is reserved for the record-name only, as the most-inclusive grouping for a record. Less-inclusive groupings are given higher numbers, but not necessarily successively. The numbers can range up to 49. Figure 4-1 illustrates the use of level within a record.

For an item to be elementary, it cannot have subordinate levels. Therefore, the smallest element of a data description is called an elementary item. In figure 4-1, MONTH, DAY, YEAR, MILLING, and FINISHING are elementary items. Since ITEM-NO, LOT-NO, STANDARD-COST, ASSEMBLY, INSPECTION, and WARRANTY-CODE do not have subsidiary clauses, they also represent elementary items.

A level that has further subdivisions is called a group item. In figure 4-1, ITEM-DATE, PRODUCTION-CODE, and MACHINE-SHOP represent items on a group level. A group is defined as being composed of all group and elementary items described under it. A group item ends when a level-number of equal or lower numeric value than the group item itself is encountered. In figure 4-1, group item PRODUCTION-CODE ends with INSPECTION. A group item can only consist of a level-number and a data-name followed by a period. COBOL defines all group items to be alphanumeric and will be byte aligned by the compiler. The FILLER ADDED message will appear where such alignment has taken place. Apart from level-numbers 01 through 49, three additional level-numbers exist in COBOL. These are numbers 66, 77, and 88. They represent level-numbers within RENAMES, WORKING-STORAGE, and Condition-Name entries respectively.

To reiterate, a level-number is the first required element of each record and data description entry. In value it can range from 01 through 49 (1, 2, etc. is also permissible), plus special numbers of 66, 77, and 88. It is important to remember that multiple level 01 entries of a given File Description of the File Section represent implicit redefinition of the same memory area.

QUALIFICATION.

The data-names of the DATA DIVISION need not be unique as long as the parent item of that data-name is unique in itself. Qualification is accomplished by following the data-name to be qualified with either

BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01    01  PRODUCTION-RECORD.                           (record-name)
02        03 ITEM-NØ PICTURE 99999.                    (elementary item)
03        03 LØT-NØ PICTURE 999999.                    (elementary item)
04        03 ITEM-DATE.                                (group item)
05            05 MØNTH PICTURE 99.                     (elementary item)
06            05 DAY   PICTURE 99.                     (elementary item)
07            05 YEAR  PC 99.                          (elementary item)
08        03 STANDARD-CØST  PICTURE 9(5)V99.           (elementary item)
09        03 PRØDUCTIØN-CØDE.                          (grøup item)
10            05 MACHINE-SHØP.                         (grøup item)
11                07 MILLING  PICTURE 999.             (elementary item)
12                07 FINISHING  PICTURE 99.            (elementary item)
13            05 ASSEMBLY  PC 9999.                    (elementary item)
14            05 INSPECTIØN  PICTURE XXXXX.            (elementary item)
15        03 WARRANTY-CØDE ....  .
16
17
18
19
20
21
22
23
24
25
```

Figure 4-1. Coding of Level-Number

IN or OF and the qualifying data-name, record-name or file-name. In the example below, all item descriptions (except the data-name PREFIX) are unique. In order to refer to either PREFIX item, qualification must be used. Otherwise, if reference is made to PREFIX only, the compiler would not know which of the two is desired. Therefore, in order to move the contents of PREFIX into PREFIX of other, the PROCEDURE DIVISION must be coded with one of the following sentences:

a. MOVE PREFIX OF ITEM-NO TO PREFIX IN CODE-NO.
b. MOVE PREFIX OF ITEM-NO TO PREFIX IN MASTER-FILE.
c. MOVE PREFIX OF TRANSACTION-TAPE TO PREFIX IN CODE-NO.
d. MOVE PREFIX OF TRANSACTION-TAPE TO PREFIX IN MASTER-FILE.

Example:

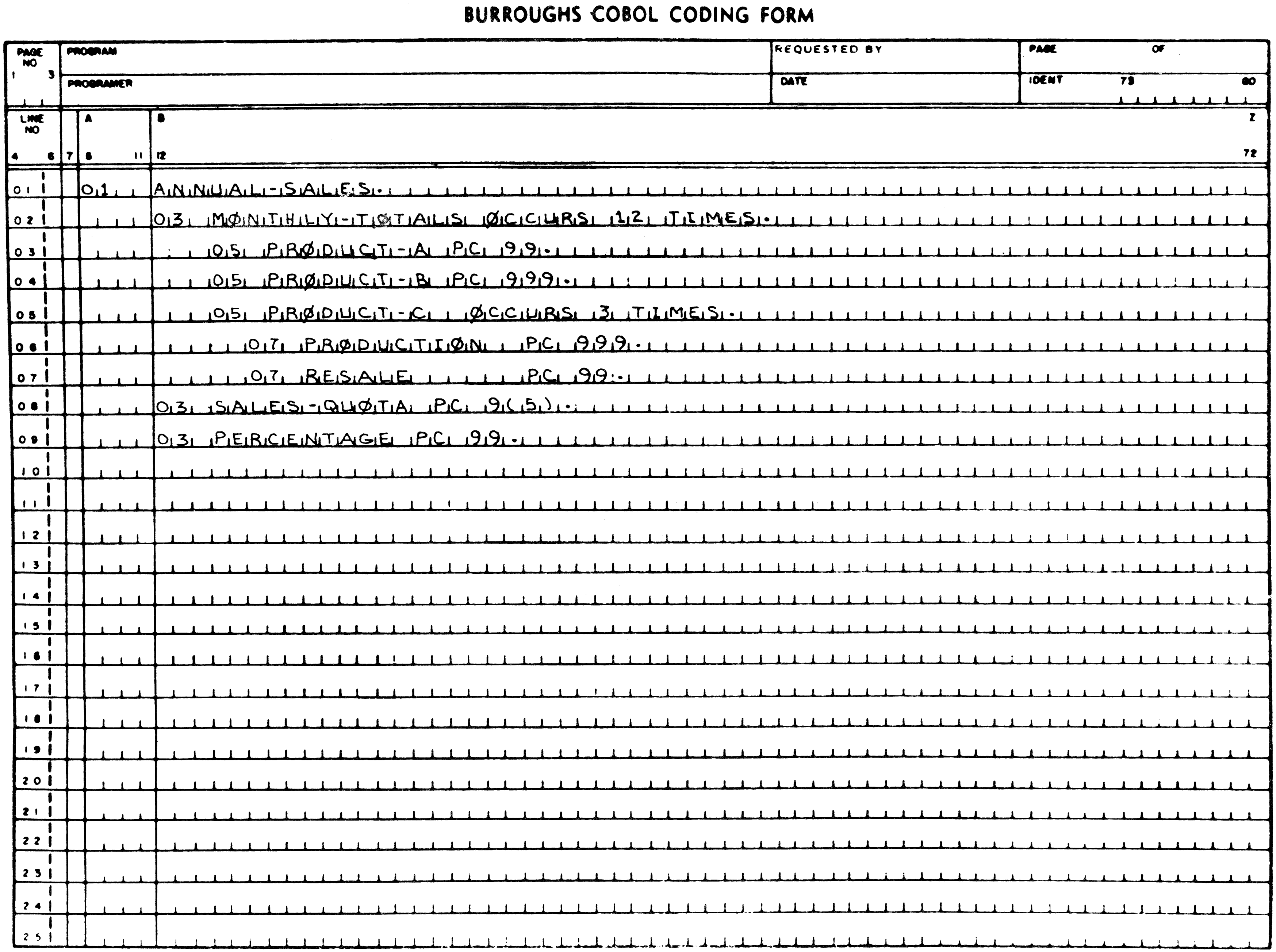
```
01 TRANSACTION-TAPE .....      01 MASTER-FILE .....
   03 ITEM-NO .....               03 CODE-NO .....
      05 PREFIX .....                05 PREFIX ....
      05 CODE ....                   05 SUFFIX ....
   03 QUANTITY .....              03 DESCRIPTION .....
```

TABLES.
Frequently, the need arises to describe data which appears in a table or an array. For example, an annual sales total record might have to be broken down by months. In order to accomplish this, January sales would have to be referred to by a given data-name, February sales by another, etc. By using the OCCURS clause, the same result can be obtained without the need for 12 different data-names. Figure 4-2 illustrates how the OCCURS clause may be used in order to have the compiler build a table of 12 elements, each having a structure like MONTHLY-TOTALS. The first element will be known as 1 of the table, the second as 2, etc. The technique of referring to elements within a table or an array is known as subscripting.

The OCCURS clause may appear at any level except the 01 level which is reserved for record-names. For more detailed information, refer to the OCCURS clause.

The repetition of data elements applies to all subordinate fields. OCCURS may be nested to describe tables of more than one dimension when the OCCURS clause is applied to a subordinate name. The compiler permits tables of up to three dimensions.

SUBSCRIPTING.
When a data-name OCCURS more than once, the particular element desired within the array is referred to by the use of subscripts. The subscripts follow the data-name representing the array in a COBOL statement. A space may separate the data-name and the subscript bounded by parentheses. A subscript can either be a numeric literal or a data-name. A data-name being used as a subscript can not be subscripted. If the value of a subscript is changed in a series (e.g., MOVE A (B) to C (B), B, D (B).) the subscript for D (B) is re-evaluated.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 Z 72

```
01  ANNUAL-SALES.
    03 MØNTHLY-TØTALS ØCCURS 12 TIMES.
          05 PRØDUCT-A PC 99.
          05 PRØDUCT-B PC 999.
          05 PRØDUCT-C  ØCCURS 3 TIMES.
              07 PRØDUCTIØN  PC 999.
              07 RESALE      PC 99.
    03 SALES-QUØTA PC 9(5).
    03 PERCENTAGE PC 99.
```

Figure 4-2. Coding of Multi-Dimensioned Table

In order to reference the first occurrence of MONTHLY-TOTALS of figure 4-2, one could write: ...MONTHLY-TOTALS (data-name), where data-name must contain a 1, or MONTHLY-TOTALS (1).

If a data-name INCREMENTER were used to refer to the desired element in a table (using the terms of the sample illustration) MONTHLY-TOTALS (INCREMENTER) would be written. In this case, the INCREMENTER would have to contain a value that represents the desired element. If a specific RESALE item within a given month is again required, RESALE (INCREMENTER, CODE-X) would have to be programmed. CODE-X is a data-name that can have a value of 1, 2, or 3 depending on what level is required.

Indexing into a table follows much the same logic as subscripting. There is a limit of three indexes per operand (e.g., A(INDEX-1, INDEX-2, INDEX-3). The use of a relative index modifies the index name without actually changing its value.

Example:

```
     A (INDEX-1 + 3, INDEX-2 -4, INDEX-3)
```

Relative indexing is indicated by a + or a - integer following an index-name, and causes the affected index to be incremented or decremented by the number of elements within the table.

At the point in time when a data-name is used for subscripting purposes, any SIGN associated with the data-name will be ignored and the contents of the field will be treated as a positive integer, its value must be greater than zero, but not greater than the value shown in the corresponding OCCURS clause. The generated object code checks the validity of data-name values used for subscripting or indexing. Should the program reference a subscripted data-name or an index-data-name containing a value of zero, or a value above the defined subscript or index range as reflected in the OCCURS clause pertaining to that array, the MCP will DS the program indicating PROGRAM SUBSCRIPT ERROR.

When qualification and subscripting are used simultaneously, the qualifications must be followed by the subscripting.

FILE SECTION.
This section contains descriptions of the files used by the object program.

FILE DESCRIPTION.
The function of this paragraph is to furnish information to the compiler concerning the physical structure, identification, and record names pertaining to a given file.

The construct of this paragraph contains four options:

Option 1:

```
┌
│ FD    file-name   COPY   library-name
└
    REPLACING  { word-1       }  BY  { word-2       }
               { data-name-1  }      { data-name-2  }
                                     { literal-1    }

    ┌                                { word-4      }      ┐   ┐
    │ { word-3      }    BY          { data-name-4 }  ... │ . │
    └ { data-name-3 }                { literal-2   }      ┘   ┘
```

Option 2:

```
   FD file-name-1   ┌ RECORDING MODE IS  { ASCII        } ┐
                    │                    { STANDARD     } │
                    └                    { NON-STANDARD } ┘

 [ FILE CONTAINS integer-1 [BY integer-2] RECORDS ]

 ┌                                               { RECORDS    } ┐
 └ BLOCK CONTAINS [integer-3 TO] integer-4       { CHARACTERS } ┘

 [ RECORD CONTAINS [integer-5 TO] integer-6 CHARACTERS ]

 ┌ LABEL  { RECORD IS    } { OMITTED  } ┐
 └        { RECORDS ARE  } { STANDARD } ┘
```

```
[ { VA    }  OF  { ID             }  IS  { "literal-1"   }
[ { VALUE }      { IDENTIFICATION }      {  data-name-1  }

      [/"literal-2"]  [SAVE-FACTOR IS literal-3] ]

[      { RECORD IS   }                                     ]
[ DATA { RECORDS ARE }   data-name-2  [data-name-3 ...]    ] .
```

Option 3:

```
  SD  sort-file-name COPY library-name

                                        { word-2      }
[  REPLACING  { word-1      }  BY       { data-name-2 }
[             { data-name-1 }           { literal-1   }

                                        { word-4      } ]          ]
               [ { word-3      }  BY    { data-name-4 } ]    ...   ] .
               [ { data-name-3 }        { literal-2   } ]          ]
```

Option 4:

```
  SD  sort-file-name
[ FILE CONTAINS integer-1 [BY integer-2]  RECORDS ]
 [ RECORD CONTAINS [integer-3 TO integer-4 CHARACTERS] ]

[                                             { RECORDS    } ]
[ BLOCK CONTAINS [integer-5 TO] integer-6     { CHARACTERS } ]
[      { RECORD IS   }                                 ]
[ DATA { RECORDS ARE } data-name-1 [data-name-2] ...   ] .
```

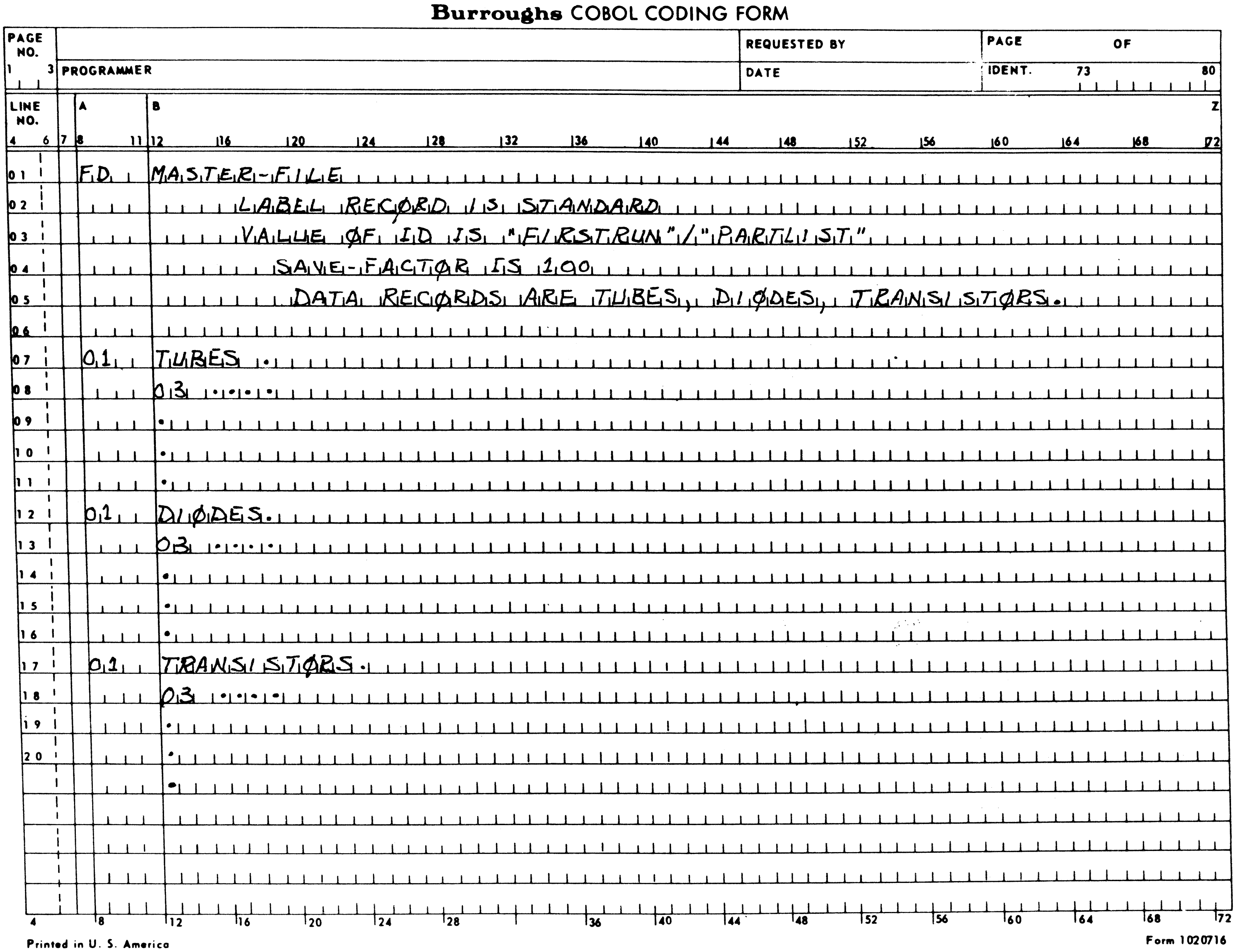
The level indicator, FD and SD identify the beginning of a File Description or a Sort File Description and must precede the file statement. Both entries must commence under Area A of the coding form. Only one period is allowed in the entry and it must follow the last used clause.

Options 1 and 3 can be used when the Systems library contains the library-name entry, otherwise, Option 2 and/or Option 4 must be used.

In many cases, the clauses within the File Description, or Sort File Description sentence are optional. Each clause is discussed in detail.

NOTE

Figure 4-3 illustrates the use of the File Description sentence followed by data record entries. It is further noted that the three 01 levels implicitly redefine the record area and that the DATA RECORDS clause is treated by the compiler as being documentational only and does not cause an explicit redefinition of the area.



Burroughs COBOL CODING FORM

PAGE NO. 1 3	PROGRAMMER	REQUESTED BY DATE	PAGE OF IDENT. 73 80

```
LINE
NO.
4  6 7 8   11 12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
01     FD     MASTER-FILE
02                 LABEL RECØRD IS STANDARD
03                 VALUE ØF ID IS "FIRSTRUN"/"PARTLIST"
04                   SAVE-FACTØR IS 100
05                    DATA RECØRDS ARE TUBES, DIØDES, TRANSISTØRS.
06
07     01     TUBES .
08            03 .....
09            .
10            .
11            .
12     01     DIØDES.
13            03 .....
14            .
15            .
16            .
17     01     TRANSISTØRS.
18            03 .....
19            .
20            .
              .
```

Printed in U. S. America

Form 1020716

Figure 4-3. Coding of FD and DATA RECORDS

BLOCK.
The function of this clause is to specify the size of a physical record (block).

The construct of this clause is:

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

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

This clause is required if the block contains more than one logical record.

When only integer-2 is used, it will represent logically blocked, fixed length, records if its value is other than 1. When the integer-1 TO integer-2 option is used, it will represent the minimum to maximum size of the physical record and indicates the presence of blocked variable-length records. Integer-1 is for documentation purposes only.

The maximum value of the integer used in this clause is shown in table 4-1 and refers to the number of characters in a block.

The word CHARACTERS is an optional word in the BLOCK clause. Whenever the key word RECORDS is not present, the integers represent characters.

For object program efficiency, the use of blocked records is recommended. The physical size of the block should be as large as possible depending on memory availability.

Blocks of records are READ into the input record buffer area by the MCP, and the delivery of each record to the programs record work-area (required by an explicit READ command) is completed.

Blocking or un-blocking of records is of no concern to the programmer.

Table 4-1

Maximum Value of Integers

I/O Medium	Maximum Block Size - Characters
READER	80/96
PUNCH	80/96
TAPE	Limited only by the amount of memory available.
DISK	Limited only by the amount of memory available.
PRINTER	One print line.
PT-READER	Limited only by the amount of memory available.
PT-PUNCH	Limited only by the amount of memory available.

Every explicit WRITE verb causes compiler generated object code to deliver a record to a files output record buffer area, and to accumulate the number of logical records required to create a specified block size before notifying the MCP to write the block. When a file is CLOSEd, the records left in the output buffer area will be written as a short block by the MCP before the file is physically CLOSEd. The coding of record area to buffer is automatic, and is of no concern to the programmer.

The user must specify the actual size of variable-length records in the first four bytes of each record. This four-character indicator is counted in the physical size of each record.

The BLOCK clause is not applicable to the READER, PT-PUNCH, or PT-READER peripherals.

This clause may be omitted for unblocked files.

DATA RECORDS.
The function of this clause is to document the names of the logical record(s) actually contained within the file being described.

The construct of this clause is:

```
[        ( RECORD IS   )                                        ]
[ DATA   ( RECORDS ARE )   data-name-2   [data-name-3...]       ]
```

This statement is only for documentation purposes. The compiler will obtain this information from 01 level record description entries.

FILE CONTAINS

FILE CONTAINS.
The function of this clause is to indicate the number of logical records in a file. This statement is required for disk files, and optional for all other files.

The construct of this clause is:

```
[ FILE  CONTAINS  integer-1  [BY integer-2] RECORDS ]
```

The indicated integers must be positive values.

LABEL.
The function of this clause is to specify the presence or absence of file label information as the first and last record of an input or output file.

The construct for this clause is:

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

If this statement is not specified it is assumed that the file either contains, or has been created with STANDARD ANSI labels.

STANDARD specifies that labels exist for the file or device to which the file is assigned. It also specifies that output labels conform to the standards as implemented.

STANDARD, when specified for disk files, indicates that the 20-character contents of the VALUE or ID clause will be inserted into the disk file header. Should VALUE of ID be omitted, the first 10 characters FD or SD file-name will be inserted into the disk file header.

OMITTED specifies that physical labels do not exist for the specific input file to which the file is ASSIGNed. During object program execution, the operator will be queried by the MCP as to which unit possesses the input data. The operator must reply with "mix-index" UL unit-mnemonic" control message.

The user's portion of the STANDARD label may be of any length. The BURROUGH'S clause specifies that labels exist for the file assigned a magnetic tape input, or will be created for an output magnetic tape file in the BURROUGH'S standard format.

NON-STANDARD indicates that the files physical magnetic tape label is formatted as an EDP installations own standard label which has been appropriately defined in the System Specification Deck at "cold start" time. (See MCP Reference Manual for specifications).

OMITTED specifies that labels are not to be created for the specific output file ASSIGNed.

The BURROUGHs Standard label record serves as both the beginning and ending label record, and is comprised of the following parts:

Positions	Field Description
1	Invalid character for card files and blank for other files

Positions	Field Description
2-8	"LABELbb"
9-18	"Multiple-file-id" or zeros
19	Blank
20-29	"File-identifier"
30	Blank
31-33	Reel number within a magnetic tape file
34-38	Date written (creation date YYDDD)
39-40	Cycle (distinguishing multi-runs of the program)
41-45	Purge-date (YYDDD) at which time the MCP assumes a magnetic tape as "scratch"
46	Sentinel (0 = End-of-File and 1 = End-of-Reel)
47-51	Block count (ending label only)
52-58	Record count (ending label only)
59-63	External magnetic tape library reel number
64-80	Reserved
81	User's portion

The COBOL compiler will obtain the value of "multiple-file-id" from the I-O-CONTROL MULTIPLE FILE TAPE clause.

The COBOL compiler will obtain the value of the "file-identifier from the FD VALUE OF ID IS clause, or if it has been omitted, it will be taken from the first ten characters of the FD-name.

The STANDARD label record serves both the beginning label record and the ending label record. Its format is as follows:

Positions	Field Description
1-3	HDR
4	1
5-14	"multiple-file-id"
15-24	"file-identifier"
25-27	blanks
28	0 (zero)
29-31	nnn (reel number within a magnetic tape file)
32-35	nnn (file sequence number)
36-39	Blanks (generation number optional)
40-41	nn (cycle number-generation version number optional)
42-47	bYYDDD (creation date)
48-53	bYYDDD (purge date)
54	Blank (accessability)
55-60	nnnnnn (block count (end label count))
61-67	nnnnnnn (record count (end label record count))
68-72	nnnnn (physical tape number)
73	B (optional)
74-80	blanks
81-	User's portion

RECORD

RECORD.
The function of this clause is to specify minimum and/or maximum variable record lengths.

The construct of this clause is:

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

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

If integer-1 and integer-2 are indicated, the variable-length record technique is utilized.

If only integer-2 is indicated, the compiler will treat the clause as being documented only. The record size will be determined by the structure of the record description.

If integer-1 and integer-2 are indicated, they refer to the minimum and maximum size of the variable records to be processed. At least one record description must reflect the maximum size record length as specified in the RECORD CONTAINS clause.

The user must specify the actual size of variable-length records in the first four bytes of each record and the record size must contain an even number of characters (MOD 2). The four-character variable-size indicator is counted in the physical size of each record.

This clause is applicable to disk or magnetic tape files sequentially OPENed INPUT or OUTPUT.

RECORDING MODE.
The function of this clause is to specify the recording mode for peripheral devices where a choice can be made.

The construct for this clause is:

```
                                  STANDARD
    RECORDING MODE IS             NON-STANDARD
                                  ASCII
```

STANDARD RECORDING MODE is assumed if this clause is absent from the FD sentence. The MCP automatically checks the parity of input magnetic tapes and will read the tape in the intelligent mode. For this reason, this clause is required only for tapes when the output is to be NON-STANDARD.

The MCP will automatically assign STANDARD RECORDING MODE on 9-channel magnetic tape drives if a SELECT clause indicates TAPE, even though the programmer has designated the unit as being NON-STANDARD.

The recording modes for the peripheral devices are provided in table 4-2.

Table 4-2

Recording Modes for Peripheral Devices

Device	Standard	Non-Standard
TAPE-7	Odd Parity	Even Parity
TAPE-9	Odd Parity	-
DISK	Memory Image	-
READER	Documentational Only	-
PUNCH	EBCDIC or BCD	-
PT-READER	BCL	Binary
PT-PUNCH	BCL	Binary
PRINTER	BCL	-

VALUE-OF-ID

VALUE-OF-ID.
The function of this clause is to define the identification value assigned, or to be assigned, to a file of records and to declare the length of time that a file is to be saved.

The construct of this clause is:

```
[ { VALUE }  OF  { ID             }  IS  { "literal-1" [/ "literal-2"] }
[ { VA    }      { IDENTIFICATION }      { data-name-1                 }

        [SAVE-FACTOR IS literal-3]  ]
```

This clause may be used when label records are present in the file being described. If this clause is not present the compiler will take the VALUE-OF-ID from the first 10 characters of the file-name (FD or SD) and place that ID in the ID entry of the label where the value of literal-1 would normally be found. The file-name must be uniquely constructed so that the MCP will be able to recognize the files.

Example:

```
     FD   SCHEDULE-DISK1     Would create a value of ID as
     FD   SCHEDULE-DISK2     SCHEDULE-D for both files and
                             cause a dup file action by
                             the MCP.
```

To make them unique:

```
     FD   DISKOUTPAY         Would create a VALUE OF ID as
     FD   DISKOUTTAX         DISKOUTPAY and one of DISKOUTTAX
                             thus causing no MCP confusion
                             during object program execution.
```

Each file will have two names each consisting of 10 characters. the first name for a magnetic tape file is a common name of a MULTI-FILE tape and the second name will be the name of a file within the MULTI-FILE. The first name of a magnetic tape file will be taken from the MULTI-FILE clause in the I-O-CONTROL paragraph. The second name will be taken from the value of literal-1, data-name-1, or by default from the FD name.

A disk file can be named in two different ways. It can have one name up to 10 characters long or two names divided by a slash (/) mark each of which can be up to 10 characters long. A file with one name (main directory name) will be placed in the main directory by means of a scramble technique. The address following the name will point to the disk file header. A file with two names adds another level to the directory. The first name is the multi-file or main directory name. The main directory name will be scrambled to a directory with the file-type set to "2". The "2" designates that the address following the name is the address of a sub-directory. The second name or sub-directory name is then placed in this additional directory. The address in the sub-directory now points to the file header of the file. The sub-directory entry will not be scrambled into the directory as is the main directory entry which has the location of the sub-directory. When the MCP finds the sub-directory it must search for the sub-directory file-name.

When data-name-1 is used, it must be defined in the WORKING-STORAGE section of the program and must be described as being alphabetic or alphanumeric.

The VALUE OF ID declared for OUTPUT disk files will cause up to 20 characters of literal-1 and literal-2, or the value of data-name-1 to be inserted into the disk file header. Inversely up to 20 characters of literal-1 and literal-2, or the value of data-name-1 will be checked against the MCP Disk File Directory to obtain the physical disk location of the file when declared as being INPUT or INPUT-OUTPUT disk files.

SAVE-FACTOR is used only for output magnetic tape files. Literal-3 represents the number of days the file is to be saved before it can be manually purged and used for other purposes by the system. Literal-3 is limited to an unsigned positive integer not to exceed three digits in length with values from 001 to 999.

SAVE-FACTOR, when declared for a disk file, is only for documentational purposes due to the fact that files residing on disk should only be purged by mutual consent within an EDP organization and can only be performed as a physical action by the systems operator.

If SAVE-FACTOR isn't specified, tapes are automatically assigned a SAVE-FACTOR of one day to preclude expiration action when the system is being operated during the period just prior to midnight and thereafter.

RECORD DESCRIPTION

RECORD DESCRIPTION.
This portion of a COBOL source program follows the file description entries and serves to completely identify each data element within a record in a given file.

The construct of these entries contain four options which are:

Option 1:

```
   01 data-name-1 COPY library-name

 [ REPLACING  { word-1      }  BY  { word-2      }
              { data-name-2 }      { data-name-3 }
                                   { literal-1   }

   [ { word-3      }  BY  { word-4      }     ... ] ]
     { data-name-4 }      { data-name-5 }
                          { literal-2   }
```

Option 2:

```
 { 01           }  { FILLER      }   [REDEFINES data-name-2]
 { lever-number }  { data-name-1 }

 [ { PC      }
   { PIC     }  IS   (allowable PICTURE characters) ]
   { PICTURE }

 [ { BZ              } ] [ { OC     } { integer-1 TIMES              }
   { BLANK WHEN ZERO }     { OCCURS } { integer-2 TO integer-3 TIMES }

       [DEPENDING ON data-name-3] ]

 [ { ASCENDING  }
   { DESCENDING }   KEY IS data-name-4  [data-name-5] ...  ] ...
```

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

                     DISPLAY
                     CMP
                   ( CMP-1                  )
   [               ( CMP-3                  ) ]
   [               ( COMP                   ) ]  [ ( JS        )         ]
   [ [USAGE IS]   {  COMPUTATIONAL           } ]  [{  JUST      }  RIGHT  ]
   [               ( COMPUTATIONAL-1        ) ]  [ ( JUSTIFIED )         ]
   [               ( COMPUTATIONAL-3        ) ]
                   ( INDEX                  )
                     ASCII

  [ { VA    }                [ { THRU    }              ]
  [ { VALUE } IS literal-1   [ { THROUGH }   literal-2  ]

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

  [ ( SY           )                  ]
  [{  SYNC          } { LEFT  }       ]     .
  [ ( SYNCHRONIZED )  { RIGHT }       ]
```

Option 3:

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

Option 4:

```
    88    condition-name   { VA    }  IS literal-1
                           { VALUE }

    [literal-2...]  [ { THRU    }  literal-n... ] .
                    [ { THROUGH }               ]
```

The optional clauses shown may occur in any order, except if REDEFINES is used it must follow data-name-1.

The record description must be terminated by a period.

Level-numbers in Option 2 may be any number from 1-49.

The clauses PICTURE, BLANK WHEN ZERO, JUSTIFIED, and SYNCHRONIZED must occur on elementary item level only.

Option 1 can be used when the COBOL library contains the record description entry. Otherwise, one of the other options will have to be used.

In many cases, the clauses within the record description sentence are optional. Each clause is discussed in detail.

In Option 4, there is no practical limit to the number of literals in the condition-name series.

The SYNCHRONIZED clause is for documentation only.

BLANK WHEN ZERO.
The function of this clause is to supplement the specification of a PICTURE.

The construct of this clause is:

$$\left[\left\{\begin{array}{l}\underline{\text{BZ}}\\ \underline{\text{BLANK}}\text{ WHEN }\underline{\text{ZERO}}\end{array}\right\}\right]$$

BLANK WHEN ZERO may be abbreviated BZ.

This clause overrides the zero suppress float sign functions in a PICTURE. If the value of a field is all zeros, the BZ clause will cause the field to be edited with spaces. However, it does not override the check protect function (zero suppression with asterisks) in a PICTURE.

The BZ clause can only be used in conjunction with an item on an elementary level.

BLANK WHEN ZERO may be associated only with PICTUREs describing numeric or numeric edited fields.

CONDITION-NAME.
Condition-name is a special name which the user may assign to a given code within a data element. This value may then be referred to by the specified condition-names.

The construct of this clause is:

```
                            { VA    }
88  condition-name          { VALUE }    IS   literal-1

                        [ { THRU    }                ]
[literal-2...]          [ { THROUGH }   literal-n... ]
```

Since the testing of data is a common data processing practice, the use of conditional variables and condition-names supplies a short-hand method which enables the writer to assign meaningful names (condition-names) to particular code values that may appear in a data-field (conditional variable).

When defining condition-names, the following rules must be observed:

a. Each condition-name requires a separate entry with the level-number 88.

b. If reference to a conditional variable requires subscripting, then references to its condition-names also require subscripting.

<u>Examples</u>:

```
02 CONDITION-VARIABLE PC A, OCCURS 10 TIMES.
   88 GIRL  VALUE IS "G".
   88 BOY   VALUE IS "B".
   88 MAN   VALUE IS "M".
   88 WOMAN VALUE IS "W".

IF CONDITION-VARIABLE (SUB) = "G" THEN GO TO
SEE-IF-SHES-PURDY.

IF GIRL (SUB) THEN GO TO SEE-IF-SHES-PURDY.
```

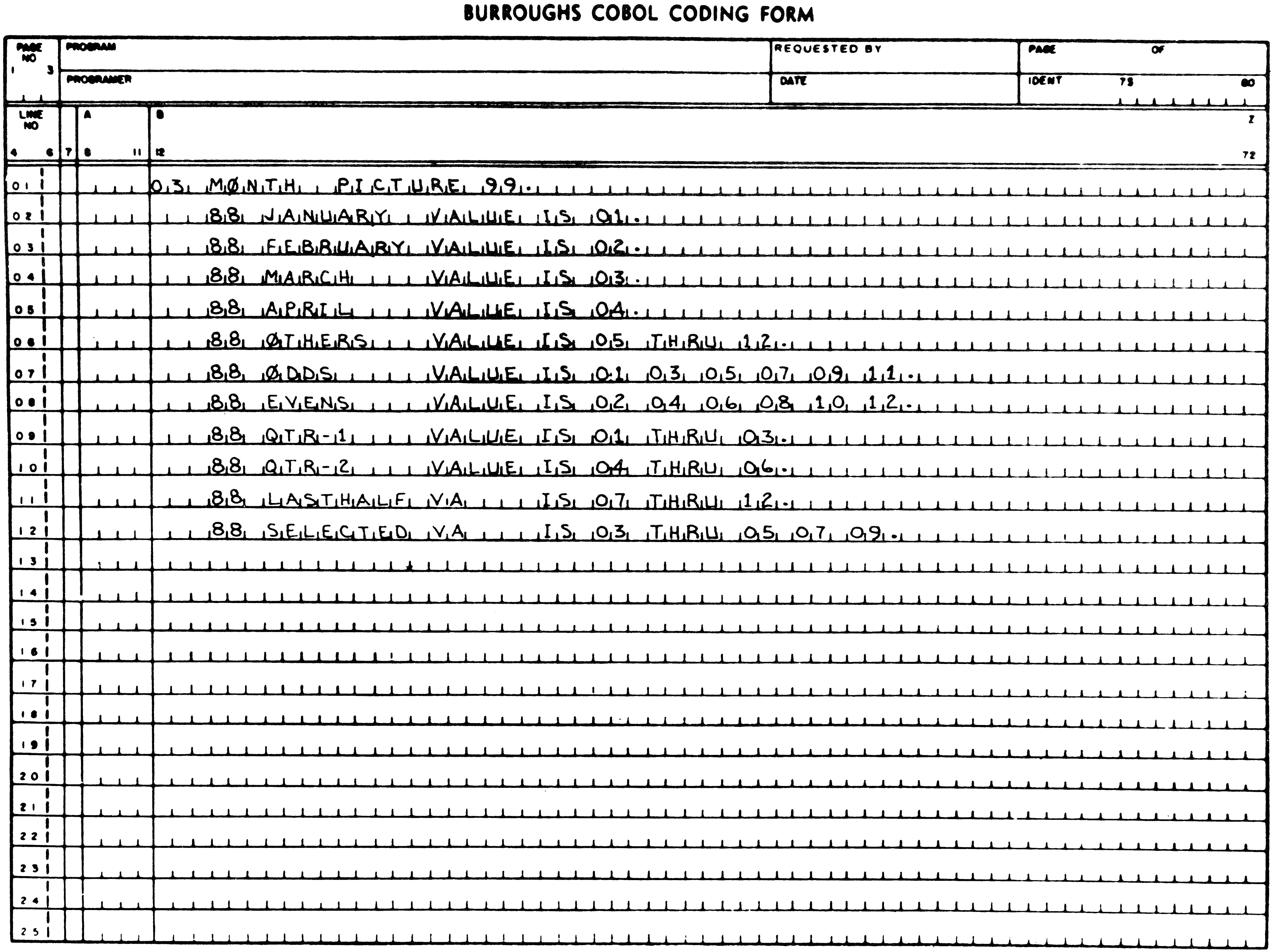
Both of the above examples will generate object code to accomplish the same result.

c. A conditional variable may be used as a qualifier for any of its condition-names.

d. Condition-names can only appear in conditional statements.

e. Condition-names cannot be associated with index-data-names.

f. Figure 4-4 provides an example of the coding of condition-name.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01      03 MØNTH PICTURE 99.
02          88 JANUARY  VALUE IS 01.
03          88 FEBRUARY VALUE IS 02.
04          88 MARCH    VALUE IS 03.
05          88 APRIL    VALUE IS 04.
06          88 ØTHERS   VALUE IS 05 THRU 12.
07          88 ØDDS     VALUE IS 01 03 05 07 09 11.
08          88 EVENS    VALUE IS 02 04 06 08 10 12.
09          88 QTR-1    VALUE IS 01 THRU 03.
10          88 QTR-2    VALUE IS 04 THRU 06.
11          88 LASTHALF VA      IS 07 THRU 12.
12          88 SELECTED VA      IS 03 THRU 05 07 09.
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Figure 4-4. Coding of Condition-Name

DATA-NAME.
The purpose of this mandatory clause is to specify the name of each data element to be used in a program. If a data element requires a definite label, a data-name is assigned. Otherwise, the word FILLER can be used in its place.

The construct of this clause is:

$$\left[\begin{Bmatrix}\underline{\text{FILLER}}\\ \text{data-name-1}\end{Bmatrix}\right]$$

The word FILLER can be used to name a contiguous description area that does not require programmatic reference.

This entry must immediately follow a level-number. FILLER is only applicable to elementary levels.

A data-name need not be unique if it can be made unique through qualification by using data-names on higher levels than itself.

It is not permissible to relationally compare an index-data-name against data-name-1.

JUSTIFIED

JUSTIFIED.
The function of this clause is to specify a non-standard MOVE of alphabetic or alphanumeric data within a receiving data field.

The construct of this clause is:

$$\left[\left\{ \begin{array}{l} \underline{\text{JS}} \\ \underline{\text{JUST}} \\ \underline{\text{JUSTIFIED}} \end{array} \right\} \quad \underline{\text{RIGHT}} \right]$$

The JUSTIFIED clause can be specified only on an elementary item level where the receiving field is described as being alphabetic or alphanumeric. JUSTIFIED can be abbreviated as JS or JUST.

This clause cannot be specified for a receiving field described as being numeric or numeric edited.

When the receiving field is described with the JUSTIFIED clause and the sending field is larger than the receiving field, the left-most characters are truncated.

Example:

```
             SENDING                 RECEIVING

     PC X(7)    A123CDE      PC X(5)    23CDE
```

When the receiving field is described with the JUSTIFIED clause and the sending field is smaller than the receiving field, the data will be positioned right with space fill to the left.

Example:

```
             SENDING                RECEIVING

     PC X(5)    A123C       PC X(7)    A123C
```

JUSTIFIED cannot be associated with an index-data-name.

LEVEL-NUMBER.
The function of this clause is to show the hierarchy of data within a logical record. Its further function is to identify entries for condition-names, non-contiguous constants, working-storage items, and for re-grouping.

The construct of this clause is:

$$\left[\ \text{level-number}\quad \left\{\begin{array}{l}\underline{\text{FILLER}}\\ \text{data-name-1}\end{array}\right\}\right]$$

A level-number is the first required element of each record and data-name description entry.

Level-numbers may be as follows:

a. 01 to 49 - record description and WORKING-STORAGE entries.

b. 66 - RENAMES clause used as a record description or WORKING-STORAGE entry.

c. 77 - applicable to WORKING-STORAGE only as non-contiguous items and must precede all other level-numbers.

d. 88 - condition names clause used as a record description or WORKING-STORAGE entry.

Level-numbers 01 through 49 are used for record or WORKING-STORAGE descriptions. Level number 01 is reserved for the first entry within a record description. Level-number 66 is reserved for RENAMES entries. Level-number 77 is used for miscellaneous elementary items in the WORKING-STORAGE SECTION when these items are unrelated to any record. They are called non-contiguous items since it makes no difference as to the order in which they actually appear. Level-number 88 is used to define the entries relating to condition-names in record descriptions or WORKING-STORAGE entries.

For additional information on level-numbers, see LEVEL-NUMBER CONCEPT on page 4-2.

OCCURS.

The function of this clause is to define a sequence of data-items which possess identical formats, and to define a subscripted item or indices.

The construct of this clause is:

```
  [ { OC     }   { integer-1 TIMES                    }
  [ { OCCURS }   { integer-2 TO integer-3 TIMES       }

      [DEPENDING ON data-name-3] ]

  [ { ASCENDING  }
  [ { DESCENDING }   KEY IS data-name-4 [data-name-5]...  ] ...

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

This clause cannot be used in a record description entry whose level-number is 01, and can only be used with fixed-size items. Any item described with this clause must be subscripted or indexed whenever referenced in a statement other than SEARCH, and all subdivisions of the item must also be subscripted or indexed. Up to three levels of subscripting are acceptable. OCCURS can be abbreviated OC.

If only integer-1 appears, it refers to the exact number of occurrences of the data. Integer-1 must not be zero. Integer-2 TO integer-3 indicates a variable number of occurrences of this item. When integer-2 TO integer-3 is used, the following rules must be observed:

a. Integer-3 must be greater than integer-2 and both must be positive integers.

b. The item must be the last area of a record. No part of a record may follow an item of variable occurrences.

c. Only the first dimension of a table can be defined with this clause. The following definition is not permitted:

```
02 RATE-TABLE  OCCURS 10 TIMES ...
   03 WHOLE-TABLE ...
   03 AGE  OCCURS 4 TO 8 TIMES
```

d. The user must employ his own tests to determine how many occurrences of the item are actually present in the record at any time. The DEPENDING ON option is for documentational purposes only.

Integer-2 TO integer-3 indicates variable-length records and the user must specify the actual size of variable-length records in the first four bytes of each record and the record size must contain an even number of characters (MOD2). The four-character variable size indicator is counted in the physical size of each record.

The following 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; e.g., ITEM-4 (2, 5, 4). A reference to ITEM-3 requires two subscripts; e.g., ITEM-3 (I,J). In the example below there are 50 ITEM-4's.

Example:

```
.         .        .
.         .        .
.         .        .
02 ITEM  OCCURS 2 TIMES ...
   03 ITEM-1 ...
   03 ITEM-2  OCCURS 5 TIMES ...
      04 ITEM-3 ...
      04 ITEM-4  OCCURS 5 TIMES ...
         05 ITEM-5 ...
         05 ITEM-6 ...
         .          .        .
         .          .        .
         .          .        .
```

The following example shows another use of the OCCURS clause. Assume that the user wishes to define a record consisting of five "amount" items, followed by five "tax" items. Instead of describing the record as containing 10 individual data items, it could be described in the following manner:

Example:

```
1 TABLE ...
  2 AMOUNT  OCCURS 5 TIMES ...
  2 TAX  OCCURS 5 TIMES ...
  .        .        .
  .        .        .
  .        .        .
```

The above example would result in memory allocated for five AMOUNT fields and five TAX fields. Any reference to these fields is made by addressing the field by name (AMOUNT or TAX) followed by a subscript denoting the particular occurrence desired.

The ASCENDING/DESCENDING KEY option is for documentation only.

The operands in the INDEXED BY option are *index-names* or *indices*. The operands of an INDEXED BY option must appear in association with an OCCURS clause and are usable only when referencing *that* level of the table. When using three-level indexing, each level must have an INDEXED BY option and in a given indexing operation, only one operand from each option may be used.

Other than their use as an index into an array, an index-name may be referred to only in a SET, SEARCH, PERFORM, or in a relation condition. All index-names must be unique. Index-names have an assumed construction of PC 9(5) COMPUTATIONAL.

Using an index-name associated with one (row of a) table for indexing into another (row of a) table will not cause a syntax error, but will, in most cases, cause incorrect object time results since it is the index-name that contains the information pertinent to the element sizes.

When using an index-name series (e.g., INDEXED BY A, B, C):

a. The indexes should be used only when referencing the associated row.

b. All "assumed" references are to the first index-name in a series. Others in the series are affected only during an explicit reference.

Indexing into a table follows much the same logic as subscripting. There is a limit of three indexes per operand (e.g., A (INDEX-1, INDEX-2, INDEX-3)). The use of a relative index allows modification of the index-name without actually changing the value of the index-name.

Example:

```
     A (INDEX-1 +3, INDEX-2 -4, INDEX-3)
```

Relative indexing is indicated by a + or a - integer following an index-name and causes the affected index to be incremented or decremented by that number of elements within the table.

A data-name whose USAGE is defined to be INDEX is an index-data-name.

Condition-names, PICTURE, VALUE, SYNCHRONIZED or JUSTIFIED *cannot* be associated with an index-data-name.

The COBOL compiler will assign the construction of a PC 9(5) COMPUTATIONAL area for each index-data-name specified.

It is not permissible to relationally compare an index-data-name against a literal or a regular data-name.

PICTURE.
The function of this clause is to describe the size, class, general characteristics, and editing requirements of an elementary item.

The construct of this clause is:

$$\left\{\begin{array}{l}\underline{\text{PC}}\\ \underline{\text{PIC}}\\ \underline{\text{PICTURE}}\end{array}\right\}\quad \text{IS}\quad \text{(character string)}$$

The word PICTURE may be abbreviated as PC or PIC. Character string denotes letters of the alphabet, special characters, and digits which are used in conjunction with one another to describe a data-name. See USAGE for a description of characters and digits.

The maximum number of characters and symbols allowed in the character string used to describe a data-name or FILLER is 30. A character string consists of a certain allowable combination of characters defined as PICTURE descriptors, plus insert characters encompassing the entire character set employed by the systems line printer that have no PICTURE descriptor value or action.

This clause must appear for every elementary item level entry and cannot be used at group levels.

PICTURE cannot be associated with an index-data-name.

A PICTURE of A(5) indicates that the item is a five character (byte) alphabetic field. The integer within parentheses indicates how many times A occurs in order to constitute the desired PICTURE. The PICTURE A(5) can also be represented by AAAAA. The value of the integer within parentheses must always be greater than zero.

Record descriptions do not necessarily have to conform to the physical characteristics of an ASSIGNed hardware-name. The flow of input-output data will terminate at the end of the prescribed PICTURE size. For example:

> READER (can read 80 columns) description can be PICTUREd from 1 through 80.
>
> PUNCH (can punch 80 columns) description can be PICTUREd from 1 through 80.
>
> CARD96 (can read or punch 96 columns) description can be PICTUREd from 1 through 96.
>
> PRINTER (120/132 character lines) description can be PICTUREd from 1 through maximum.

SPO (one character at a time) description can be PICTUREd from 1 to any limit.

There are five categories of data that can be described with a PICTURE clause. These are alphabetic, numeric, alphanumeric, alphanumeric-edited, and numeric-edited.

The symbols used to define the category of an elementary item and their functions are explained as follows:

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

b. The letter B in a character string represents a position into which the space character is to be inserted.

c. The letter J in a character string indicates that the operational data sign is appearing as an over-punch in the least-significant digit position if USAGE IS DISPLAY is associated with the item. However, if USAGE has been indicated as COMPUTATIONAL, J takes on the same function as an S. A J is not counted in the length of a DISPLAY item. Only one operational sign may appear in any one PICTURE and, if specified, the J must appear as the left-most character of the PICTURE. Data elements requiring a J PICTURE descriptor may not be described by a VALUE clause with a signed literal. PICTURE J should be used only in those cases where PICTURE S is not applicable.

NOTE

If J appears within a PICTURE descriptor, it no longer performs as an operational sign but serves to reinitiate zero suppression.

d. The letter K in a character string indicates the presence of an 8-bit (byte) sign appearing in the first character position of a PICTURE descriptor when USAGE is implicitly or explicitly DISPLAY and is counted in the length of the ITEM. If USAGE IS COMPUTATIONAL, the letter K becomes the same as an S. Data elements requiring a K PICTURE descriptor may not be described by a VALUE clause with a signed literal.

e. The letter P in a character string indicates an assumed decimal scaling position and is used for specifying 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 length of the allowable number of characters within a PICTURE description. Scaling position characters are counted in determining the maximum number of digit positions (125) in numeric edited items or numeric items which appear as operands in arithmetic statements. The character P can appear only to the left or right as a continuous string of P's within a PICTURE description. Since it implies an assumed decimal point (to the left of the P's are left-most PICTURE characters and to the right of P's are right-most PICTURE characters), the assumed decimal point symbol V is redundant as either the left-most or right-most character within such a PICTURE description.

f. The letter S in a character string is used to indicate the presence of the standard operational sign in the form of an overpunch in the most-significant digit position of an item if USAGE IS DISPLAY and is not counted in the length of the PICTURE. If USAGE IS CMP, it will denote an operational sign digit in front of the most-significant digit position and is counted in the length of the PICTURE. The S must be written as the first character of the character string of a PICTURE. A signed item may not be more than 125 characters/digits in length. Wherever possible, PICTURE S should be used rather than J or K.

g. The letter V in a character string indicates the location of an assumed decimal point and may only appear once in a character string. It does not represent a character position and therefore is not counted in the length of the item. When the assumed decimal point character V is the right-most character of the PICTURE character string, it is redundant. The maximum number of decimal places is 125.

h. The letter X in a character string indicates an alphanumeric position which can contain any allowable character in the computer's character set.

i. Each letter Z in a character string represents a zero suppress editing action and may only be used to cause the left-most leading numeric character positions to be replaced by a space at object time when the contents of that character position is zero. Each Z is counted as part of the PICTURE length. Zero suppression is terminated with the first non-zero numeric character in the data. Insertion characters are also replaced by spaces while suppression is in effect. Z can also appear to the right of J when the J symbol is used to reinitiate zero suppression. For additional information on zero suppression, see the BLANK WHEN ZERO clause. FILLER entries cannot be defined by the letter Z usage.

j. The number 9 in a character string represents numeric data. If USAGE IS explicitly or implicitly DISPLAY, the data will be operated on as 8-bit (byte) characters. If USAGE IS CMP, it will be operated on as 4-bit digits. Each 9 is counted in the length of the PICTURE.

k. The number 0 (zero) in a character string represents a position into which zero is to be inserted when that item is a receiving field and it is counted in the length of the PICTURE.

l. The special character comma in a character string represents a position into which a comma will have to be inserted. It is counted as part of the PICTURE length. (Also see DECIMAL-POINT IS COMMA in section 3 of this document.) If zero suppression is indicated, a blank character will replace each applicable comma until meaningful data is encountered in the data stream.

m. The special character period in a character string is an editing symbol which represents the decimal point for data alignment purposes. In addition, it represents a character position into which a period will be inserted. It is counted as part of the PICTURE length. If more than one period is indicated in the PICTURE, the left-most period determines the scale of the PICTURE. The PICTURE must not terminate with a period except when it is used to indicate the end of the item clause. For a given program, the function of the period and comma are exchanged if the DECIMAL-POINT IS COMMA clause appears in the SPECIAL-NAMES paragraph. If exchanged, the rules that apply to the use of periods apply to commas and vice versa. (Also see DECIMAL-POINT IS COMMA in section 3 of this document.)

n. The symbols +, -, CR, and DB are used as editing sign control symbols. When used, they 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 PICTURE is counted in the length.

   1) Fixed insertion characters. A single + or - can be used at the extreme left or right of a PICTURE. The CR and DB can be used only at the extreme right end of a PICTURE. The CR and DB symbols represent a two character position and are counted in the length of the item. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE. The currency symbol ($) must be the leftmost character position except that it can be preceded by either a + or - symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character string. Editing sign control symbols (sometimes referred to as report signs) produce the results shown in table 4-3, depending upon the value that the item contains.

Table 4-3

Editing Sign Control Symbol Results

Editing Symbol In Picture Character String	Result	
	Data Item Positive	Data Item Negative
+	+	-
-	Space	-
CR	2 Spaces	CR
DB	2 Spaces	DB

2) Floating Insertion Characters. When used as floating replacement and suppression characters, + and - are written from the extreme left of the PICTURE to represent each leading numeric character into which the sign (+ or -) is to be floated. At least two symbols must be shown to use the subject symbols as floating characters. The floating symbol may not appear to the right of the decimal point unless all replacement positions consist of that symbol. In this case, the field will consist of all spaces when the value is zero. The currency symbol and editing symbols + and - are the insertion characters, and they are mutually exclusive as floating insertion characters in a PICTURE character string.

3) In a PICTURE character string, there are only two ways of representing floating insertion editing. One way is to represent, by the insertion characters, any or all of the leading numeric character positions to the left of the decimal point. The other way is to represent all of the numeric character positions in the PICTURE character string by the insertion characters. If the first method is employed, a single insertion character will be placed into the character position immediately preceding the first non-zero digit in the data represented by the insertion symbol string to the decimal point, whichever is encountered first. If the second method is used, 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. The PICTURE must contain at least

one more floating insertion character than the maximum number of significant digits in the item to be edited.

o. The special character asterisk in a character string represents a leading numeric character position into which an asterisk will be placed when the content of that position is zero and asterisk replacement was not disabled. Asterisk replacement is disabled when the first non-zero character is encountered, or when the decimal point (implicit or explicit) is reached. When the PICTURE character string specifies only asterisks (*), and the value of the item is zero, the entire output item will consist of asterisks and the decimal point, if present. BLANK WHEN ZERO does not override the insertion of asterisks.

p. The special character dollar sign in a character string represents a character position into which a currency symbol is to be inserted. The currency symbol in a character string is represented automatically by a dollar sign ($). If the CURRENCY clause of the SPECIAL-NAMES paragraph is indicated, the dollar sign is replaced by the character specified as a replacement CURRENCY SIGN and is counted in the length of the item.

   1) Fixed insertion character. The currency sign may appear anywhere in the PICTURE.

   2) Floating insertion character. At least two currency signs must appear as the left-most characters in the PICTURE. The currency sign is written to represent each leading numeric character position into which the currency sign may be floated. A single sign is placed in the least-significant suppressed position shown by the currency symbol in the PICTURE. The output item must contain at least one more currency sign character position than the maximum number of significant digits in the source item.

The length of an elementary item, where the length means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols which represent character positions.

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

To define an item as alphabetic, its PICTURE character string can only contain the symbols A and B.

To define an item as numeric, its character string of the PICTURE can only contain the symbols 0, 9, J, K, P, S, and V. Its contents, when represented in standard data format, must be a combination of the numerals 0, 1 through 9. The item may include an operational sign symbol.

To define an item as alphanumeric, its PICTURE character string is restricted to certain combinations of the symbols A, X, and 9.

The item is treated as if the character string contained all X's. The PICTURE character string which contains all A's or all 9's does not define an alphanumeric item.

To define an item as alphanumeric edited, its PICTURE character string is restricted to the following combinations of symbols:

a. The character string must contain at least one X and one B or 0 (zero).

b. Another alternative is that the character string must have at least one A and one B or 0 (zero).

To define an item as numeric edited, its PICTURE character string is restricted to certain combinations of the symbols B, J, K, P, V, Z, 0, 9, comma, period, *, +, -, CR, DB, and the currency sign. The allowable combinations are determined by the order of precedence of symbols and the editing rules. The number of positions which may be represented in the character string is 99.

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

a. Simple insertion.
b. Special insertion.
c. Fixed insertion.
d. Floating insertion.

There are two types of suppression and replacement editing modes:

a. Zero suppression and replacement with spaces.
b. Zero suppression and replacement with asterisks.

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

Simple insertion editing involves the usage of comma, B, and 0 (zero) as the insertion characters. The insertion characters are counted in the length of the item and represent the position in the item into which the character will be inserted.

Special insertion editing character period (.) is used to represent the decimal point for alignment in addition to acting as an insertion character. The insertion character used for the actual decimal point is counted in the length 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 period (.) in the same PICTURE character string is disallowed. If the insertion character is the last symbol in the character string, it must be immediately followed by one of the punctuation characters, semicolon, or period, followed by a space. 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. Any character or digit other than those defined with PICTURE meanings can be used as special insertion characters and will be counted in the size of the PICTURE.

Example:

> 99/99/99 could be a date mask and 999=99=9999
> could represent a social security number mask.

Zero suppression editing of leading zeros in numeric character positions is indicated by the use of the 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 length of the item. If Z is used, the replacement character will be the space and if the asterisk is used, the replacement character will be *.

Zero suppression and replacement is indicated in a PICTURE character string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the character 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 two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression characters. The other way is to represent all of the numeric character positions in the PICTURE character string by suppression characters. 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 non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first. If all numeric positions in the PICTURE character string are represented by suppression symbols, and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire data item will be spaces if the suppression symbol is Z, whereas asterisks will cause the field (except for decimal point) to be replaced with asterisks. Even if the BLANK WHEN ZERO clause is used in conjunction with asterisks, the replacement of character positions containing zeros will be conducted with asterisks. BLANK WHEN ZERO will be ignored if used in the same picture clause with CHECK PROTECT (*).

The symbols +, -, and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character string. At least two floating replacement characters must appear as the left-most characters in the PICTURE.

Table 4-4 shows the order of precedence when using characters as symbols in a character string. For a given function in the left column, a small x in its row indicates that the arguments, used as column headings, are the only ones that may immediately precede the first appearance of the function in a particular string. Arguments appearing in braces ( ) indicate that the symbols are mutually exclusive. The currency symbol is represented by $.

The symbols A, B, V, X, 0, 9, period, and comma can be preceded by any symbols in the PICTURE character string except CR and DB. The V has one other exception and that is it cannot be preceded by an A or X.

> NOTE
>
> When the + or - appears on the right of a character string and the P is also on the right, the sign precedes the P.

To simplify the explanation of allowable character pairs in the character string of a PICTURE, tables 4-5 and 4-6 are provided. These tables have been constructed so that they reflect the use of all allowable symbols, depending upon whether the item is numeric, alphabetic, or alphanumeric. For example, if the item is numeric and the programmer wishes to determine whether the symbol V can follow a 9, then table 4-5 should be used. In the numeric item section of table 4-5, the letter Y (Yes) can be found at the crossing point or horizontal, first symbol, 9 and vertical, second symbol, V. On the other hand, the use of J after 9 is indicated with N (No).

Table 4-4

Order of Precedence
When Using Characters As Symbols

	S	P	$	(+	-)	(ZZ	**	$$	++	--)
S										
P	X			(1)	(1)	X	X	X	X	X
$		X		X	X					
+		X				X	X	X		
-		X				X	X	X		
ZZ		X	X	X	X					
**		X	X	X	X					
$$		X		X	X					
++			X	X						
--		X	X							

Table 4-5

Numeric and Alphabetic Items

			SECOND SYMBOL							
			Numeric Item						Alphabetic Item	
			9	V	S	J	K	P	A	B
FIRST SYMBOL	Numeric Item	9	Y	Y	N	N	N	Y		
		V	Y	N	N	N	N	Y		
		S	Y	Y	N	N	N	Y		
		J	Y	Y	N	N	N	Y		
		K	Y	Y	N	N	N	Y		
		P	Y	Y	N	N	N	Y		
	Alphabetic Item	A							Y	Y
		B							Y	Y

Table 4-6

Alphanumeric Items

			SECOND SYMBOL																	
			Non-Editing				Editing													
			9	X	A	B	J	9	V	,	.	+	-	Z	*	CR	DB	B	O	$
FIRST SYMBOL	NoN Editing	9	Y	Y	Y	Y														
		X	Y	Y	Y	Y														
		A	Y	Y	Y	Y														
		B	Y	Y	Y	Y														
	Editing	9					Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N
		V					Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N
		,					Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N
		.					Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N
		+					Y	Y	Y	Y	Y	Y	N	Y	Y	N	N	N	Y	Y
		-					Y	Y	Y	Y	Y	N	Y	Y	Y	N	N	N	Y	Y
		Z					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N
		*					Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N
		CR					N	N	N	N	N	N	N	N	N	N	N	N	N	N
		DB					N	N	N	N	N	N	N	N	N	N	N	N	N	N
		B					Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N
		O					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N
		$	(BUT NOT FIRST SYMBOL IN PC)				Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
		J					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y

Table 4-7 demonstrates the editing function of the PICTURE clause.

Table 4-7

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	- 123.00+
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
9(7)	0012003	ZZ99JZ9	12 3

REDEFINES

REDEFINES.
The function of this clause is to allow an area of memory to be referred to by more than one data-name with different formats and sizes.

The construct of the REDEFINES clause is:

```
    [level-number data-name-1 REDEFINES data-name-2]
```

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

This clause must not be used in 01 level entries of the FILE SECTION as an implicit REDEFINES is assumed when multiple 01 level entries within a file description are present. The size of the record(s) causing implicit redefinition do not have to be equal to that of the record being redefined. The various sizes of implicitly redefined record descriptions create no restriction as to which description is to be coded first, second, third, etc., in the source program.

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

When the level-number of data-name-2 is other than 01 (REDEFINES can not be used on the 01 level in the FILE SECTION), it must specify a storage area of the same size as specified by data-name-1. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not simply of the data items occupying that area.

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

The data description entry being redefined cannot contain an OCCURS clause, nor can it be subordinate to an entry which contains an OCCURS clause.

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

Data-name-2 need not be qualified.

RENAMES.
The function of this clause is to permit alternative and possibly overlapping, grouping of elementary items.

The construct of this clause is:

```
    66  data-name-1 RENAMES  data-name-2

      [ { THRU    }
        { THROUGH }    data-name-3 ] .
```

All RENAMES entries associated with a given logical record must immediately follow its last data description entry.

Data-name-2 and data-name-3 must be names in the associated logical record and cannot be the same data-name or have the same logical address. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.

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

Data-name-2 must precede data-name-3 in the Record Description, and data-name-3 cannot be subordinate to data-name-2.

One or more RENAMES entries can be written for a logical record.

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).

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.

When data-name-3 is specified, none of the elementary items within the range, including data-name-2 and data-name-3, can be of variable-length.

USAGE.
The function of this clause is to specify the format of a data item in compiler storage.

The construct of this clause is:

```
                         DISPLAY
                         CMP
  [                  {   CMP-1                  }  ]
  [                  {   CMP-3                  }  ]
  [   [USAGE IS]     {   COMP                   }  ]
  [                  {   COMPUTATIONAL          }  ]
  [                  {   COMPUTATIONAL-1        }  ]
  [                  {   COMPUTATIONAL-3        }  ]
                         INDEX
                         ASCII
```

The USAGE clause can be written at any level. If USAGE is written on group level, it applies to each elementary item in that group.

COMPUTATIONAL-1 and CMP-1 are acceptable substitutes for, and are equivalent to, COMPUTATIONAL, COMP, or CMP entries.

A warning message of POSSIBLE CMP GROUP USAGE ERROR will appear whenever the receiving field is a group CMP item. It indicates that the resultant contents during object program execution of the group CMP item may not contain expected results.

> NOTE
> Group moves are performed whenever the sending <u>or</u> receiving field is a group item and both will be treated as alphanumeric (byte) data, regardless of USAGE.

The USAGE of an elementary item cannot contradict the USAGE of a group to which the item belongs.

USAGE is a declaration for the EBCDIC internal representation of the system and is defined as follows:

a. When USAGE IS DISPLAY, the data item consists of 8-bit (byte) characters; two such characters comprise a computer word.

b. When USAGE IS COMPUTATIONAL, the data item consists of 4-bit coded digits.

c. When USAGE IS INDEX, a PICTURE may not be specified. For example, "77 ABC USAGE IS INDEX."

d. When USAGE IS COMPUTATIONAL-3 or CMP-3 it specifies the data item consists of 4-bit coded digits with the low-order digit (LSD) containing the sign.

e. The USAGE IS ASCII clause can only be used for 77 level or 01 level data names in the WORKING-STORAGE SECTION. A FILE with recording mode of ASCII will be ASCII USAGE by default.

The PICTURE of a COMPUTATIONAL item can contain only 9's, the operational sign character S, J, or K, the decimal point character V, one or more P's and the insertion character 0 (zero).

COMPUTATIONAL items may be declared for 9-channel magnetic tape files (TAPE-9), disk file (DISK), Supervisory Printer, paper tape files (PT-READER or PT-PUNCH), or for WORKING-STORAGE SECTION items.

A DISPLAY item is automatically converted to its 4-bit equivalent whenever the receiving area is defined as COMPUTATIONAL except when the receiving area is a group item. A CMP item is automatically converted to its 8-bit equivalent whenever the receiving area is declared DISPLAY except when the sending CMP item is a group item.

In the absence of a USAGE clause, USAGE IS DISPLAY will be assumed.

For the most efficient use of hardware storage and internal record storage areas, records should be devised so as to avoid inter-mixing of odd-length COMPUTATIONAL items with DISPLAY items. This rule is due to the compiler automatically placing the machine addresses of DISPLAY areas to modulo two.

When the USAGE IS ASCII is used it specifies the data item consists of ASCII coded data. A DISPLAY or COMPUTATIONAL item will be automatically converted to its ASCII equivalent whenever the receiving area is defined as ASCII. An ASCII item will be automatically converted to its numeric or EBCDIC equivalent when the receiving field is COMPUTATIONAL or DISPLAY.

VALUE.
The function of this clause is to declare an initial value to WORKING-STORAGE items, or the value associated with a condition-name.

The construct of this clause is:

```
[ { VA    }                  [ { THRU    }             ]
  { VALUE } IS literal-1       { THROUGH }   literal-2 ]

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

Abbreviation VA can be used in lieu of VALUE.

Literals may consist of Figurative Constants; e.g., ZEROS, QUOTES, etc.

Literals may be replaced by the reserved word DATE-COMPILED. If DATE-COMPILED is used in the VALUE clause, the date that the program was compiled will be placed in the data-name in the JULIAN form of YYDDD.

In the FILE SECTION, the VALUE clause is allowed only in condition-name (88 level) entries. VALUE entries in other data descriptions in the FILE SECTION are considered as being documentation only.

In the WORKING-STORAGE SECTION, the entire VALUE caluse may be used with condition-name entries. All levels other than 88 are restricted to the use of literal-1 only.

The VALUE clause must not be stated in a Record Description entry with an OCCURS clause, or in an entry which is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

The VALUE clause must not conflict with other clauses in the data description of an item or in a data description within the hierarchy of the item. The following rules apply:

a. If a category of an item is numeric, all literals in the VALUE clause must be numeric literals; e.g., VA 1, 3 THRU 9, 12, 16 THRU 20, 25 THRU 50, 51, 56.

b. If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be specifically stated non-numeric literals; e.g., VA IS "A", "B", "C", "F", "M", "N", "O", "P", "Q", "Z".

c. All literals in a VALUE clause of an item must have a value which requires no editing to place that value in the item as indicated by the PICTURE clause.

d. The function of any editing clause or editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the length of the item.

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

If this clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric literal (byte characters). The group area is intialized without consideration for the USAGE of the individual elementary items. Subordinate levels within the group cannot contain VALUE clauses.

The VALUE clause must not be specified for a group containing items requiring separate handling due to the USAGE clause.

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

A literal must not contain a sign when the VALUE clause is used with a data-name whose PICTURE specifies a J or K sign position.

In a VALUE clause, there is no practical limit to the number of literals in a series. VALUE cannot be associated with an index-data-name.

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 WORKING-STORAGE and the FILE SECTION is that the former deals with data that is not associated with an input or output file.

ORGANIZATION.
Whereas the FILE SECTION is composed of file description (FD or SD) entries and their associated record description entries, the WORKING-STORAGE SECTION is composed only of record description entries and non-contiguous items. The WORKING-STORAGE SECTION begins with a section-header and a period, followed by item description entries for non-contiguous WORKING-STORAGE items, and then by record description entries for WORKING-STORAGE records, in that order. The format for WORKING-STORAGE SECTION is as follows:

```
[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]
```

NON-CONTIGUOUS WORKING-STORAGE.
Items in WORKING-STORAGE which bear no relationship to one another need not be grouped into records provided they do not need to be further subdivided. Instead, they are classified and defined as non-contiguous items. Each of these items is defined in a separate record description entry which begins with the special level-number 77. The following record description clauses are required in each entry:

a. Level-number.
b. Data-name.
c. PICTURE clause or equivalent.

The OCCURS clause is not meaningful on a 77 level item and will cause an error at compilation time if used. Other record description clauses are optional and can be used to complete the description of the item if necessary.

All level 77 items must appear before any 01 levels in WORKING-STORAGE.

WORKING-STORAGE RECORDS.
Data elements in WORKING-STORAGE which bear a definite 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 record description, including REDEFINES, OCCURS, and COPY. Each WORKING-STORAGE record-name (01 level) must be unique since it cannot be qualified by a file-name. Subordinate data-names need not be unique if they can be made unique by qualification.

INITIAL VALUES.
The initial value of any item in the WORKING-STORAGE SECTION is specified by using the VALUE clause of the record description. If VALUE is not specified, the initial values are set to 4-bit zeros (COMPUTATIONAL).

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 4-5 illustrates the coding of the WORKING-STORAGE SECTION.

BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMMER	DATE	IDENT 73 80

```
LINE NO    A       B                                                        Z
4    6  7  8   11  12                                                       72
01         WØRKING-STØRAGE SECTIØN.
02         77  DISK-CØNTRØL PICTURE 9(8) CØMPUTATIØNAL.
03         77  TØTA-SALES PC 9(11) VALUE ZERØS.
04         77  SALES-QUØTA PC 9(10).
05         01  STATE-TABLE.
06             02 STATES.
07                 03 MICH PC 9999.
08                 03 ØHIØ PC 9(4).
09                 03 PENN PC 9(4).
10             02 STATE-KEY REDEFINES STATES ØCCURS 3 TIMES.
11                 03 STATE-CØDE PC 9.
12                 03 CØUNTY PC 99.
13                 03 CITIES PC 9.
14         01  HDG-LINE.
15             03 FILLER PC A(52) VALUE SPACES.
16             03 FILLER PC A(17) VA "SALES PERFØRMANCE".
17             03 FILLER PC A(51) VA SPACES.
18         01  TRANS-TABLE MØD.
19             03 A ....
20
21
22
23
24
25
```

Figure 4-5. WORKING-STORAGE SECTION Coding

SECTION 5

PROCEDURE DIVISION

GENERAL.

The fourth part of the COBOL source program is the PROCEDURE DIVISION. This division contains the procedures needed to solve a given problem. These procedures are written as sentences which may be combined to form paragraphs, which in turn may be combined to form sections.

RULES OF PROCEDURE FORMATION.

COBOL procedures are expressed in a manner similar (but not identical) to normal English prose. The basic unit of procedure formation is a sentence, or a group of successive sentences. A procedure is a paragraph, or a group of successive paragraphs, or a section, or a group of successive sections within the PROCEDURE DIVISION. The first entry following the PROCEDURE DIVISION header must be DECLARATIVES a section-name or a paragraph-name. If the first entry is a section-name, then it must be followed by a paragraph-name. Sentence structure is not governed by the rules of English grammer, but rather, dictated by the rules and formats outlined in this manual.

STATEMENTS.

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

IMPERATIVE STATEMENTS.

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. A single imperative statement is made up of a verb followed by its operand. A sequence of imperative statements may contain either a GO TO statement or a STOP RUN statement which, if present, must appear as the last imperative statement of the sequence. Some of the imperative verbs are:

ACCEPT
ADD(*1)
ALTER
CLOSE
COMPUTE(*1)
DISPLAY
DIVIDE(*1)
EXAMINE
EXIT
GO
MOVE
MULTIPLY(*1)
OPEN
PERFORM
SEARCH
SEEK
SET
STOP
SUBTRACT(*1)
WRITE(*2)

[1] Without the SIZE ERROR Option.

[2] Without the INVALID KEY Option.

CONDITIONAL STATEMENTS.
A conditional statement specifies that a truth value of a condition is to be determined for subsequent action of the object program.

COMPILER-DIRECTING STATEMENTS.
A compiler-directing statement is one that consists of a compiler directing verb (COPY and NOTE) and its operand(s).

SENTENCES.
There are three types of sentences: imperative sentences, conditional sentences, and compiler-directing sentences. A sentence consists of a sequence of one or more statements, the last of which is terminated by a period.

IMPERATIVE SENTENCES.
An imperative sentence is an imperative statement terminated by a period. An imperative sentence can contain either a GO TO statement or a STOP RUN statement which, if present, must be the last statement in the sentence. Examples would be:

```
     ADD MONTHLY-SALES TO TOTAL-SALES, THEN GO TO PRINT-TOTAL.
```

or

```
     DISPLAY "PGM-END" THEN STOP RUN.
```

CONDITIONAL SENTENCES.
A conditional sentence is a conditional statement which may optionally contain an imperative statement and must always be terminated by a period.

Examples:

```
     IF HEIGHT IS GREATER THAN SIX-FEET-NINE GO TO
     TALL-MEN, ELSE ADD 1 TO PUNIES, GO GET-ANOTHER-
     RECORD.

     IF SALES IS EQUAL TO BOSSES-QUOTA THEN MOVE SALESMAN
     TO HONOR-ROLL OTHERWISE MOVE HIS-NAME TO PINK-SLIP-
     LIST, GO TO NEXT-SENTENCE.
```

If the phrase NEXT-SENTENCE immediately precedes a period, then the phrase may be eliminated and a GO TO NEXT-SENTENCE will be implied.

COMPILER-DIRECTING SENTENCES.
A compiler-directing sentence is a single compiler-directing statement terminated by a period.

Example:

```
     COPY SCANER.
```

SENTENCE PUNCTUATION.

VERB FORMATS.
Punctuation rules for individual verbs are as shown in the verb formats and in section 1 of this manual.

SENTENCE FORMATS.
The following rules apply to the punctuation of sentences:

a. A sentence is terminated by a period.

b. A separator is a word or character used for the purpose of enhancing readability. The use of a separator (other than a space) is optional.

c. The allowable separators are: spaces, the semicolon (;), the comma (,), and the reserved word THEN.

d. Separators may be used in the following places:

   1) Between statements.
   2) In a conditional statement.

      a) Between the condition and statement-1.
      b) Between statement-1 and ELSE.

e. A separator (other than a space) should be followed by at least one space but is not required.

EXECUTION OF IMPERATIVE SENTENCES.

An imperative sentence is executed in its entirety and control is passed to the next applicable procedural sentence.

EXECUTION OF CONDITIONAL SENTENCES.

In the conditional sentence:

$$\underline{\text{IF}}\ \text{condition}\quad \text{statement-1}\ \left\{\begin{array}{l}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{array}\right\}\ \text{statement-2}$$

the condition is an expression which is TRUE or FALSE. If the condition is TRUE, then statement-1 is executed and control is immediately transferred to the next sentence. If the condition is FALSE, statement-2 is executed and control passes to the next sentence.

If statement-1 is conditional, then the conditional statement must be the last (or only) statement comprising statement-1. For example, the conditional sentence would then have the form:

```
IF  condition-1  imperative-statement-1  IF  condition-2

                 { OTHERWISE }                  { OTHERWISE }
    statement-3  { ELSE      }    statement-4   { ELSE      }

    statement-2.
```

If condition-1 is TRUE, imperative-statement-1 is executed. If condition-2 is TRUE, statement-3 is executed and control is transferred to the next sentence. If condition-2 is FALSE, statement-4 is executed and control is transferred to the next sentence. If condition-1 is FALSE, statement-2 is executed and control is transferred to the next sentence. Statement-3 can in turn be either imperative or conditional and, if conditional, can in turn contain conditional statements to an arbitrary depth. In an identical manner, statement-4 can either be imperative or conditional, as can statement-2. The execution of the phrase NEXT SENTENCE causes a transfer of control to the next sentence written in order, except when it appears in the last sentence of a procedure being PERFORMed, in which case control is passed to the return control.

EXECUTION OF COMPILER-DIRECTING SENTENCES.

The compiler-directing sentences direct activities during compilation time. On the other hand, procedural sentences denote action to be taken by the object program. Compiler-directing sentences may result in the inclusion of routines into the object program. They do not directly result in either the transfer or passing of control. The routines themselves, which the compiler-directing sentences may have included in the object program, are subject to the same rules for transfer or passing of control as if those routines had been created from procedural sentences only.

CONTROL RELATIONSHIP BETWEEN PROCEDURES.

In COBOL, 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 7), 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 to a named procedure.

In executing procedures, 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 COBOL. In writing procedures in accordance with the rules of the PROCEDURE DIVISION and the requirements of the coding form (section 7), the source 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. The last paragraph in the PROCEDURE DIVISION is the optional special paragraph-name END-OF-JOB which will be the last card in the source program the compiler will use to generate code for the object program.

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. They may be used in GO, PERFORM and ALTER statements if desired.

SECTIONS.

A section consists of 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, then the DECLARATIVE sentence (i.e., 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 paragraphs following it until another section-name is found. It is not required that a program be broken into sections, but this technique is exceptionally useful in trimming down the physical size of object programs by stating a priority number to declare overlayable program storage (see SEGMENT CLASSIFICATION).

Since paragraph-names and section-names both have the same designated position on the reference format (i.e., position A), section-names, when specified, are written on one line followed by a paragraph name on a subsequent line. When PERFORM is used in a non-DECLARATIVE procedural section to call another section, the same rules apply as when PERFORM is used in a DECLARATIVE section.

DECLARATIVES.

Declaratives are procedures which operate under the control of the input-output system. Declaratives consist of compiler-directing sentences and their 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. There are two statements that are called declarative statements in the COBOL Compiler. These are the USE and the COPY statements. The next source statement following the END DECLARATIVES statement must be a Section or paragraph name.

USE STATEMENT.

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 itself is never executed. Within a USE procedure, there must be no reference to the main body of the PROCEDURE DIVISION. The format 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 verb.

COPY STATEMENT AS A DECLARATIVE.
A COPY declarative is used to incorporate a DECLARATIVE library routine in the source program. That is, a routine which is a USE declarative. The format of the COPY declarative is:

```
          section-name SECTION.  COPY  library-name .
```

Complete rules for writing the format for COPY are stated under the COPY verb.

ARITHMETIC EXPRESSIONS.
An arithmetic expression is an algebraic expression which is defined as:

a. An identifier of a numeric elementary item.

b. A numeric literal.

c. Such identifiers and literals separated by arithmetic operators.

d. Two arithmetic expressions separated by an arithmetic operator.

e. An arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary + or -. The permissible combinations of identifiers, literals, and arithmetic operators are given in table 5-1. Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic operation may be performed.

Table 5-1

Combination of Symbols
in Arithmetic Expressions

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

NOTE

In the above table, the letter P represents a permissable pair of symbols. The character - represents an invalid character pair. Variable represents an identifier or literal.

ARITHMETIC OPERATORS.

There are five arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters which must be preceded by a space and followed by a space.

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

FORMATION AND EVALUATION RULES.

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

Unary + or -
**

* and /
+ and -

The symbols + and -, if used without parenthesizing, may only follow one of the arithmetic operators **, *, /, or appear as the first symbol in a formula. Parentheses have a precedence higher than any of the operators and are used to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in formulas where it is necessary to have some 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. Thus, expressions ordinarily considered to be ambiguous, e.g., A / B * C, A / B / C, and A**B**C are permitted in COBOL. They are interpreted as if they were written (A / B) * C, (A / B) / C, and (A**B) **C, respectively. Without parenthesizing, the following example:

```
A + B / C + D ** E * F - G
```

would be interpreted as:

```
A + (B / C) + ((D ** E) * F) - G
```

with the sequence of operations working from the inner-most parentheses toward the outside, i.e., first exponentiation, then multiplication and division, and finally addition and subtraction.

The way in which operators, variables, and parentheses may be combined in an arithmetic expression is summarized in table 5-1.

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

CONDITIONS.

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

a. Relation condition.

b. Class condition.

c. Condition-name condition.

d. Sign condition.

e. NOT condition.

f. Condition $\left\{\begin{array}{l}\underline{\text{AND}}\\ \underline{\text{OR}}\end{array}\right\}$ condition.

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

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

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

Table 5-2 indicates the relationships between the logical operators and conditions A and B. Table 5-3 indicates the way in which conditions and logical operators may be combined.

RELATION CONDITION.
A relation condition causes comparison of two operands, each of which may be a data-name, a literal, or an arithmetic expression (formula). Comparison of two elementary numeric items is permitted regardless of the format as specified in individual USAGE clauses. However, for all other comparisons the operands must have the same USAGE. Group numeric items are defined to be alphanumeric. It is not permissible to compare an index-data-name to a literal or a data-name.

Table 5-2

Relationship of Conditions,
Logical Operators, and Truth Values

Condition		Condition and Value		
A	B	A AND B	A OR B	NOT A
TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

Table 5-3

Combinations of Conditions
and Logical Operators

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

NOTE
The letter P represents a
permitted pair of symbols.
The character - represents
an invalid character pair.

The general format for a relation condition is as follows:

$$\left\{\begin{array}{l}\text{data-name-1}\\ \text{literal-1}\\ \text{arith. expression-1}\end{array}\right\} \quad \text{relational-operator} \quad \left\{\begin{array}{l}\text{data-name-2}\\ \text{literal-2}\\ \text{arith. expression-2}\end{array}\right\}$$

The first operand, data-name-1, literal-1, or arithmetic expression-1 is called the subject of the condition. The second operand, data-name-2, literal-2, or arithmetic expression-2 is called the object of the condition. The object and the subject may not both be literals.

RELATIONAL OPERATORS.
The relational operators specify the type of comparison to be made in a relation condition. The relational operators must be preceded by a space and followed by a space. Relational operators are:

a. IS [NOT] GREATER THAN.
b. IS [NOT] LESS THAN.
c. IS [NOT] EQUAL TO.
d. IS [NOT] >.
e. IS [NOT] <.
f. IS [NOT] =.

COMPARISON OF OPERANDS.

NON-NUMERIC. For non-numeric (byte) operands, a comparison will result when determination is made that one operand is less than, equal to, or greater than the other with respect to a specified internal collating sequence of characters. The size of an operand is the total number of characters in the operand. Non-numeric operands may be compared only when their USAGE is the same, implicitly or explicitly. There are two cases to consider:

a. If the operands are of equal size, characters in corresponding character positions of the two operands are compared starting from the high-order end through the low-order end. If all pairs of characters compare equally through the last pair, the operands are considered equal when the low-order end is reached. The first pair of unequal characters to be encountered is compared to determine their respective relationship. The operand that contains the character that is positioned higher in the internal collating sequence is considered to be the greater operand.

b. If the operands are of unequal size, the comparison of characters proceeds from high-order to low-order positions until a pair of unequal characters is encountered, or until one of the operands has no more characters to compare. If the end of the shorter operand is reached and the remaining characters in the longer operand are spaces, the two operands are considered to be equal.

NUMERIC. For operands that are numeric, a comparison results in the determination that one of them is less than, equal to, or greater than the other with respect to the algebraic value of the operands. The length of the operands, in terms of number of digits, is not significant. Zero is considered a unique value regardless of the sign. Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparisons.

The signs of signed numeric operands will be compared as to their algebraic value of being plus (highest) or minus (lowest).

EVALUATION RULES.
The evaluation rules for conditions are analogous to those given for arithmetic expressions except that the following hierarchy applies:

a. Arithmetic expressions (formulas).
b. All relational operators.
c. NOT.
d. AND.
e. OR.

SIMPLE CONDITIONS.
Simple conditions, as distinguished from compound conditions, are subdivided into four general families of conditional tests: Relation Tests, Relative Value Tests, Class Tests, and the Conditional Variable Tests. A detailed explanation of each of these can be found under the IF verb discussion.

COMPOUND CONDITIONS.
The most common format of a compound condition is:

$$\text{simple-condition-1} \quad \left\{ \begin{array}{c} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \quad \text{simple-condition-2}$$

$$\left[\left\{ \begin{array}{c} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \ \ldots \qquad \left\{ \begin{array}{c} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \quad \text{simple-condition-n} \right]$$

Simple conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators AND, OR, and NOT are shown in table 5-2 where A and b represent simple conditions. Thus, if A is TRUE and B is FALSE, then the expression A AND B is FALSE, while the expression A OR B is TRUE.

The following are illustrations of compound conditions:

a. AGE IS LESS THAN MAX-AGE AND AGE IS GREATER THAN 20.

b. AGE IS GREATER THAN 24 OR MARRIED.

c. STOCK-ON-HAND IS LESS THAN DEMAND OR STK-SUPPLY IS GREATER THAN DEMAND + INVENTORY.

d. 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.

e. STK-ACCT IS GREATER THAN 72 AND (STK-NUMBER IS LESS THAN 100 OR STK-NUMBER EQUAL TO 76920).

Note that it is not necessary to use the same logical connective throughout. The rules for determining the logical (i.e., truth) value of a compound condition are as follows:

a. If AND's are the only logical connectives used, then the compound condition is TRUE if, and only if, each of the simple conditions is TRUE.

b. If OR's are the only logical connectives used, then the compound condition is TRUE if, and only if, one or more of the simple conditions is TRUE.

c. If both logical connectives are used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right.

Parentheses may be used to indicate grouping as specified in the examples below. Parentheses must always be paired the same as in algebra, i.e., the expressions within the parentheses will be evaluated first. In the event that nested parenthetical expressions are employed, the innermost expressions within parentheses are handled first. Examples of using parentheses to indicate grouping are:

a. To evaluate C1 and (C2 OR NOT (C3 OR C4)), use the first part of rule c above and successively reduce this by substituting 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

   This can be evaluated by table 5-2.

b. To evaluate C1 OR C2 AND C3, use the second part of rule C and reduce this to C1 OR (C2 AND C3), which can now be reduced as in example a.

c. To evaluate C1 AND C2 OR NOT C3 AND C4, group first by AND from left to right, resulting in:

    (C1 AND C2) OR (NOT C3 AND C4)

   which can now be evaluated as in example a.

d. To evaluate C1 AND C2 AND C3 OR C4 OR C5 AND C6 AND C7 OR C8, group from the left by AND to produce:

    ((C1 AND C2) AND C3) OR C4 OR ((C5 AND C6)
    AND C7) OR C8

   which can now be evaluated as in example a.

e. The following is using a condition-name as part of the statement.

    IF CURRENT-MONTH AND DAY = 15 OR 30... would be treated as:

    IF (CURRENT-MONTH AND DAY = 15) OR 30... the actual test desired is:

    IF CURRENT-MONTH AND (DAY = 15 OR 30)...

   The required result is that CURRENT-MONTH be true as well as DAY containing either 15 or 30.

   Without the parentheses as shown, the conditions are:

1) DAY = 30 or
2) CURRENT-MONTH is true AND DAY = 15.

ABBREVIATED COMPOUND CONDITIONS.
Any relation condition other than the first that appears in a compound conditional statement may be abbreviated as follows:

a. The subject or the subject and relational operator, may be omitted. In these cases, the effect of the abbreviated relation condition is the same as if the omitted parts had been taken from the nearest preceding complete relation condition within the same condition. That is, the first relation is a condition and must be complete.

b. If, in a consecutive sequence of relation conditions (separated by logical operators) the subjects are identical, the relational operators are identical and the logical connectors are identical, the sequence may be abbreviated as follows:

   1) Abbreviation 1 - when identical subjects are omitted in a consecutive sequence of relation conditions. An example of abbreviation 1 would be:

      IF A = B AND = C.

      This is equivalent to IF A = B and A = C.

   2) Abbreviation 2 - when identical subjects and relational operators are omitted in a consecutive sequence of relation conditions. An example of Abbreviation 2 is:

      IF A = B AND C.

      This is equivalent to IF A = B AND A = C.

c. As indicated in the previous paragraphs, compound conditions can be abbreviated by having implied subjects, or implied subjects and relational operators, providing the first simple condition is a full relation. The missing term is obtained from the last stated relation in the sentence. The following examples further illustrate the abbreviated compound conditions:

   1) IF A = B OR C is equivalent to IF A = B OR A = C.

   2) IF A < B OR = C OR D is equivalent to IF A < B OR A = C OR A = D.

SEGMENTATION.

COBOL segmentation is a facility that provides a means by which communication with the compiler, to specify object program overlay requirements, can be accomplished. COBOL 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.

PROGRAM SEGMENTS.

Although it is not mandatory, the PROCEDURE DIVISION for a source program may be written as a consecutive group of sections, each of which are operations that are designed to collectively perform a particular function. Each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

The object program is composed of two types of segments: a fixed segment and overlayable segments.

a. The fixed segment is the main program segment and is never overlaid by any other part of the program.

b. An overlayable segment is a segment which, although logically treated as if it were always in memory, can be overlaid, if necessary, by another segment to optimize memory utilization. However, such a segment, if called for by the program, is always made available in its "initial" state when the segment priority-number is 50 or greater. When the segment priority-number is 49 or less the segment will be made available in its "initial" state except for ALTERed switches which are always set to their last used state.

Also, depending on availability of memory, the number of permanent segments in the fixed and overlayable portions can be varied by changing the SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph.

SEGMENT CLASSIFICATION.

Sections which are to be segmented are classified using a system of priority numbers and the following criteria:

a. Logic requirements - sections with priority numbers from 00 thru 49 in a program may reside in the fixed segment depending on the value specified in SEGMENT-LIMIT. Sections containing a priority number lower than that specified in SEGMENT-LIMIT, regardless of their physical location in the program, will be assigned to the fixed segment; all other sections will be assigned as overlayable segments. Fall-through control from one SECTION to another SECTION is accomplished in their order of appearance in the source program.

b. Relationship to other sections - sections coded within the SEGMENT-LIMIT range will become the fixed segment and can communicate freely with each other. Those coded outside the stated SEGMENT-LIMIT range fall into the overlayable category and can also communicate from one to the other.

The compiler will create one non-overlayable (fixed) program area which will include all sections with priority numbers below the value specified in SEGMENT-LIMIT. The overlayable sections will be called into memory as needed by the program. When memory is available more than one overlayable section will be in memory at the same time. This will reduce the number of disk accesses which in turn will cause the program to have a shorter run time.

PRIORITY NUMBERS.
Section overlay classifications are accomplished by means of a system of priority numbers. The priority number is included in the section header. The general format of a section header is as follows:

section-name SECTION priority-number.

The priority number must be an integer ranging in value from 00 through 99 (also 0, 1, 2, etc., are permissible priority numbers). If the priority number is omitted from the section header, the priority number is assumed to be 0. Segments with priority numbers ranging from 0 up to, but not including, the value specified in the SEGMENT-LIMIT clause (or 50 if no SEGMENT-LIMIT clause has been specified) are considered as being located in the fixed (non-overlayable) portion of the object program. Segments with priority numbers equal to or higher than, the value specified in SEGMENT-LIMIT, but not exceeding 99, are independent segments (overlayable) and fully ALTERable; however, segments with priority numbers greater than 49 will be made available in their "'nitial" state each time they are referenced. A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. Sections in DECLARATIVES are assumed to be 00 and must not contain priority numbers in their section headers. Priority numbers may be stated in any sequence and need not be in direct sequence. The fixed segment does not end when the first priority number equal to or greater than SEGMENT-LIMIT is encountered.

All segments, regardless of their physical location in the source program, whose priority number is less than that which is specified in SEGMENT-LIMIT will be "gathered" into a single non-overlayable segment. All other segments equal to, or greater than that which is specified in SEGMENT-LIMIT will be "gathered" into overlayable segments according to equal priority numbers regardless of their physical location in the source program.

The use of the "gathering" technique will allow programmers to create tailored segments which will reduce disk access times. For example:

Program A: SEGMENT-LIMIT equals 17.

Non-Gathered

Segment	Description	Size in Digits
00-16	Main body of the program	20,000
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	20,000
17	Used frequently	1,000
18	Used infrequently	5,000
19	Used infrequently	4.000
20	Used at EOJ	500
17	Used frequently (was segment 21)	2,000
19	Used at BOJ (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	20,000
17	Used frequently	3,500
18	Used infrequently	5,000
19	Used infrequently	5,000
20	Used infrequently	5,000

"Fall through" will be performed in the sequence as outlined in the above Non-Gathered example and not as they appear in the Results of the Gathering example above, therefore preserving the logical integrity of the original program.

The COBOL interpreter will automatically check to see if an overlay being called for by an object program is already present in the object programs overlayable memory storage area. If it is present, no disk access is required and the program is interrupted. If it is not present, the COBOL interpreter interrupts the program and will access the disk for the desired overlayable portion of the program. The COBOL interpreter uses overlay segments directly from the program library where the object program was compiled to and is called in as an overlay in its <u>initial generated code each and every time it is required</u> by the operating program. Although the initial code is retrieved each time, the latest addresses of ALTERed exits are still applicable and are in force by the use of an automatic ALTER table for segments with a priority number of 49 or less.

<u>INTERNAL PROGRAM SWITCHES.</u>
Every compiled object program contains eight programmatic switches provided automatically. Switches SW1 through SW8 are composed of one unsigned digit in length and are located in memory locations (base relative) 0 through 7.

These switches can be referred to in the PROCEDURE DIVISION by the use of the reserved words SW1, SW2...SW8. Each individual switch setting can be changed during operation by a MOVE, ADD, SUBTRACT, etc.. For example:

```
MOVE 0 TO SW1.
ADD 1 TO SW2.
SUBTRACT 1 FROM SW3.
```

Note that SW6 has an affect on the MONITORING DEPENDING....requirement if the statement is present.

The switch memory locations are reserved and operate exactly like the reserved TALLY locations.

<u>VERBS.</u>
Some of the verbs available for use with the COBOL Compiler are categorized below. Although the word IF is not a verb in the English language, it is utilized as such in the COBOL language. Its occurrence is a vital feature in the PROCEDURE DIVISION.

a. Arithmetic:
   ADD
   SUBTRACT
   MULTIPLY
   DIVIDE
   COMPUTE

b. Compiler directing declaratives:

NOTE
USE

c. Compiler directing:
    COPY

d. Data manipulations:
    MOVE
    EXAMINE
    SORT

e. Ending:
    STOP

f. Input-output:
    WRITE
    READ
    OPEN
    CLOSE
    ACCEPT
    DISPLAY
    SEEK

g. Logical Control:
    IF

h. Procedure Branching:
    GO
    ALTER
    PERFORM
    EXIT
    ZIP

i. Source-level Debugging:
    TRACE

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 function of this verb is to permit the entry of low-volume data from the console typewriter.

The construct of this verb is:

```
   [                                 { SPO           } ]
   [ ACCEPT  data-name     FROM      { mnemonic-name } ]
```

This statement causes the operating object program to halt and wait for appropriate data to be entered on the SUPERVISORY PRINTER (SPO). The SPO entry will replace the contents of memory specified by the data-name. The systems operator answers an ACCEPT halt by keying in the following message:

mix-index AXdata-required

If a blank appears between the AX and data-required, the blank character will be included in the data-stream.

The number of characters ACCEPTed must correspond to the size of the receiving data-name.

If mnemonic-name is used, it must appear in the SPECIAL-NAMES paragraph equated to the hardware-name SPO.

The receiving data-name may be a group level entry and cannot be subscripted.

Because of the inefficiency of entering data through the keyboard, this technique of data transmission should be solely restricted to low-volume input data.

The maximum number of characters per ACCEPT statement is unlimited.

ACCEPT's of greater than 60 characters must be entered thru the SPO in exact groups of 60 characters, except for the last group, which can be of any size up to 60.

ADD.
The function of this verb is to add two or more numeric data items and adjust the value of the receiving field(s) accordingly.

The construct of this verb has three options.

Option 1:

```
          { literal-1    }  [ { literal-2    }           ]
    ADD   { data-name-1  }  [ { data-name-2  }    ...    ]

    TO  data-name-3 [ROUNDED] [ data-name-n [ROUNDED] ... ]

    [ON SIZE ERROR any statement]
```

Option 2:

```
          { literal-1    }  { literal-2    } [ { literal-3    }       ]
    ADD   { data-name-1  }  { data-name-2  } [ { data-name-3  }   ... ]

    GIVING  data-name-n [ROUNDED]

    [ON SIZE ERROR any statement]
```

Option 3:

```
          { CORR          }
    ADD   { CORRESPONDING }   data-name-1 TO data-name-2

    [ROUNDED]   [ON SIZE ERROR any statement]
```

With Option 1, the value(s) of the operand(s) preceding the word TO will be added together and the sum will be added to the existing value(s) of operand(s) following the word TO. A resumation does not occur if the value of one of the data-names changes in the process. For example:

ADD A TO B,A,C.

In Option 2, the sum of the operands preceding the word GIVING will be inserted as a replacement value of data-name following the word GIVING.

In Options 1 and 2, the data-names must refer to elementary numeric items only, except that data-names appearing only to the right of the word GIVING may refer to data-names which contain editing symbols.

An ADD statement must have at least two operands.

Editing items can only be used as the receiving field with the GIVING format. Operational signs and implied decimal points are not considered as editing symbols.

The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data item that follows the word GIVING, aligned on their decimal points, must not contain more than 125 digits/characters.

The internal format of operands referred to in an ADD statement may differ among each other. Any necessary format transformation and decimal point alignment is automatically supplied throughout the calculation.

Each literal must be a numeric literal.

If, after point alignment with the receiving data item, the calculated result would extend to the right of the receiving data item (i.e., a data-name whose value is to be set equal to the sum), truncation will occur. Truncation is always in accordance with the size associated with the resultant data-name. When the ROUNDED option is specified, it causes the resultant data-name to have its absolute value increased by 1 whenever the most-significant digit of the truncated portion is greater than or equal to five.

Whenever the magnitude of the calculated result exceeds the largest magnitude that can be contained in a resultant data-name, a size error condition arises. In the event of a size error condition, one of two possibilities will occur, depending on whether or not the ON SIZE ERROR option has been specified. The testing for the size error condition occurs only when the ON SIZE ERROR option has been specified.

a. In the event that ON SIZE ERROR is not specified and size error conditions arise, the value of the resultant data-name is unpredictable.

b. If the ON SIZE ERROR option has been specified and size error conditions arise, then the value of the resultant data-name will not be altered. After determining that there is a size error condition, the "any imperative-statement" associated with the ON SIZE ERROR option will be executed.

If Option 3 is used, multiple operations are performed. The operations are executed by pairing identical data-names of numeric elementary items subordinate in hierarchy to data-name-1 and data-name-2. Data-names match if they, and all their possible qualifiers up to, but not including data-name-1 and data-name-2, are the same. All general rules pertaining to the ADD verb apply to each individual ADD operation. For instance, if the size of matched data-names does not correspond in that the decimal point is out of alignment or the sizes differ, the decimal point alignment or truncation takes place according to the rules previously discussed.

In the process of pairing identical data-names, any data-name with the REDEFINES clause is ignored. Similarly, data-names which are subordinate to the subordinate data-names with the REDEFINES clause are ignored.

> NOTE
>
> This restriction does not preclude data-name-1 or data-name-2 themselves from having REDEFINES clauses or from being subordinate to data-names with REDEFINES clauses.

If the CORR or CORRESPONDING option is used, no item in the group referred to can contain an OCCURS clause.

If, in Option 3, either data-name-1 or data-name-2 is a group item which contains RENAMES entries, the entries are not considered in the matching of names.

In Option 3, data-name-1 and data-name-2 must not have a level number of 66, 77, or 88.

If corresponding data-names are not elementary numeric items the ADD operation will be ignored.

In Option 3, CORR is an acceptable substitute for CORRESPONDING.

ALTER

ALTER.
The function of this verb is to modify a predetermined sequence of operations by changing the operand of a labeled GO TO paragraph.

The construct of this verb is:

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

Procedure-name-1, procedure-name-3, ... are names of paragraphs, each of which contains a single sentence consisting of only a GO TO statement as defined under Option 1 of the GO TO verb. Procedure-name-2, procedure-name-4, ... are not subject to the same restrictions and they may be either paragraph names or section names.

When control passes to procedure-name-1, control is immediately passed to procedure-name-2 rather than to the procedure-name referred to by the GO TO statement in procedure-name-1. Procedure-name-1 is therefore a "gate" which remains set until again referenced by another ALTER statement.

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority.

All other uses of the ALTER statement are valid and are performed even if the GO TO which the ALTER refers to is in an overlayable section, as long as the section priority number is less than 50.

CLOSE.
The function of this verb is to communicate to the MCP that the designated file-name being operated on or created is programmatically completed, and also to fulfill the stated action requirements.

The construct of this verb is:

```
                                    ┌              ⎧ LOCK      ⎫ ┐
                                    │              ⎪ PURGE     ⎪ │
   CLOSE  file-name-1  [REEL]       │  WITH        ⎨ RELEASE   ⎬ │
                                    │              ⎪ NO REWIND ⎪ │
                                    └              ⎩ REMOVE    ⎭ ┘

     [file-name-2...]
```

File-names must not be those defined as being SORT files. A file must have been OPENed previously before a CLOSE statement can be executed for the file. File space in memory will not be allocated until the file has been OPENed. When a file is programmatically CLOSEd, the memory allocated for that file will be returned to the MCP. A unit which remains assigned to the program after the file on that unit has been CLOSEd, will be reflected in the I/O assignment table in the MCP.

The above statement applies to the following categories of input and output files.

a. Files whose input and output media involve print files, card files, etc.

b. Files which are contained entirely on one reel of magnetic tape, and are the only files on that reel.

c. Files which may be contained on more than one physical reel of magnetic tape. Furthermore, the number of reels might possibly be higher than the number of physical tape units provided on the system.

d. Disk files.

To show the effects of the CLOSE options, each type of file will be discussed separately.

a. Card Input.

   1) CLOSE - releases the input memory areas, but does not release the reader.

   2) CLOSE WITH NO REWIND - same as CLOSE.

   3) CLOSE WITH RELEASE - releases the input memory areas and returns the reader to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

6) CLOSE WITH REMOVE - same as CLOSE.

b. Card Output.

1) CLOSE - punches the trailer label (if any) releases the output memory areas, but does not release the punch.

2) CLOSE WITH NO REWIND - same as CLOSE.

3) CLOSE WITH RELEASE - releases the output memory areas and returns the punch to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

6) CLOSE WITH REMOVE - same as CLOSE.

c. Magnetic Tape Input.

1) CLOSE - checks the trailer label (if any) rewinds the tape and releases the input memory areas. The unit remains assigned to the program.

2) CLOSE WITH NO REWIND - same as CLOSE except the tape is not rewound.

3) CLOSE WITH LOCK - releases the input memory areas, checks the trailer label (if any) rewinds the tape, and the MCP marks the unit not ready.

4) CLOSE WITH RELEASE - releases the memory input areas, checks the trailer label (if any), rewinds the tape, and returns the unit to the MCP.

5) CLOSE WITH PURGE - releases the input memory areas, checks the trailer label (if any), rewinds the tape, and if a write ring is in the reel, over-writes the label, making the tape a scratch tape which becomes a candidate for use by the MCP. The unit is returned to the MCP.

6) CLOSE WITH REMOVE - same as CLOSE.

d. Magnetic Tape Output.

1) CLOSE - releases the output memory areas, writes the trailer label (if any), and rewinds the tape. The unit remains assigned to the program.

2) CLOSE WITH NO REWIND - releases the output memory areas, writes the trailer label (if any). The tape remains positioned beyond the trailer label (or tape mark if there is no trailer label). The unit remains assigned to the program.

3) CLOSE WITH LOCK - releases the output memory areas, writes the trailer label (if any), rewinds the tape, and the MCP marks the unit not ready.

4) CLOSE WITH RELEASE - releases the output memory areas, writes the trailer label (if any), rewinds the tape, and returns the unit to the MCP.

5) CLOSE WITH PURGE - releases the output memory areas, writes the trailer label (if any), rewinds the tape, returns the unit to the MCP, and the MCP over-writes the label making it a scratch tape, which makes it a a candidate for use by the MCP.

6) CLOSE WITH REMOVE - same as CLOSE.

e. Printer Output.

1) CLOSE - prints the trailer labe (if any), releases the output memory areas but does not release the printer.

2) CLOSE WITH NO REWIND - same as CLOSE.

3) CLOSE WITH RELEASE - releases the output memory areas and returns the printer to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

6) CLOSE WITH REMOVE - same as CLOSE.

f. Disk Files. The actions taken on files ASSIGNED to DISK will be discussed in terms of old files and new files. An old file is one that already exists on disk and appears in the MCP Disk Directory. A new file is one created by the program and does not appear in the Directory. A new file may only be referenced by the program which creates it.

1) CLOSE - releases the input/output memory areas.

a) For an old file, the file is left in the Directory and is available to other programs.

b) For a new file, the file is not entered in the directory, however, it remains on the disk and may be OPENed again by this program.

2) CLOSE WITH NO REWIND - not permitted on disk files.

3) CLOSE WITH RELEASE - releases the input/output memory areas.

   a) For an old file, the file is left in the directory and is available to other programs.

   b. For a new file, the file is not entered in in the directory and the memory and disk areas are returned to the MCP for use by other programs.

4) CLOSE WITH LOCK - releases the input/output memory areas.

   a) For an old file, the file remains in the Directory and is made available.

   b) A new file is entered in the Directory. Subsequent action is identical to an old file.

5) CLOSE WITH PURGE - releases the input/output memory areas.

   a) An old file is immediately removed from the disk and deleted from the Directory.

   b) A new file will be immediately removed from the disk.

6) CLOSE WITH REMOVE - releases the input/output memory areas. This option will cause the MCP to REMOVE a file from the disk directory that has the same file-id as the file being closed. This action will take place prior to entering the closing files file-ID in the disk directory. Use of this option will eliminate the DUPLICATE FILE condition and reduce operator intervention. If the REMOVE option is not used, the "RM" SPO input message will accomplish the same results.

If a file has been specified as being OPTIONAL, the standard END-OF-FILE processing is not permitted whenever the file is not present.

If a CLOSE statement without the REEL option has been executed for a file, a READ, WRITE, or SEEK statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

The CLOSE REEL option signifies that the file-name being CLOSEd is a multi-reel magnetic tape input/output file. The reel will be CLOSEd at the time of encountering the CLOSE REEL statement and an automatic OPEN of the next sequential reel of the multi-reel file will be performed by the MCP.

COMPUTE

COMPUTE.
The function of this verb is to assign to a data item the value of a numeric data item, literal, or arithmetic expression.

The construct of this verb is:

```
                                           ( data-name-2          )
    COMPUTE  data-name-1  [ROUNDED]  =    { numeric-literal        }
                                           ( arithmetic expression )

      [ON SIZE ERROR any statement]
```

The literal must be numeric literal.

Data-name-2 must refer to an elementary numeric item. Data-name-1 may describe a data item which contains editing symbols.

The arithmetic expression option permits the use of any meaningful combination of data-names, numeric literals, arithmetic operators, and parenthesization, as required.

All rules regarding ON SIZE ERROR, ROUNDED options, truncation and editing are the same as for ADD.

If numeric-literal exponents are used, the results are accurate up to 18 digits in length or to as many decimal places.

COPY.
The function of this verb is to allow library routines contained on a source language library file to be incorporated into the program.

The construct of this verb contains two options which are:

<u>Option 1</u>:

```
    COPY  library-name .
```

<u>Option 2</u>:

```
    COPY  library-name

  [                                        ( word-2         )
  [  REPLACING   ( word-1        )   BY    { data-name-2    }
  [              ( data-name-1   )         ( literal-1      )

                                           ( word-4         )
               [ ( word-3        )   BY    { data-name-4    }   ... ] . ]
               [ ( data-name-3   )         ( literal-2      )
```

The COPY statement may refer only to one library entry in the library for every time it is used. Library-name is the value placed in a library entry bounded by quotes or a procedure-name type word. The library entry bounded by quotes cannot contain more than 20 characters, separated by a slash (/).

If the library-name is a procedure-name type word and is numeric, it must be separated from the period (if present) by a space.

The library file is inserted in the source program immediately after the COPY statement at compilation time. The result is the same as if the library data were actually a part of the source program.

Library data can encompass an entire procedure which may be any number of statements, paragraphs, or entire source program divisions or parts thereof.

Library files may not contain COPY statements.

No statement may appear to the right of the COPY statement on the same source card.

COPY during the PROCEDURE or ENVIRONMENT divisions must follow a SECTION or paragraph-name and <u>all</u> information contained in the library file is included and can be fully referenced.

On a COPY during the DATA DIVISION, the FD file-name, or the level 01 data-name preceding the COPY is saved and the relative constructs from the library file are discarded. For example, the statement

```
FD MASTER-INPUT COPY "MASTER".
```

will cause the library file titled MASTER to be inserted into the source program immediately following the COPY statement. The source program must refer to the FD file-name as MASTER-INPUT and not as MASTER. The library FD file-name will appear on the output listing, but cannot be referenced in the source program.

Library files copied from the library are flagged on the output listing by an L preceding the sequence number.

In Option 2, a word is defined as being any COBOL word that is not a COBOL Reserved Word. For example, the following statement reflects non-reserved COBOL words AAA,BBB and 1234, where AAA and BBB are data-names and 1234 is a COBOL word:

```
MULTIPLY AAA BY BBB, THEN GO TO 1234.
```

If the COPY REPLACING option is specified, each word-1 or data-name-1 stipulated will be replaced BY the word-2 or data-name-2 entries specified in the option. Data-names may not be subscripted, indexed or qualified.

Use of the COPY REPLACING option requires that the "library-name" COBOL source image file be present, on disk, prior to compiling the source program containing the COPY REPLACING option. The use of this option will not cause alteration of the library file residing on disk.

In Option 2, literals contained in a library file cannot be replaced by literals, words or data-names.

In Option 2, if an integer is used for a word and it is the last entry in a replacing list, it must be followed by a blank and then a period. For example:

```
COPY REPLACING AAA BY HOURS,
BBB BY PAY-SCALE, 1234 BY 58b.
```

The COPY REPLACING option is exceptionally beneficial for conversion of generalized COBOL source language library routines into specific and well-named routines within a given program. For example, a generalized COBOL source language library routine may use the following data-names for their noted purposes:

Data-name	Purpose
AAA	Monthly hours worked per employee.
BBB	Employee pay-rate.

Data-name	Purpose
CCC	Employee social security number.
DDD	Employee income tax rate.
EEE	Employee year to date gross income.
FFF	Employee year to date net income.
GGG	Employee gross pay for month.
	Employee net pay for the month.
.	.
.	.
.	.
1234	Specifies a GO TO exit from the routine.

A program calling upon the above generalized routine can replace the non-descript data-names with descriptive names as defined in the programs record description or WORKING-STORAGE area. For example:

```
COPY...REPLACING AAA BY HOURS-WORKED
COPY...REPLACING BBB BY RATE-OF-PAY
COPY...REPLACING CCC BY SOC-SEC-NR
COPY...REPLACING DDD BY INC-TAX-RATE
COPY...REPLACING EEE BY YR-TO-DATE-GROSS
COPY...REPLACING FFF BY YR-TO-DATE-NET
COPY...REPLACING GGG BY THIS-MONTHS-GROSS
COPY...REPLACING HHH BY THIS-MONTHS-NET
                       .
                       .
                       .
COPY...REPLACING 1234 BY WRITE-EMPLOYEE-DRAFT.
```

The specified source program data-names and exit points will be inserted into the library file routine at every occurrence of the assigned generalized names within the routine.

LIBRARY CREATION. A library file will be created only during a COBOL compilation each time that a source card is encountered containing an L in column 7 with a library-name, bounded by quotation marks starting in Field A of the same card. A library-file may contain up to a maximum of 20,000 card images.

Each library file in the source program will be terminated when a card containing an L in column 7 followed by all blanks or another library-name is encountered.

Library-names cannot start with a blank character or a dash (-).

Once a file has been created, it may be COPYed by other programs, or the creating program in succeeding FD, 01, or procedure COPY statements.

The source data used to create an original library file will also be compiled into the object program at the point of appearance.

All assigned library-names must be unique to other library-names contained in the library to preserve the integrity of the COBOL library system.

Library files to be used with the COPY verb can be created by a user program which creates an unblocked card image file on disk.

DISPLAY.
The function of this verb is to provide for the printing of low volume data, error messages, and operator instructions on the console typewriter.

The construct of this verb is:

```
            { literal-1   }    [{ literal-2   }         ]
DISPLAY     { data-name-1 }    [{ data-name-2 }     ... ]

[           { SPO           }]
[ UPON      { mnemonic-name }]
```

Each literal may be any figurative constant except ALL.

All special registers (DATE, TIME, etc.) may be DISPLAYed.

The DISPLAY statement causes the contents of each operand to be written on the supervisory printer (SPO) from the MCP SPO queue to ensure that a program is not operationally deterred while a message is printing.

If a figurative constant is specified as one of the operands, only a single character of the figurative constant is displayed.

The data-names may be subscripted and can be PICTUREd as COMPUTATIONAL or DISPLAY items.

An infinite amount of characters may be displayed with one statement. The compiler will supply automatic carriage returns and line feeds, as may be appropriate.

The DISPLAY series option will cause the literals or data-names to be printed on one line and, if required, the compiler will cause automatic carriage returns and line feeds for information extending to other lines of print. The compiler will format each line so that a partial word at an end of a line will not be printed on that line, and continued on the following lines.

When mnemonic-name is used, it must appear in the SPECIAL-NAMES paragraph equated to the hardware-name SPO.

DIVIDE

DIVIDE.
The function of this verb is to divide one numerical data-item into another and set the value of an item equal to the result.

The construct of this verb contains two options which are:

Option 1:

```
                   { literal-1     }
DIVIDE  [MOD]      { data-name-1   }    INTO  data-name-2 [ROUNDED]

     [ON SIZE ERROR any statement]
```

Option 2:

```
                  { literal-1    }  { BY   } { literal-2    }
DIVIDE  [MOD]     { data-name-1  }  { INTO } { data-name-2  }

    GIVING data-name-3 [ROUNDED]

    REMAINDER data-name-4 [ROUNDED]

    [ON SIZE ERROR any statement]
```

Data-name-3 and data-name-4 of Option 2 may refer to a data item that contains editing symbols.

Each literal must be a numeric literal.

Division by zero is not permissible and, if executed, will result in a size error indication. This can be handled programmatically, either by doing a zero test prior to the division, or by the use of the SIZE ERROR clause. If SIZE ERROR is not written, an attempt to divide by zero will result in unpredictable results. Processing will continue.

All data-names must refer to elementary numeric items.

In Option 1, the value of the operand preceding the word INTO will be divided into the operand following INTO and the resulting quotient stored as the new value of the latter.

The use of the BY option will cause literal-1/data-name-1 to be divided _by_ literal-2/data-name-2, whereas the INTO option will cause literal-1/data-name-1 to be divided _into_ literal-2/data-name-2.

In Option 2, the resulting quotient will be stored as the new value of data-name-3. The value of the operands immediately to the left of the word GIVING will remain unchanged.

The ROUNDED option and ON SIZE ERROR clause and truncation are the same as discussed for the ADD statement (refer to page 5-22).

The size of the operands is determined by the sum of the divisor and the quotient. The sum of the two cannot exceed 99 digits.

The use of the MOD option will cause the remainder to be placed in data-name-2 of Option 1 and data-name-3 of Option 2. The remainder will be carried to the same degree of accuracy as defined in the PICTURE of the quotient and all extra positions will be filled with zeros.

Literals cannot be used as dividends.

The use of the REMAINDER option will cause the remainder to be placed in data-name-4 and data-name-3 will contain the quotient, unless the MOD option is also included. If the MOD option is included, both data-name-3 and data-name-4 will contain the remainder.

END-OF-JOB

END-OF-JOB.
The function of this verb is to notify the COBOL Compiler that all source statements within a program have been read.

The construct for this indicator is:

```
     END-OF-JOB.
```

The END-OF-JOB statement is for documentation only but if used it must be the last source program card in a COBOL deck. It immediately precedes the MCP END Control Card.

EXAMINE.
The function of this verb is to replace a specified character, and/or to count the number of occurrences of a particular character in a data item.

The construct of this verb is:

```
    EXAMINE  data-name

  ( TALLYING  ( ALL         )  ( literal-1     ) [ REPLACING BY  ( literal-2     ) ] )
  (           ( LEADING     )  ( data-name-1   ) [               ( data-name-2   ) ] )
  (           ( UNTIL FIRST )                                                       )
  (                                                                                 )
  ( REPLACING   ( ALL           )  ( literal-3     )      ( literal-4     )          )
  (             ( LEADING       )  ( data-name-3   )  BY  ( data-name-4   )          )
  (             ( [UNTIL] FIRST )                                                   )
```

The description of data-name must be such that USAGE is DISPLAY explicitly or implicitly.

Each literal used in an EXAMINE statement must consist of a single DISPLAY character. Figurative constants will automatically represent a single DISPLAY character.

Examination proceeds as follows:

a. For items that are not numeric (4-bit), examination starts at the left-most character and proceeds to the right. Each 8-bit character in the item specified by the data-name is examined in turn. Any reference to the first character means the left-most character.

b. If an item referenced by the EXAMINE verb is numeric, it must consist of numeric (8-bit) characters and may possess an operational sign. Examination starts at the left-most character (excluding the sign) and proceeds to the right. Each character except the sign is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE verb. Any reference to the first character means the left-most numeric character.

The TALLYING option creates an integral count (i.e., a tally) which replaces the value of a special register called TALLY. The count represents the number of:

a. Occurrences of literal-1 or data-name-1 when the ALL option is used.

b. Occurrences of literal-1 or data-name-1 prior to encountering a character other than literal-1 or data-name-1 when the LEADING option is used.

c. Characters not equal to literal-1 or data-name-1 encountered before the first occurrence of literal-1 or data-name-1 when the UNTIL FIRST option is used.

When either of the REPLACING options is used (i.e., with or without TALLYING) the replacement rules are as follows:

a. When the ALL option is used, then literal-2 or data-name-2 or literal-4 or data-name-4 is substituted for each occurrence of literal-1 or data-name-1 or literal-3 or data-name-3.

b. When the LEADING option is used, the substitution of literal-2 or data-name-2 or literal-4 or data-name-4 terminates as soon as a character other than literal-1 or data-name-1 or literal-3 or data-name-3 or the right-hand boundary of the data item is encountered.

c. When the UNTIL FIRST option is used, the substitution of literal-2 or data-name-2 or literal-4 or data-name-4 terminates as soon as literal-1 or data-name-1 or literal-3 or data-name-3 or the right-hand boundary of the data item is encountered.

d. When the FIRST option is used, the first occurrence of literal-3 or data-name-3 is replaced by literal-4 or data-name-4.

The field called TALLY is a 5-digit field provided by the compiler. Its usage is COMPUTATIONAL and will be reset to zero automatically when the EXAMINE...TALLY option is encountered.

EXIT.
The function of this verb is to provide a terminating point for a PERFORM loop, whenever required.

The construct of this verb is:

```
EXIT.
```

If the EXIT statement is used, it must be preceded by a paragraph-name and appear as a single one-word paragraph. EXIT is documentational only, but if used, must follow the rules of COBOL.

The EXIT is normally used in conjunction with conditional statements contained in procedures referenced by a PERFORM statement. This allows branch paths within the procedures to rejoin at a common return point.

If control reaches an EXIT paragraph and no associated PERFORM or USE statement is active, control passes through the EXIT point to the first sentence of the next paragraph and is treated for all intents and purposes as a NOP (No Operation).

GO TO

GO TO.
The function of this verb is to provide a means of breaking out of the sequential, sentence by sentence, execution of code, and to permit continuation at some other location indicated by the procedure-name(s).

The construct of this verb has two options which are:

Option 1:

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

Option 2:

```
    GO TO procedure-name-1 procedure-name-2 [procedure-name-3...]

        DEPENDING ON data-name.
```

Each procedure-name is the name of a paragraph or section in the PROCEDURE DIVISION of the program.

In Option 2, GO TO... DEPENDING... may specify up to 1023 procedure-names in a single statement.

In Option 2, the data-name in the format following the words DEPENDING ON must be a numeric elementary item described without any positions to the right of the assumed decimal point. Furthermore, the value must be positive in order to pass control to the procedure-names specified. Control will be transferred to procedure-name-1 if the value of the identifier is 1, to procedure-name-2 if the value is 2, etc. If the value of the identifier is anything other than a positive integer, or if its value is zero, or its value is higher than the number of procedure-names specified, control will be passed to the next statement in normal sequence. For example:

```
GO TO MFG, RE-SALE, STOCK, DEPENDING ON S-0.
```

Value of S-0	GO TO Procedure-name
-1	next sentence
0	next sentence
1	MFG
2	RE-SALE
3	STOCK
4	next sentence

Whenever a GO TO statement (represented by Option 1) is executed, control is unconditionally transferred to a procedure-name, or to another procedure-name if the GO TO statement has been changed by an ALTER statement.

A GO TO statement is unrestricted as to where it branches to in a segmented program. It can call upon any segment (fixed or overlayable) at either section level or paragraph levels nested to any depth within a section.

When, in Option 1, the GO TO is referred to by an ALTER statement, the following rules apply regardless of whether or not procedure-name is specified:

a. The GO TO statement must have a paragraph-name.

b. The GO TO statement must be the only statement in the paragraph.

c. If the procedure-name is omitted, and if the GO TO statement is not referenced by an ALTER statement prior to the first execution of the GO TO statement, the MCP will terminate the job and cause an error message reflecting an invalid address.

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

IF.
The function of this verb is to control the sequence of commands to be executed depending on either a condition, the class status of a field, or the relative value of two quantities. The purpose of a condition is to cause the object program to select between alternate paths depending on the passing or failing of the test.

The conditions are subdivided into six major categories which are:

a. Simple conditional tests.
b. Conditional statements.
c. Relation tests.
d. Relation value tests.
e. Class tests.
f. Conditional variable tests.

SIMPLE CONDITIONAL TESTS. The simple conditional tests are contained in option 1.

Option 1:

```
IF  condition-1  statement-1
```

CONDITIONAL STATEMENTS. A conditional statement specifies that the truth value of "yes" in a given condition is to be determined and that subsequent action of the object program is contingent upon the resultant value. READ and WRITE statements which specify an INVALID KEY option, or arithmetic statements (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT) which specify a SIZE ERROR option are considered as being conditional.

In Option 2, statement-1 or statement-2 can be either imperative or conditonal. If conditional, it can in-turn contain conditional nested statements to an arbitrary depth.

Option 2:

```
IF condition  { statement-1   }  [ { OTHERWISE }
              { NEXT SENTENCE }  [ { ELSE      }

      { statement-2   } ]
      { NEXT SENTENCE } ]
```

RELATION TESTS. A relation test involves a comparison of two operands; either of which can be a data-name, a literal, or a formula. The comparison of two literals is not permitted. Comparison of elementary numeric items is permitted regardless of their individual USAGEs. All other comparisons require that the USAGE of the items being compared be the same. Group numeric items are defined to be alphanumeric. It is not permissible to compare an index-data-name against a literal or a data-name. The format of relation test is shown in Option 3.

Option 3:

```
                                                        (  =            )
                                                        (  >            )
     IF  ( literal-1                )                   (  <            )
         { data-name-1              }   IS [NOT]        {  EQUAL TO     }
         ( arithmetic expression-1  )                   (  LESS THAN    )
                                                        (  GREATER THAN )
                                                        (  EQUALS       )

         ( literal-2                )
         { data-name-2              }
         ( arithmetic expression-2  )
```

RELATIVE VALUE TESTS. The relative value test is an alternate way of stating a comparison of the value zero with a formula, or with data-name. An item or formula is POSITIVE only if its value is greater than zero. An item or formula is NEGATIVE only if its value is less than zero. The value zero is considered neither POSITIVE nor NEGATIVE. This form of comparison with zero is not considered a relational test. The format of relative value tests is as follows:

Option 4:

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

CLASS TEST. The class test is used to determine whether the contents of the data-name is made up entirely of NUMERIC or ALPHANUMERIC characters. For example:

JOHN DOE is ALPHABETIC	[PC X(8)]
R. JOHN DOE is not ALPHABETIC	[PC X(11)]
37373 is NUMERIC	[PC 9(5)]
-37452 is NUMERIC	[PC S9(5)]
685.57 is not NUMERIC	[PC X(6)]

The format of the class test is as follows:

Option 5:

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

CONDITIONAL VARIABLE TESTS. A conditional variable test is one in which an item is tested to determine whether or not the value associated with a condition-name is present. The rules for comparing a conditional variable with a conditional value are the same as those for relation tests. The format for a conditional variable test is:

Option 6:

```
IF [NOT] condition-name
```

Not Logic.
The statement:

```
IF A IS NOT EQUAL TO B OR C OR D, GO TO paragraph-name-1
            ELSE GO TO paragraph-name-2.
```

a. Condition-1. If A is not equal to B, control will transfer immediately to paragraph-name-1.

b. Condition-2. If A equals B, a test of C for inequality is set up. If C is unequal, control transfers immediately to paragraph-name-1; but if C is also equal, a test of D for inequality is set up. If D is unequal, control transfers immediately to paragraph-name-1; but if D is also equal, program control transfers immediately to paragraph-name-2.

c. Conclusion. The above explanation reflects that a test of field A versus the fields B OR C OR D for unequal status in all fields during one operation is an impossibility when using NOT/OR logic. The first data field reflecting inequality will cause a branch to be executed to paragraph-name-1.

d. In the above example, had AND logic been applied, the tests would have been accomplished in the very same manner.

MOVE.
The function of this verb is to transfer data from one area of memory to one or more data areas (receiving fields). The data will be automatically edited or adjusted as to the applicable PICTURE and USAGE clauses.

The construct of this verb is:

Option 1:

```
           { literal-1    }
   MOVE    { data-name-1  }   TO data-name-2 [data-name-3,...]
```

Option 2:

```
           { CORR          }
   MOVE    { CORRESPONDING }   data-name-1 TO data-name-2
```

The MOVE statement without the CORR or CORRESPONDING option may not be used to MOVE a group item if editing or conversion of elementary items is desired. To do this, either the CORR or CORRESPONDING option must be used, or each elementary item must be moved individually. CORR is an acceptable substitute for CORRESPONDING.

If the CORR or CORRESPONDING option is used, selected sending fields are MOVEd to selected receiving fields. Data-name-1 and data-name-2 must be group items. A pair of data items, one from data-name-1 and one from data-name-2, correspond if the data items in both have the same name and the same qualification up to, but not including, data-name-1 and data-name-2. At least one of the data items of both data-name-1 and data-name-2 must be an elementary item. Neither data-name-1 nor data-name-2 may be data items with levels 66, 77, or 88. Each data item which is subordinate to data-name-1 and data-name-2, and which contains a RENAMES clause, is ignored. Furthermore, a data item that is subordinate to data-name-1 and data-name-2 and contains a REDEFINES or OCCURS clause is ignored. However, data-name-1 and data-name-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with these clauses.

The CORR or CORRESPONDING option generates the following:

a. Elementary to elementary.

b. Elementary to group.

c. Group to elementary MOVEs within the two data descriptions.

Any MOVE in which the sending field and receiving items are elementary items is an elementary MOVE. Every elementary item belongs to one of the following categories: alphabetic, numeric, alphanumeric, numeric edited, or alphanumeric edited. These categories are discussed in PICTURE. Numeric literals belong to the numeric (4-bit) category, and non-numeric literals belong to the alphanumeric (byte) category. The following rules apply to an elementary MOVE between these categories:

a. In a MOVE of ALPHABETIC information to numeric field, the results will be unpredictable.

b. A numeric edited, alphanumeric edited, or alphabetic data item must not be MOVEd to a numeric or numeric edited data item.

c. A numeric or numeric edited data item must not be MOVEd to an alphabetic item.

d. A numeric item whose implicit decimal point is not immediately to the right of the least-significant digit must not be MOVEd to an alphanumeric or alphanumeric edited data item.

e. All other elementary moves are legal and are performed according to the rules outlined below:

   1) An alphanumeric to alphanumeric elementary MOVE passes data constructed of bytes to a receiving field constructed of bytes.

   2) When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item, left justification occurs and any necessary space filling takes place to the right. If the length of the sending item is greater than the length of the receiving item, the right-most characters are truncated (see JUSTIFIED for the inverse procedure).

   3) When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero filling takes place except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, the excess digits are truncated. If the sending item contains non-numeric characters, the following actions occur:

      a) Zone bits will be stripped if the receiving field is COMP.

b) Zone bits may be replaced with the numeric stick if the receiving field is DISPLAY.

4) Any necessary conversion of data from one form of internal representation to another takes place during the MOVE, along with any specified editing in the receiving item.

Any MOVE in which one or both operands is a group item, regardless of USAGE, is treated exactly as if it were an alphanumeric to alphanumeric elementary MOVE. There will be no conversion of data from one form of internal representation to another unless one of the fields is an elementary COMPUTATIONAL item. Group COMPUTATIONAL receiving fields are treated as if they are alphanumerically declared.

The following are examples of the MOVE statement:

a. The following examples show truncation of digits in moving numeric information.

Receiving Field Picture	9999	9900	9009	990099	0099	99/99
Value	1234	1234	1234	1234	1234	1234
Receiving Field	1234	3400	3004	120034	0034	12/34
Warning Message	No	Yes	Yes	No	Yes	No

b. The following examples show alignment of decimal points in moving numeric data. The symbol V denotes the assumed decimal point given by item description PICTURE clause, but which is not physically present.

Sending Field	Receiving Field	
Before and After	Before	After
123V45	0020V20	0123V45
123V45	002V020	123V450
123V45	00202V0	00123V4

c. The following example shows results of MOVE ALL statements. The use of a figurative constant ZERO in a MOVE statement will result in the entire DISPLAY or COMPUTATIONAL elementary receiving field being composed of zeros, with or without the use of the reserved word ALL. Therefore, MOVE ALL ZEROS, MOVE ZEROS, and MOVE ALL 0 are synonymous and will cause the DISPLAY <u>or</u> COMPUTATIONAL elementary receiving field to be composed of 8-bit or 4-bit zeros respectively.

Statement	Five Position Receiving Field After Execution	
	COMPUTATIONAL	DISPLAY
MOVE ALL 9 (or "9")	99999	F9F9F9F9F9
MOVE ALL 57	57575	F5F7F5F7F5
MOVE ALL 057	05705	F0F5F7F0F5
MOVE ALL "ABC"	*	C1C2C3C1C2
MOVE ALL ZEROS	00000	F0F0F0F0F0
MOVE ALL 0	00000	F0F0F0F0F0

The asterisk above designates the data as being unpredictable.

MULTIPLY.
The function of this verb is to multiply two operands and store the results in the last-named field (which must be a numeric data-name).

The construct of this verb is:

```
                  { literal-1   }            { literal-2   }
    MULTIPLY      { data-name-1 }    BY      { data-name-2 }

          [GIVING data-name-3]  [ROUNDED]

          [ON SIZE ERROR any statement]
```

All rules specified under the ADD statement regarding the presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing results apply to the MULTIPLY statement, except the maximum operand size is 125 digits for the sum of two operands.

The data-names must be elementary item references. If GIVING is used, the data description of data-name-3 may contain editing symbols. In all other cases, the data-names used must refer to numeric items only.

If the GIVING option is used, the result of the multiplication replaces the contents of data-name-3, otherwise, it replaces the contents of data-name-2. If GIVING is not used, literal-2 is not permitted, i.e., data-name-2 must appear.

NOTE

NOTE.
The function of this verb is to allow the programmer to write explanatory statements in his program which are to be produced on the source program listing for documentational clarity.

The construct of this verb is:

Option 1 - Paragraph NOTE:

```
label. NOTE any comment.
```

Option 2 - Paragraph NOTE:

```
NOTE. any comment.
```

Option 3 - Sentence NOTE:

```
NOTE any comment.
```

Any combination of the characters from the allowable character set may be included in the character string of a NOTE statement.

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be commentary. Either Option 1 or Option 2 may be used as NOTE statements on a paragraph level.

If a NOTE statement appears as other than the first sentence of a paragraph, only the sentence constitutes a commentary. The first period after encountering the word NOTE will cause the compiler to resume compilation unless the new sentence commences with the word NOTE.

Refer to page 7-3 of section 7, CONTINUATION INDICATOR, for an explanation of notes (* or / in column 7) appearing anywhere within the source program.

OPEN.
The function of this verb is to initiate the processing of both input and output files. The MCP performs checking or writing, or both, of labels and other input-output operations.

The construct of this verb is:

```
   OPEN

 [                        [( WITH LOCK [ACCESS] )]                  ]
 [ INPUT file-name-1      [{ REVERSED           }]  [file-name-2...] ]
 [                        [( WITH NO REWIND     )]                  ]

 [ OUTPUT file-name-3 [WITH NO REWIND] [file-name-4 ...] ]

   [ ( INPUT-OUTPUT )                                ]
   [ ( I-O          ) file-name-5   [file-name-6...] ]

    [ O-I file-name-7 [file-name-8...] ]
```

File-names must not be those defined as being SORT files.

At least one of the options must be specified before a file can be read. A statement of OPEN INPUT.......OUTPUT.......I/O........ O/I.......can appear in one source language card. Continuation of source card lines is allowed.

The I-O, INPUT-OUTPUT and O-I options pertain to disk storage files.

The OPEN statement must be executed prior to the first SEEK, READ, or WRITE statement for that file.

A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

A file area will not exist in memory until an OPEN statement is executed, which in turn, causes the MCP to allocate memory for the file work area, and any alternate areas or buffers. The MCP will obtain the needed information from the File Parameter Block to determine the file's characteristics. Once the file has been OPENed, memory will remain allocated until the file is programmatically CLOSEd.

The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record.

When checking or writing the first label, the user's beginning label subroutine is executed if it is specified by a USE statement.

The REVERSED and the NO REWIND options can only be used with SEQUENTIAL, single reel, tape files.

If the peripheral ASSIGNed to the file permits rewind action, the following rules apply:

a. When neither the REVERSED nor the NO REWIND option is specified, execution of the OPEN statement for the file will cause the file to be positioned ready to read the first data-record.

b. When either the REVERSED or the NO REWIND option is specified, execution of the OPEN statement does not cause the file to be positioned. When the REVERSED option is specified, the file must be positioned at its physical end. When the NO REWIND option is specified, the file must be positioned at its physical beginning.

c. When the NO REWIND option is specified, it applies only to sequential, single reel files stored on magnetic tape units.

When the REVERSED option is specified, the subsequent READ statements for the file makes the data-records available in reverse record order starting with the last record. Each record will be read into its record-area, and will appear as if it has been read from a forward moving file.

If an input file is designated with the OPTIONAL clause in the File-Control paragraph of the ENVIRONMENT DIVISION, the object program causes an interrogation to the MCP for the presence or absence of a pertinent file. If this file is not present, the first READ statement for this file causes the imperative statement in the AT END clause to be executed.

The I-O or INPUT-OUTPUT option permits the OPENing of a disk file for input and or output operations. This option demands the existence of the file to be on the disk and cannot be used if the file is being initially created. That is, the file to be OPENed must be present in the MCP Disk Directory, or has previously been created and CLOSEd in the same run of the program.

When the I-O or INPUT-OUTPUT option is used, the MCP immediately checks the MCP Disk Directory to see if the file-name is present, or has been created and CLOSEd in the same program run. The system operator will be notified in its absence, and the file can then be loaded if it is available or the program can be DSed (Discontinued). If the decision is to load the file, the operator does so and then notifies the MCP to proceed with the program by a "mix-index OK" message.

The O-I option is identical to OPEN I-O with the exception being that the file is assumed to be a new file to the Disk Directory. The OPEN O-I option will short cut the usual method of initially creating I-O work files within a program, e.g., OPEN OUTPUT, write record(s), CLOSE WITH RELEASE, OPEN I-O, etc. The O-I option does not, nor was it intended to, replace the OPEN I-O option, since the use of OPEN O-I assumes that a new file is to be created each time.

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

The contents of the data-names specified in the FILE-LIMIT clause of the File-Control paragraph (at the time the file is OPENed) is used for all checking operation while that file is OPEN. The FILE-LIMIT clause is dynamic only to this extent.

When an OPEN OUTPUT statement is executed for a magnetic tape file, the MCP searches the assignment table for an available scratch tape, writes the label as specified by the program, and executes any USE declaratives for the file. If no scratch tape is available, a message to the operator is typed and the program is suspended until the operator mounts one, or one becomes available due to the termination of a multiprocessing program.

OPENing of subsequent reels of multi-reel tape files is handled automatically by the MCP and requires no special consideration from the programmer.

PERFORM

PERFORM.
The function of this verb is to depart from the normal sequence of execution in order to execute one or more procedures, either a specified number of times or until a specified condition is satisfied. Following this departure, control is automatically returned to the normal sequence.

The construct of this verb has four options which are:

Option 1:

```
                                     ⎡⎧ THRU    ⎫                      ⎤
     PERFORM procedure-name-1        ⎣⎩ THROUGH ⎭   procedure-name-2   ⎦
```

Option 2:

```
                                     ⎡⎧ THRU    ⎫                      ⎤
     PERFORM procedure-name-1        ⎣⎩ THROUGH ⎭   procedure-name-2   ⎦

       ⎧ integer-1   ⎫
       ⎩ data-name-1 ⎭      TIMES
```

Option 3:

```
                                     ⎡⎧ THRU    ⎫                      ⎤
     PERFORM procedure-name-1        ⎣⎩ THROUGH ⎭   procedure-name-2   ⎦

        UNTIL condition-1
```

Option 4:

```
                                     ⎡⎧ THRU    ⎫                      ⎤
     PERFORM procedure-name-1        ⎣⎩ THROUGH ⎭   procedure-name-2   ⎦

                                           ⎧ index-name-2      ⎫
     VARYING    ⎧ index-name-1 ⎫   FROM    ⎨ data-name-2       ⎬     BY
                ⎩ data-name-1  ⎭           ⎩ numeric-literal-1 ⎭

   ⎧ data-name-3       ⎫                           ⎡          ⎧ index-name-3 ⎫
   ⎩ numeric-literal-2 ⎭  UNTIL condition-1        ⎣  AFTER   ⎩ data-name-4  ⎭
```

```
         { index-name-4      }        { data-name-6       }
  FROM   { data-name-5       }   BY   { numeric-literal-4 }
         { numeric-literal-3 }

                              ]   [          { index-name-5 }
  UNTIL   condition-2         ]   [ AFTER    { data-name-7  }    FROM

  { index-name-6      }
  { data-name-8       }   BY     { data-name-9       }
  { numeric-literal-5 }          { numeric-literal-6 }
                               ]
  UNTIL   condition-3          ]
```

PERFORM is the means by which subroutines are executed in COBOL. The subroutines may be executed once, or a number of times, as determined by a variety of controls. A given paragraph may be PERFORMed by itself, in conjunction with another paragraph, control may pass through it in sequential operation, and it may be the object of a GO statement, all in the same program. The range of a PERFORM starts with the first executable statement of procedure-name-1 and continues in logical sequence through the last executable statement of:

a. THRU procedure-name-2, if specified, automatically sets up a return to the statement following the PERFORM statement.

b. Procedure-name-1 only, if procedure-name-2 is not specified automatically sets up a return to the statement following the PERFORM statement.

c. The automatic return is implied as immediately following the last statement in a PERFORM range.

Each procedure-name is the name of a section or a paragraph in the PROCEDURE DIVISION.

Each data-name is a numeric elementary item described in the DATA DIVISION. All literals must represent numeric items with no positions to the right of the assumed decimal point.

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

If the object program control passes to procedure-name-1 or procedure-name-2 from a statement other than a PERFORM, the procedure(s) will be accomplished and control will fall through to the next sentence following the procedure(s). If procedure-name-2 consists of an EXIT, program control will pass to the next sentence following procedure-name-2.

If a statement within procedure-name-1 or procedure-name-2 contains a nested PERFORM, object program control will pass to the procedure-name contained in the nested statement and the procedure will be accomplished. Program control will automatically return to the next sentence following the executed PERFORM statement. Nested PERFORM statements are allowed to any reasonable depth. However, the procedure named must return to the statement following the previously executed PERFORM and cannot contain a GO TO out of range of procedure-name-1 or procedure-name-2.

A PERFORM statement is not restricted by overlayable segment boundries and may reference a procedure-name anywhere within the PROCEDURE DIVISION.

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

Option 2 is the TIMES option and, when used, the procedures are performed the number of times specified by data-name-1 or integer-1. Data-name-1 cannot be described as larger than 6 digits in length. The value of data-name-1 or integer-1 must be positive. Control is transferred to the statement following the PERFORM statement. If the value is zero, control passes immediately to the statement following the PERFORM sentence. Once the PERFORM statement has been initiated, any reference to or manipulation of data-name-1 will not affect the number of times the procedures are executed.

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

Option 4 is the VARYING option. This option is used when it is desired to augment the value of one or more data-names or index-names in an orderly fashion during the execution of a PERFORM statement. When index-names are used, the FROM and BY clause have the same effect as in a SET statement.

In Option 4 where only one condition is required to control the number of iterations that a procedure is to be PERFORMed, the following actions take place:

a. Data-name-1 is set at the start of the PERFORM to a starting value as contained in data-name-2 (or numeric-literal-1).

b. Condition-1 is compared for an EQUAL condition. If condition-1 is true, control passes to the next statement.

c. Procedure-name will be executed one time.

d. Data-name-3 is added to the contents of data-name-1.

e. Loop to step b above.

The above cycle continues until an equal comparison occurs, at which point program control directly passes to the next sentence following the executed PERFORM statement.

In Option 4 where two conditions are required to control the number of iterations that a given procedure is to be PERFORMed, the following actions occur:

a. Data-name-1 and data-name-4 are set at the start of the PERFORM to starting values as contained in data-name-2 (or numeric-literal-1) and data-name-5 (or numeric-literal-3) respectively.

b. Condition-1 is compared to data-name-1 and:

   1) If an equal condition occurs, control is passed to the next sentence following the executed PERFORM statement, or else:

   2) Condition-2 is compared to data-name-4 and:

      a) If an equal condition occurs, data-name-4 is set to the value contained in data-name-5. Data-name-3 is added to the data-name-1 and loop to step b above, or else:

      b) Procedure-name will be executed one time, after which data-name-6 is added to data-name-4 and loop to step a above.

The above cycle continues until an equal comparison occurs, at which point program control directly passes to the next sentence following the executed PERFORM statement.

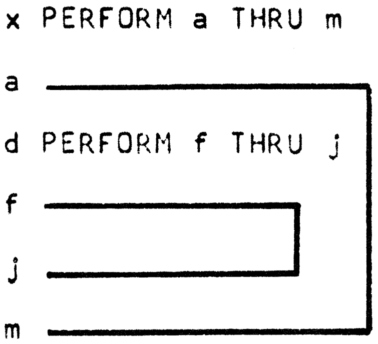
NOTE

Data-name-3, data-name-6 and data-name-9 cannot contain zeros.

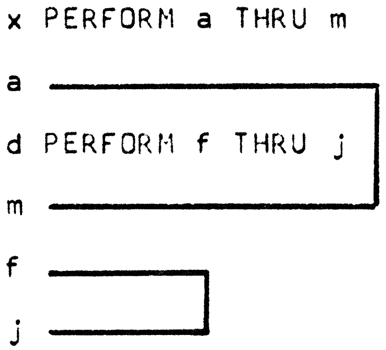
In Option 4 where three conditions are required to control the number of iterations that a given procedure is to be PERFORMed, the mechanism is the same as for two-conditional control except that data-name-7 goes through a complete cycle each time that data-name-6 is added to data-name-4, which in turn goes through a complete cycle each time that data-name-1 is varied.

After the completion of Option 4, data-name-4 and data-name-7 contain their initial values, while data-name-1 contains a value which exceeds its last used setting by one increment or decrement unless condition-1 is TRUE when the PERFORM statement is entered, in which case data-name-1, data-name-4 and data-name-7 all contain their initial values.

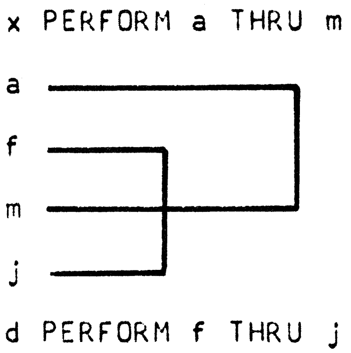
Since the return control information is placed in the stack rather than directed through instruction address modification, a PERFORM statement executed within the range of another PERFORM is not restricted in the range of paragraph names it may include. The examples shown below are permitted and will execute correctly.



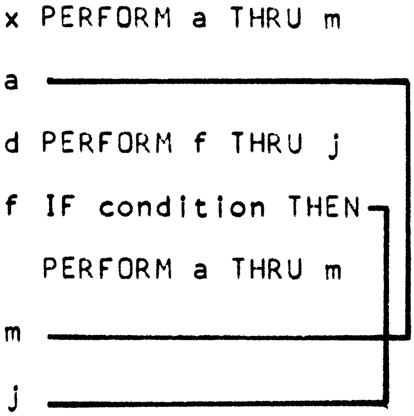
```
x PERFORM a THRU m

a

d PERFORM f THRU j

f

j

m
```



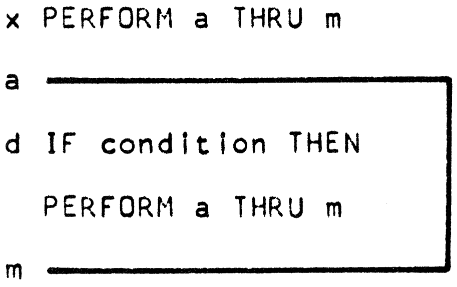
```
x PERFORM a THRU m

a

d PERFORM f THRU j

m

f

j
```



```
x PERFORM a THRU m

a

f

m

j

d PERFORM f THRU j
```



```
x PERFORM a THRU m

a

d PERFORM f THRU j

f IF condition THEN

  PERFORM a THRU m

m

j
```



```
x PERFORM a THRU m

a

d IF condition THEN

  PERFORM a THRU m

m
```

READ.
The function of this verb is twofold, namely:

a. When processing sequential input files, a READ statement will cause the next sequential record to be moved from the input buffer area to the actual work area, thus making the record available to the program. If the file has been declared BLOCKED, or, if an ALTERNATE AREA has been ASSIGNed this will be in addition to the normal buffer.

   All sequential records will be physically read into the buffer area of the file. Physical READs are performed as a function of the MCP. The READ statement permits the performance of a specified statement when an end-of-file condition is detected by the MCP.

b. For random file processing, the READ statement communicates with the MCP to explicitly cause the reading of a physical record from a disk file and also allows performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item is found to be invalid.

The construct of this verb is:

```
                                          [ { AT END      }
READ file-name RECORD [INTO data-name]    [ { INVALID KEY }

         any statement ]
```

The AT END of file clause is used only for non-disk files or for disk files being processed in the sequential access mode. If no AT END or INVALID KEY clause is stated, and one of these conditions occurs, the program will be terminated with a DS or DP message.

If, during execution of a READ statement with AT END, the logical End-of-File is reached and an attempt is made to READ that file, the statement specified in the AT END phrase is executed. After the execution of the imperative statement of the AT END phrase, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file.

When the AT END clause is specified in a conditional sentence, all exits within the sentence are controlled by using the rules pertaining to the matching of IF...ELSE pairs. For example:

```
IF AAA = BBB THEN READ FILE-A, AT END
    GO TO WRAP-UP, ELSE STOP RUN.
```

a. When AAA does not equal BBB, control will be passed to STOP RUN.

b. When AAA equals BBB, FILE-A is read, end-of-file is tested and if the result is "TRUE" program control will be transferred to the WRAP-UP procedure, however, a result of "FALSE" will cause program control to be transferred to the next sentence.

The INVALID KEY applies to files that are ASSIGNed to disk. The access of the file is controlled by the value contained in ACTUAL KEY.

An AT END or INVALID KEY clause must be specified when reading a file described as containing FILE-LIMITS.

The INTO option may only be used when the input file contains records of one type. The data-name must be the name of a WORKING-STORAGE area or output record area.

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

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

If the INTO option is specified, the current record is MOVEd from the input area to the area specified by data-name according to the rules for the MOVE statement without the CORRESPONDING option. If multiple 01 levels are declared in the file description, the size of the first 01 level is used.

When the INTO option is used, the record being read is available in both the data area associated with data-name and the input record area.

If a file described with the OPTIONAL clause is not present, the imperative statement in the AT END phrase is executed on the first READ. The standard End-of-File procedures are not performed. (See the OPEN and USE statements, and the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION.)

If the end of a magnetic tape file is recognized during execution of a READ statement, the following operations are carried out:

a. The standard ending reel label procedure and the user's ending reel label procedure, if specified by the USE statement, are carried out. The order of execution of these two procedures is specified by the USE statement.

b. A tape swap is performed.

c. The standard beginning reel label procedure and the user's beginning label procedure, if specified, are executed. The order of execution is again specified by the USE statement.

d. The first data record on the new reel is made available.

READ with INVALID KEY is used for disk files in the random access mode. The READ statement implicitly performs the functions of the SEEK statement, except for the function of the KEY CONVERSION option for a specific disk file. If the contents of the associated ACTUAL KEY data item is out of the range indicated by FILE LIMITS, the INVALID KEY phrase will be executed.

For random disk files, the sensing of an INVALID KEY does not preclude further READs on that file nor need it be closed and reopened before doing so.

RELEASE

RELEASE.
The function of this verb is to cause records to be transferred to the initial phase of a SORT operation.

The construct of this verb is:

```
    RELEASE record-name [FROM data-name]
```

A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement.

In the FROM option, the data-name must refer to a WORKING-STORAGE, or an input-record area.

Record-name and data-name must name different memory areas when specified.

The RELEASE statement causes the contents of record-name to be released to the initial phase of a sort. Record-name will be transferred to the specified sort-file (SD) and becomes controlled by the sort operation.

In the FROM option, the contents of data-name are MOVEd to record-name, then the contents of record-name are released to the initial phase of a sort. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING option. The record-name area will not contain intelligible data after the MOVE, however, the information in data-name is still available.

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

RETURN.
The function of this verb is to obtain sorted records from the final phase of a SORT operation.

The construct of this verb is:

```
    RETURN  file-name  RECORD  [INTO data-name]

          [AT END any statement]
```

File-name must be a sort file with a Sort File Description (SD) entry in the DATA DIVISION.

A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name.

The INTO option may only be used when the input file contains just one type of record. The data-name specified must be the name of a WORKING-STORAGE, or an output-record area.

Records automatically share the same area when a file consists of more than one type record and only the information pertinent to the current record is available.

The execution of the RETURN statement causes the next record, in the order specified by the Keys listed in the SORT statement, to be made available for processing in the record area associated with the SORT file (SD).

Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option.

When the INTO option is specified, the sorted data is available in both the input-record area and the data-area specified by data-name.

RETURN statements may not be executed within the current SORT output procedure after the AT END clause has been executed.

SEARCH

SEARCH.
The function of this verb is to cause a search of a table to locate a table-element that satisfies a specific condition and, in turn, to adjust the associated index-name to indicate that table-element.

The construct of this verb has two options which are:

Option 1:

```
SEARCH data-name-1     [ VARYING { index-name-1 } ]
                       [         { data-name-2  } ]

     [AT END any statement]

     WHEN condition-1      { imperative statement-2 }
                           { NEXT SENTENCE          }

   [ WHEN condition-2      { imperative statement-3 }      ]
   [                       { NEXT SENTENCE          }  ... ]
```

Option 2:

```
SEARCH ALL data-name-3 [AT END any statement-4]

     WHEN condition-3     { imperative statement-5 }
                          { NEXT SENTENCE          }
```

Data-name-1 and data-name-3 may not be subscripted or indexed, but their descriptions must contain an OCCURS clause and an INDEXED BY option.

When Option 2 is specified, the description of data-name-3 may optionally contain the ASCENDING/DESCENDING KEY clause.

When using the VARYING option, data-name-2 must be described as USAGE IS INDEX, or as the name of a numeric elementary item described without any positions to the right of the assumed decimal point. Data-name-2 will be incremented at the same time as the occurrence number (and by the same amount) represented by the index-name associated with data-name-1.

When using Option 1, condition-1, condition-2, etc., may be comprised of any conditional as described by the IF verb.

When using Option 2, condition-3 may consist of a relational condition incorporating the relation EQUAL, or a condition-name condition where the VALUE clause that describes the condition-name contains only a single literal. Condition-3 may be a compound condition formed from simple conditions of the type just mentioned, with AND being the only acceptable connective.

When using Option 2, any data-name that appears in the KEY option of data-name-3 may appear as the subject _or_ object of a test, _or_ be the name of the conditional variable with which the tested condition-name is associated.

When using Option 1, a serial type search operation takes place, starting with the current index setting. The search is immediately terminated if, at the start of execution of the statement, the index-name associated with data-name-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for data-name-1. Then, if the AT END option is specified, statement-1 is executed; if AT END is not specified, control passes to the NEXT SENTENCE.

When using Option 1, if at the start of execution of the SEARCH statement, the index-name associated with data-name-1 contains a value that corresponds to an occurrence number that is _not_ greater than the highest permissible occurrence number for data-name-1, the SEARCH statement will begin evaluating the conditions in the order that they are written, making use of index settings wherever specified, to determine the occurrences of those items to be tested. If none of the conditions are satisfied, the index-name for data-name-1 is incremented to obtain a reference to the next occurrence. The process is repeated using the new index-name setting for data-name-1 which corresponds to a table element which exceeds the last setting by one more occurrence until such time as the highest permissible occurrence number is exceeded, in which case the SEARCH terminates as indicated in the previous paragraph.

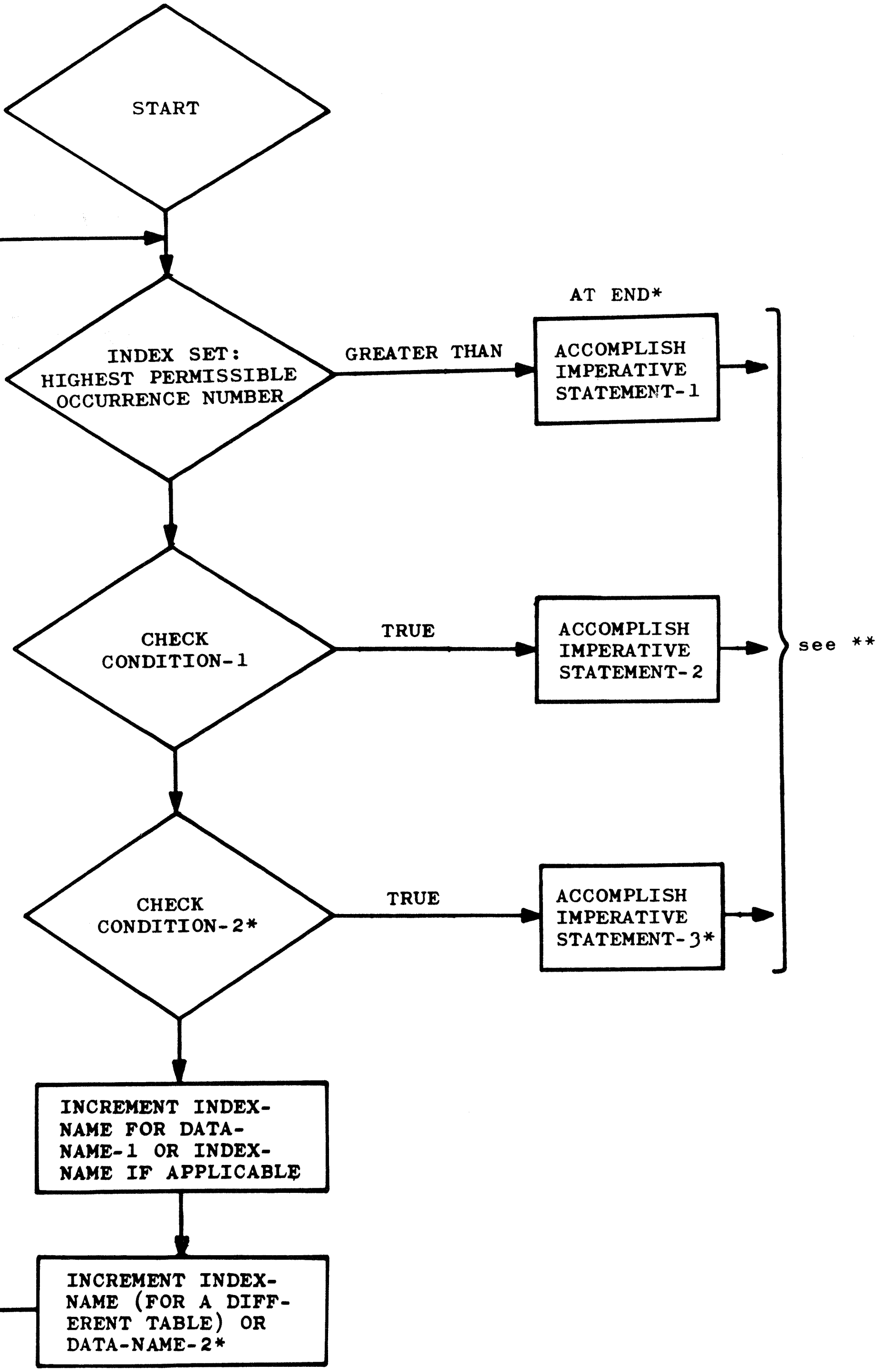
When using Option 1, if one of the conditions is satisfied upon its evaluation, the SEARCH terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

In Option 1 and 2, if the specified imperative statements do not terminate with a GO statement then program control will pass to the next sentence after the execution of the imperative statement.

In the VARYING option, if index-name-1 appears in the INDEXED BY option of data-name-1, then _that_ index-name will be used for the SEARCH, otherwise, the first index-name given in the INDEXED BY option of data-name-1 will be used. If index-name-1 appears in the INDEXED BY option of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with data-name-1 is incremented.

In Option 2, the initial setting of the index-name for data-name-3 is ignored, the effect being the same as if it were SET to 1.

In Options 1 and 2, if data-name-1 and data-name-3 is an item in a group, or a hierarchy of groups, whose description contains an OCCURS clause, then each of these groups must also have an index-name associated with it. The settings of these index-names are used throughout the execution of the SEARCH statement to refer to data-names-1 and 3, or items within its structure. These index settings are not modified by the execution of the SEARCH statement (unless stated as index-name-1) and only the index-name associated with data-name-1 and 3 (and data-name-2 or index-name-1) is incremented by the SEARCH. Figure 5-1 provides an example of SEARCH operation as related to Option 1.



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

** Each of the control transfers is to NEXT SENTENCE unless the imperative statement ends with a GO statement.

Figure 5-1. Example of SEARCH Operation Relating To Option 1

SEEK

SEEK.
The function of this verb is to initiate the accessing of a disk file record for subsequent reading and/or writing.

The construct of this verb is:

```
    SEEK file-name RECORD [WITH KEY CONVERSION]
```

The specification of the KEY CONVERSION clause indicates that the user provided USE FOR KEY CONVERSION section in the DECLARATIVE SECTION is to be executed prior to the execution of the SEEK statement. If there are no DECLARATIVES for KEY CONVERSION in a SEEK statement, then the KEY CONVERSION clause will be ignored.

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

The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution, the determination is made as to the validity of the contents of the ACTUAL KEY data item for the particular disk storage file. If the key is invalid, the imperative statement in the INVALID KEY clause of the next executed READ or WRITE statement for the associated file is executed.

Two SEEK statements for a disk storage file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of a second implicit or implied SEEK statement.

An implied SEEK is executed by the MCP whenever an explicit SEEK is missing for the specified record. An implied SEEK never executes any USE KEY CONVERSION Declaratives.

If a READ/WRITE statement for a file ASSIGNed to DISK is executed, but an explicit SEEK has not been executed since the last previous READ or WRITE for the file, then the implied SEEK statement is executed as the first step of the READ/WRITE statement.

An explicit alteration of ACTUAL KEY after the execution of an explicit SEEK has been performed, but prior to a READ/WRITE, will cause the initiation of an implied SEEK of the initial record in the sequence. For example,

a. If ACTUAL KEY is 10, then
b. READ record 10, then
c. MOVE 50 to ACTUAL KEY, then
d. WRITE record 50.

An implied SEEK of record 50 will be performed between actions c. and d. above.

SET

SET.
The function of this verb is to establish reference points for table handling operations by setting index-name values associated with table elements.

The construct of this verb has two options which are:

Option 1:

```
SET   { index-name-1 }   [ { index-name-2 }          ]
      { data-name-1  }   [ { data-name-2  }      ... ]

      TO     { index-name-3 }
             { data-name-3  }
             { literal-1    }
```

Option 2:

```
SET   index-name-4  [index-name-5 ...]

      UP BY          { data-name-4 }
      DOWN BY        { literal-2   }
```

All data-items must be either index-data-names or numeric elementary items described without any positions to the right of the assumed decimal point, except that data-name-4 must not be an index-data-name. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

An index-data-name must be defined in the WORKING-STORAGE section with the USAGE IS INDEX clause.

An index-name must be defined in an OCCURS clause.

An index-data-name cannot be SET...TO... a literal or to a data-name.

A data-name cannot be SET...TO... an index-data-name, a literal or another data-name. A data-name can only be SET to an index-name.

Literals cannot be SET...TO anything.

The SET verb appears somewhat similar to the MOVE but has a major difference in that the receiving field appears as the first operand(s) in the statement. For example:

SET A TO B

The above statement causes the contents of A to change to the value contained in B. Series statements may result in more efficient object code than separate statements. For example:

```
SET A, C, D, E, F TO B
```

Depending on the operands in a SET statement, code generated will vary from a single MVN through a series of MVN, MUL and DIV instructions. Because of this, care must be used in determining what type of receiving operand is going to be SET to what type of sending operand, since this is the primary step in calculating the location within the row. For example:

```
SET INDEX-DATA-NAME-A TO INDEX-A
SET INDEX-B TO INDEX-DATA-NAME-A
```

Both of the above statements are, by COBOL definition, plain MOVEs and unless the two indexes refer to rows of exactly the same size, will probably not result in an address which the programmer has perceived. If instead, the statement had been written: SET INDEX-B TO INDEX-A, the necessary MOVE, DIVIDE and MULTIPLY instructions would be generated to reduce the "sending" index to a relative occurrence (subscript) and then to expand it to the receiving address.

SORT.
The function of this verb is to sort an input file of records by transferring such data into a disk sort-file (work file) and sorting those records on a set of specified keys. The final phase of the sort operation makes each record available from the sort-file, in sorted order, to an output procedure or to an output file.

The construct of this verb is:

```
SORT  file-name-1

[ { PURGE }                          ]
[ { RUN   }      ON ERROR            ]
[ { END   }                          ]

      { DESCENDING }
   ON { ASCENDING  }   KEY data-name-1 [data-name-2...]

 [    { DESCENDING }                                   ]
 [ ON { ASCENDING  }   KEY data-name-3 [data-name-4...] ]

{ INPUT PROCEDURE IS section-name-1  [ { THRU    }  section-name-2 ] }
{                                    [ { THROUGH }                 ] }
{ USING file-name-2     [ LOCK    ]                                  }
{                       [ PURGE   ]                                  }
{                       [ RELEASE ]                                  }

{ OUTPUT PROCEDURE IS section-name-3  [ { THRU    }  section-name-4 ] }
{                                     [ { THROUGH }                 ] }
{ GIVING file-name-3    [ LOCK    ]                                   }
{                       [ RELEASE ]                                   }
```

File-name-1 must be described in a Sort File Description (SD) entry in the DATA DIVISION and file-names-2 and 3 must be described in a File Description (FD) entry.

Section-name-1 specifies the name of the input procedure to be used before passing each record to the sort-file, while section-name-3 specifies the output procedure to be used to obtain each sorted record from the sort-file.

Each data-name must represent data-items described in records associated with file-name-1. Data-names following the word KEY are listed from left to right in the order of decreasing significance without regard as to their division into optional KEY clauses.

The PROCEDURE DIVISION of a source program may contain more than one SORT statement appearing anywhere in the program, except in the DECLARATIVES portion or in the input/output procedures associated with a sort statement.

The _input_ procedure must consist of one or more sections that are written consecutively and which do not form a part of an output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file after the object program has accomplished the required input data manipulation specified in the procedure. Input procedures can select, create and/or modify records, one at a time, as specified by the programmer.

There are three restrictions placed on procedural statements within an input or output procedure:

a. The procedure _must not_ contain any SORT statements.

b. The input or output procedures _must not_ contain any transfers of program control outside the range of the procedure; ALTER, GO and PERFORM statements within the procedure are not permitted to refer to procedure-names outside of the input or output procedure.

c. The remainder of the PROCEDURE DIVISION must not contain any transfers of program control to points within the input or output procedure; ALTER, GO, and PERFORM statements in the remainder of the PROCEDURE DIVISION must not refer to procedure-names within the range of the input or output procedure.

The _output_ procedure must consist of one or more sections that are written consecutively and which do not form a part of an input procedure. The output procedure must include at least one RETURN statement in order to make each sorted record available for processing after the file has been sorted and the object program has accomplished the required output data manipulation specified in the procedure. Output procedures can select, create and/or modify records, one at a time, as they are being returned from the sort-file.

When the ASCENDING clause is specified, the sorted sequence of the affected records is from the lowest to the highest value according to the binary collating sequence, per specified KEY.

When the DESCENDING clause is specified, the sorted sequence of the affected records is from the highest to the lowest value according to the binary collating sequence, per specified KEY.

The SD record description of the sort-file must contain fully defined data-name KEY items in the relative positions of the record as applicable. A rule to follow when using these KEY items is that when a KEY item appears in more than one type of record, the data-names must be relatively equivalent in each record and may not contain, or be subordinate to, entries containing an OCCURS clause.

When an INPUT procedure is specified, object program control will be passed to that procedure automatically as an implicit function of encountering the generated SORT verb object code compiled into the program. The compiler will insert a return-to-the-sort mechanism at the end of the last section in the input procedure and when program control passes the last statement of the input procedure, the records that have been RELEASED to file-name-1 commence being sorted.

If the USING option is specified, all records residing in file-name-2 will be automatically transferred to file-name-1 upon encountering the generated SORT verb object code. At the time of execution of the SORT statement, file-name-2 must not be OPEN. The SORT statement automatically performs the function necessary to OPEN, READ, USE and CLOSE file-name-2. If file-name-2 is a disk file, it must be in the Disk Directory before the SORT Intrinsic is called.

When an output procedure is specified, object program control will be passed to that procedure automatically as an implicit function when all records have become sorted. The compiler will insert a return-to-the-object program mechanism at the end of the last section in the output procedure and when program control passes the last statement of the output procedure, the object program will execute the next statement following the pertinent SORT statement.

If the GIVING option is specified, all sorted records residing in file-name-1 are automatically transferred to the OUTPUT file as specified in file-name-3. At the time of execution of the sort statement, file-name-3 must not be OPEN. File-name-3 will be automatically OPENed before the sorted records are transferred from the sort-file and in turn, will be automatically CLOSEd after the last record in the sort-file has been transferred.

The ON ERROR option is provided to allow programmers some control over irrecoverable parity errors when INPUT/OUTPUT PROCEDURES are not present in a program. PURGE will cause all records in a block containing an irrecoverable parity error to be dropped and processing will be continued after a SPO message giving the relative position in the file of the bad block has been printed. This option is always assumed if no other has been defined. RUN will cause the bad block to be used by the program and will provide the same SPO message as defined for PURGE. END will cause the usual DS or DP SPO message.

The PURGE, LOCK, and RELEASE options may be used to specify the type of file close on file-name-2 and file-name-3 (see CLOSE, page 5-25). The options only apply to the USING/GIVING options.

Example:

```
SORT file-name-1 ASCENDING KEY data-name-1
USING file-name-3 PURGE
GIVING file-name-3 LOCK.
```

Beginning and ending label USE procedures are provided as follows when INPUT/OUTPUT PROCEDURES are present in the SORT statement:

a. OPEN INPUT file-name.
USE. . . (The programmer's USE procedure will be invoked).

b. OPEN OUTPUT file-name.
USE. . . (The programmer's USE procedure will be invoked).

c. CLOSE INPUT file-name.
USE. . . (The programmer's USE procedure will be invoked, however, the contents of the ending input label _will not_ be available to the USE procedure).

d. CLOSE OUTPUT file-name.
USE. . . (The programmer's USE procedure will be invoked, however, the ending label will have been written prior to executing the USE procedure).

NOTE

> The above actions provide USE on label facilities at _beginning_ and _ending_ of files, but not when switching reels of multi-reel files.

STOP

STOP.
The function of this verb is to halt the object program temporarily or to terminate execution.

The construct of this verb is:

```
        { RUN     }
STOP    { literal }
```

If the word RUN is used, then all files which remain OPEN will be CLOSED automatically. Files ASSIGNED to DISK will be CLOSED WITH PURGE and all others will be CLOSED WITH RELEASE. All storage areas for the object program are returned to the MCP and the job is then removed from the MCP mix.

The STOP RUN is not used for temporary stops within a program. STOP RUN must be the last statement of the program execution sequence.

If the literal option is used, the literal will be DISPLAYed on the message printer and the program will be suspended. When the operator enters the MCP continuation message mix-index AX, program execution resumes with the next sequential operation. This option is normally used for operational halts to cause the system's operator to physically accomplish an external action.

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

SUBTRACT.
The function of this verb is to subtract one, or the sum of two or more, numeric data items from another item, and set the value of an item equal to the result(s).

The construct of this verb has three options which are:

Option 1:

```
                  { literal-1   }  [{ literal-2   }       ]
SUBTRACT          { data-name-1 }  [{ data-name-2 } . . . ]  FROM

    data-name-m  [ROUNDED  [ data-name-n  [ROUNDED] . . . ]

    [ON  SIZE ERROR  any statement]
```

Option 2:

```
                  { literal-1   }  [{ literal-2   }       ]
SUBTRACT          { data-name-1 }  [{ data-name-2 } . . . ]  FROM

  { literal-m   }
  { data-name-m }       GIVING      data-name-n  [ROUNDED]

   [ ON SIZE  ERROR  any statement ]
```

Option 3:

```
                  { CORR          }
SUBTRACT          { CORRESPONDING }        data-name-1 FROM data-name-2

    [ROUNDED]      [ ON SIZE ERROR any statement ]
```

In Options 1 and 2, the data-names used must refer only to elementary numeric items. If Option 2 is used, the data-description of data-name-n may contain editing symbols, except when data-name-n also appears to the left of GIVING.

All rules specified under the ADD statement with respect to the operand size, presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, the editing results, the handling of intermediate results, and the CORR or CORRESPONDING option apply to the SUBTRACT statement.

When the GIVING option is not used, a literal may not be specified as the minuend.

When dealing with multiple subtrahends, the effect of the subtraction will be as if the subtrahends were first summed, and then the sum subtracted from the minuends.

USE.
The function of this verb is to specify procedures for any input/output error and/or label handling which are in addition to the standard procedures supplied by the MCP, to calculate the ACTUAL KEY for files assigned to DISK, and to accomplish various user required actions when a 12 punch (overflow) in the printer carriage control tape is encountered.

The construct of this verb has three options which are:

Option 1:

```
                                               ( file-name...  )
                                               ( INPUT         )
                                               ( OUTPUT        )
USE AFTER STANDARD ERROR PROCEDURE ON          { INPUT-OUTPUT  }
                                               ( I-O           )
                                               ( O-I           )
```

Option 2:

```
USE    { AFTER  }     STANDARD       { BEGINNING }
       { BEFORE }                    { ENDING    }

  [ { REEL } ]   LABEL PROCEDURE ON    { file-name... }
  [ { FILE } ]                         { INPUT        }      .
                                       { OUTPUT       }
```

Option 3:

```
USE    FOR KEY CONVERSION    ON file-name-1 [file-name-2...].
```

A USE statement, when present, must immediately follow a section header in the DECLARATIVE portion of the PROCEDURE DIVISION and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

If the file-name option is used as part of Option 2, the File Description entry for the file-name must not specify a LABEL RECORDS ARE OMITTED clause.

A USE statement specified for input and/or output files associated with the SORT verb will not be affected when executing the SORT unless an INPUT and/or OUTPUT PROCEDURE has been included in the program.

The USE statement itself is never executed rather, it defines the conditions calling for the execution of the USE procedures.

If neither REEL nor FILE is included in Option 2, the designated procedures are executed for both REEL and FILE labels. The REEL option is not applicable to mass storage files.

Within a given format, a file-name must not be referred to implicitly or explicitly in more than one USE statement.

USE procedures will be executed by the MCP:

a. After completing the standard I/O error retry routine (this applies only to option 1) the record in error has been read, thus another READ cannot appear in the USE section since the MCP is performing the section because of a previous READ which has been completed. Upon completion of the USE procedure, control is returned to the statement following the READ which detected the error condition. In the case of blocked or unblocked magnetic tape input, the tape will be sitting ready to read the next record as soon as the Option 1 procedure is completed.

b. The USE AFTER STANDARD BEGINNING clause designates that the procedure following the clause must be called upon to check data on input magnetic tape beginning-file-labels, or to insert data as an output magnetic tape beginning-file-label before it is written.

c. When the USE BEFORE STANDARD ENDING clause designates that a following procedure must be called upon to check user created data contained on input magnetic tape ending file labels or to insert data onto the user's portion of an output magnetic tape ending file label before it is written.

d. Prior to any SEEK WITH KEY CONVERSION statement on files named in the USE FOR KEY CONVERSION statement.

References to common label items need not be qualified by a file-name within a USE statement. A common label item is defined as being an elementary data item that appears in every magnetic tape beginning and/or ending file-label record, but does not appear in any data record of the program.

A common label item must have the same name, description, and relative position in every magnetic tape file-label record and may only be referenced while in a USE...LABEL PROCEDURE for that file.

If the INPUT or OUTPUT option is specified, the USE...LABEL PROCEDUREs do not apply when files are described as having LABEL RECORDS OMITTED.

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

Option 2 is not applicable to disk files.

WRITE.
The function of this verb is to release a logical record for an output file. It is also used to vertically position forms in the printer. For mass storage files, the WRITE statement also allows the performance of a specified imperative statement if the contents of the associated ACTUAL KEY item are found to be invalid.

The construct of this verb has two options which are:

Option 1:

```
    WRITE  record-name [FROM data-name-1]

                                   ( { integer-1   }                )
   [                               ( { data-name-2 }     LINES      )
   [   { AFTER  }     ADVANCING    (                                )
   [   { BEFORE }                  (                                )
   [                               (   TO CHANNEL   { integer-2   } )
                                   (                { data-name-3 } )

         AT   { END-OF-PAGE }  imperative-statement
              { EOP         }

                                                         ]
          TO   { ERROR                              }    ]
               { AUXILIARY                          }    ]
               { STACKER     { literal-1   }        }    ]
               {             { data-name-4 }        }    ]
```

Option 2:

```
    WRITE  record-name  [FROM data-name]

        [INVALID KEY any statement]
```

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

The record-name must be defined in the DATA DIVISION by means of an 01 level entry under the FD entry for the file. The record-name and data-name-1 must not be the same name, or be in two files that have the same record area.

The ADVANCING option allows the control of vertical positioning of each record on the printed page. The options are as follows:

a. When LINES is used, data-name-2 must be declared as PC 99 COMPUTATIONAL or integer-1 must be a positive integral value of 00 THRU 99.

b. WRITE BEFORE ADVANCING is more efficient than AFTER ADVANCING.

c. When CHANNEL is used, data-name-3 or integer-2 must possess a positive integral value of 01 ... 11. Data-name-3 must be declared as PC 99 COMPUTATIONAL. The MCP will advance the line printers carriage to the carriage control channel specified.

The END-OF-PAGE option applies to a file that has been assigned to a printer. When the END-OF-PAGE punch in the carrage control tape on the printer is detected, the END-OF-PAGE branch will occur.

Option 2 must be used for writing on disk files.

If the FROM option is specified, the data is moved from the area specified by data-name-1 in Option 1, and data-name in Option 2, to the output area, according to the rules specified for the MOVE statement without the CORR or CORRESPONDING option. After execution of the WRITE statement is completed, the information in the data-name following the word FROM is available, even though that record-name is not available.

When the WRITE statement is executed at object time, the logical record is released for output and is no longer available for referencing by the object program. Instead, the record area is ready to receive items for the next record to be written. If blocking is called for by the COBOL program, the records will be automatically blocked by the MCP.

Short blocks of records which were written during EOF or EOJ will be of no programmatic concern to the user when using the file as INPUT at a later period of time.

If a write error is detected during a magnetic tape write operation, the tape record in error will be erased and a rewrite will be attempted further down the tape until the record is finally written correctly. A punch or printer write error will result in a message to the operator. The COBOL programmer need not include any USE procedures to handle write errors.

The shortest allowable blocks which can be written on 7 and 9 channel magnetic tape units are 8 and 18 bytes respectively.

If a CLOSE statement has been executed for a file, any attempt to WRITE on the file until it is OPENed again will result in an error termination.

For files which are being accessed in a SEQUENTIAL manner, the INVALID KEY clause is executed when the end of the last segment of the file (last record) has been reached and another attempt is made to WRITE into the file. The last segment of a file is specified in the FILE-LIMITS clause or the FILE CONTAINS clause. Similarly, for files being accessed in a RANDOM manner, the INVALID KEY clause will be executed whenever the value of the ACTUAL KEY is outside the defined limits. An INVALID KEY entry must be specified when writing to a file described as containing FILE-LIMITS.

Records will be written onto DISK in either a SEQUENTIAL or RANDOM manner according to the rules given under ACCESS MODE. For RANDOM accessing, SEEK statements will be explicitly used for record determination as defined under ACCESS MODE, SEEK, and READ.

If the size and blocking of records being accessed in a RANDOM manner is such that a WRITE statement must place a record into the middle of a block without disturbing the other contents of the block, then an implicit SEEK will be given to load the block desired (if an explicit SEEK has not been given). If the file is being processed for INPUT/OUTPUT, then either an explicit or implicit SEEK for a READ statement will suffice to load the block between the READ and WRITE statements.

If the value of the ACTUAL KEY is changed after a SEEK statement has been given and prior to the WRITE statement, an implied SEEK will be performed and the WRITE will use the record area selected by the implied SEEK as the output record area. The value contained in the ACTUAL KEY will not be affected.

For RANDOM access, when records are unblocked, the use of a SEEK statement related exclusively to WRITE is unnecessary, and may result in an extra loading of the record from disk because the compiler is, in general, unable to distinguish between SEEK statements that are intended to be related to a READ and those intended to be related to a WRITE.

The card record being written will be selected to the ERROR or to the AUXILIARY stackers if indicated in the particular WRITE being executed.

ZIP.
The function of this verb is to cause the MCP to execute a control instruction contained within the operating object program.

The construct of this verb is:

```
    ZIP  data-name
```

Data-name (any level) must be assigned a value equivalent to the information contained in the MCP Control Card. VALUE is always ended with a period inside of the ending quote marks. ZIP may be used for programmatic scheduling of subordinate object programs contained in the Systems Program Library or to accomplish any of the "CC" MCP control functions as performed through the SPO or card reader.

In the statement ZIP TO-CALL-PGM2, the DATA DIVISION of the source program could contain the following entry:

```
01   TO-CALL-PGM2    PIC X(13), VALUE IS "EXECUTE PGM2."
```

The MCP will be called upon when the object program encounters the ZIP statement and will reference data-name (TO-CALL-PGM2 in the above example) to find out which control function is being called for. Using the above example, the MCP will schedule PGM2. When the time comes and the priority for PGM2 is recognized and memory space becomes available, the MCP will retrieve PGM2 from the program library and place it in the MIX for subsequent operation.
The program containing the ZIP verb will proceed to the next sequential instruction following the ZIP.

CODING THE PROCEDURE DIVISION.
Figure 5-2 illustrates the manner in which the PROCEDURE DIVISION can be coded.

BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMMER	DATE	IDENT 73 80

```
LINE NO   A       B                                                        Z
4     6 7 8    11 12                                                       72
01        PRØCEDURE DIVISIØN.
02        DISK-BACK SECTIØN.
03        ØPENER.
04            ØPEN INPUT DISK-ØUT, ØUTPUT PRINT-ØUT.
05            MØVE 1 TØ DISK-CØNTRØL.
06            PERFØRM HEADER.
07        READING.
08            READ DISK-ØUT INTØ DISK-PART.
09            ADD 1 TØ DISK-CØNTRØL.
10            MØVE RELEVANT TØ CARD-IMAGE.
11            IF FINAL-CARD = "ENDER" GØ TØ FINISH.
12            IF CØUNTER > 37 MØVE 1 TØ CØUNTER GØ TØ SKIPPER.
13            WRITE RECRD FRØM NEW-PRINT BEFØRE ADVANCING 02 LINES.
14            ADD 2 TØ CØUNTER GØ TØ READING.
15        SKIPPER.
16            WRITE RECRD FRØM NEW-PRINT.
17            PERFØRM HEADER, GØ TØ READING.
18        HEADER.
19            MØVE SPACES TØ RECRD WRITE RECRD BEFØRE ADVANCING CHANNEL 01.
20            WRITE PRINT FRØM TITLE BEFØRE ADVANCING 02 LINES.
21            MØVE 3 TØ CØUNTER.
22        FINISH.
23            CLØSE PRINT-ØUT DISK-ØUT.
24            STØP RUN.
25        END-ØF-JØB.
```

Figure 5-2. Coding of PROCEDURE DIVISION

SECTION 6

DATA COMMUNICATIONS

GENERAL.

This section deals with the COBOL constructs of the PROCEDURE DIVISION required to activate the data communications equipment as defined by the ASSIGN to hardware-name clause.

SPECIFIC VERB FORMATS.

NOTE

The specific verb formats are unavailable at this time.

SECTION 7

CODING FORM

GENERAL.

The coding form, which provides a standard method for describing COBOL source programs, has been defined by CODASYL, specifications and common usage. The COBOL Compiler accepts this standard coding format, but also allows certain departures from the standard, at the user's discretion.

The same coding form is used for all four divisions of the source program. The four divisions must appear in the following order: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Each division must be written according to the rules for the coding form.

The rules for spacing given in the following discussion of the coding form take precedence over all other rules for the coding form.

CODING FORM REPRESENTATION.

The coding format for a line is represented in figure 7-1. The digits designate columns.

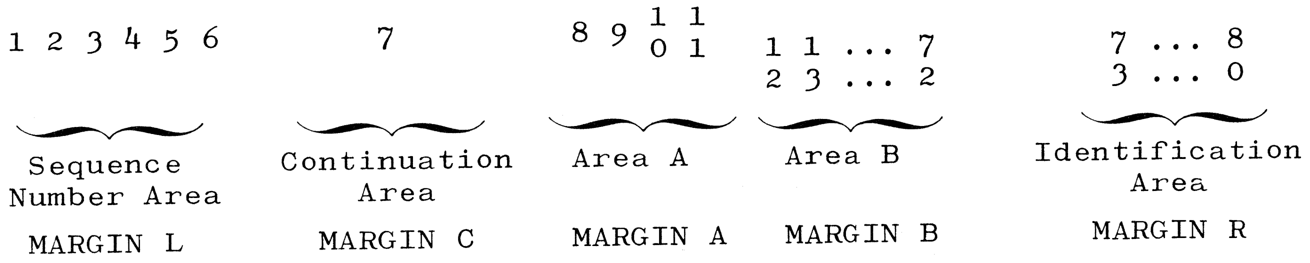


Figure 7-1. Coding Format for a Source Line

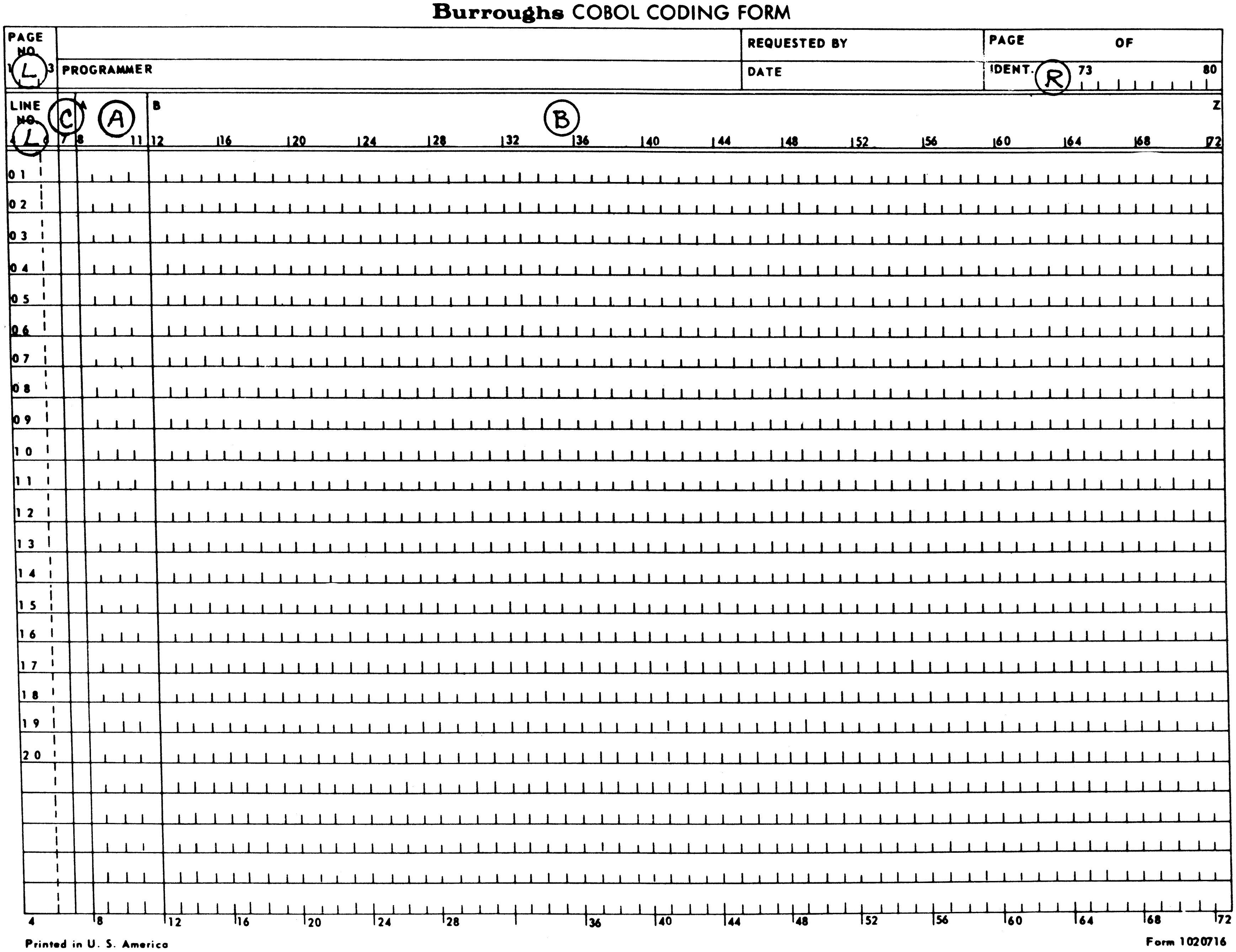
The COBOL Coding Form (see figure 7-2) is illustrated on the following page.

SEQUENCE NUMBERS (COLUMNS 1-6).

The sequence number field may be used to sequence the source program cards. Normally, numeric sequence numbers are used; however, the COBOL Compiler allows any combination of characters from the allowable character set. The compiler generates a warning message during compilation time if a sequence error (other than ascending) occurs.

CONTINUATION INDICATOR (COLUMN 7).

A hyphen in the Continuation Area of the continuation line indicates that the first character in Area B is the continuation of a word or a literal from the previous line. If a hyphen does not occur in the Continuation Area, the word or literal starting in Area B is not a continuation of an entry which started on the previous line and is separated from the previous entry with a space.



Burroughs COBOL CODING FORM

PAGE NO. 1 3 | PROGRAMMER | REQUESTED BY | DATE | PAGE OF | IDENT. 73 80

LINE NO. 4 6 | 7 | A 8 11 | B 12 | Z 72

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20

Printed in U. S. America

Form 1020716

L
L
C
A
B
R
16
20
24
28
32
36
40
44
48
52
56
60
64
68
4
8
12
16
20
24
28
36
40
44
48
52
56
60
64
68
72

Figure 7-2. COBOL Coding Form

An asterisk (*) indicates that the source line is for documentation purposes only and can appear anywhere within the source program. Continuation of following lines is denoted by an asterisk in column 7 of the continued data. All entries of this type are free form from Area A through Area B.

A slash (/) indicates that the source line is for documentation purposes only and that a skip to the head of a new page is required during the listing phase of the compiler output.

The letter L followed by a "library-name" entry, will cause all succeeding source card data to be placed into the COBOL Library File during compilation. Termination of the action takes place when an L Card is encountered followed by spaces.

CONTINUATION OF UNDIGIT LITERALS.

When an undigit literal is continued from one line to another, a hyphen is placed in the Continuation Area (column 7) of the continuation line, but the at sign (@) is not placed in the first character position of Area B (column 12). The continuation of the undigit literal commences in column 12 of Area B.

CONTINUATION OF NON-NUMERIC LITERALS.

When a non-numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area (column 7) of the continuation line and a quotation mark must be the first non blank position of Area B. The continuation of the non-numeric literal commences immediately following the quotation mark. All spaces at the end of the continued line and any spaces following the quotation mark of the continuation line and preceding the final quotation mark of the non-numeric literal are considered part of the literal.

CONTINUATION OF WORDS AND NUMERIC LITERALS.

When a word or numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area of the continuation line. This indicates that the first character of Area B of the continuation line is to follow the last non-blank character of the continued line without an intervening space.

Use of the continuation indicator for both non-numeric literals and other word entries is illustrated on the following page (see figure 7-3).

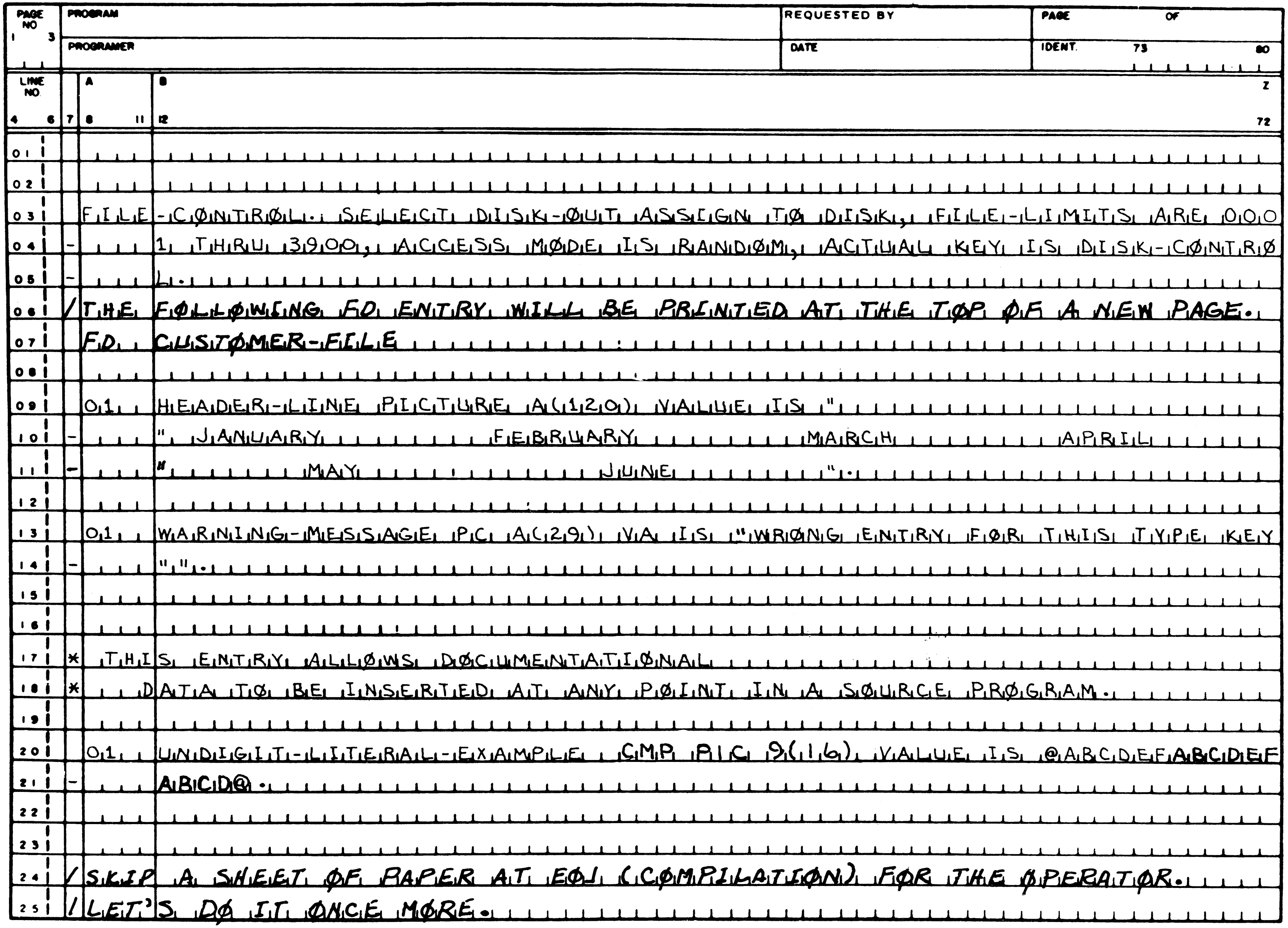
DIVISION HEADER.

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

SECTION HEADER.

The name of a section starts in Area A of any line except the first line of a division coding format. It is followed by a space, then

BURROUGHS COBOL CODING FORM



PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMER | DATE | IDENT. 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01
02
03  FILE-CØNTRØL. SELECT DISK-ØUT ASSIGN TØ DISK, FILE-LIMITS ARE 000
04 -    1 THRU 3900, ACCESS MØDE IS RANDØM, ACTUAL KEY IS DISK-CØNTRØ
05 -    L.
06 /THE FØLLØWING FD ENTRY WILL BE PRINTED AT THE TØP ØF A NEW PAGE.
07  FD  CUSTØMER-FILE
08
09  01  HEADER-LINE PICTURE A(120) VALUE IS "
10 -    " JANUARY         FEBRUARY         MARCH         APRIL
11 -    "       MAY             JUNE       ".
12
13  01  WARNING-MESSAGE PC A(29) VA IS "WRØNG ENTRY FØR THIS TYPE KEY
14 -    "".
15
16
17 * THIS ENTRY ALLØWS DØCUMENTATIØNAL
18 *   DATA TØ BE INSERTED AT ANY PØINT IN A SØURCE PRØGRAM.
19
20  01  UNDIGIT-LITERAL-EXAMPLE  CMP PIC 9(16) VALUE IS @ABCDEFABCDEF
21 -    ABCD@.
22
23
24 /SKIP A SHEET ØF PAPER AT EØJ (CØMPILATIØN) FØR THE ØPERATØR.
25 /LET'S DØ IT ØNCE MØRE.
```

Figure 7-3. Sample Coding Showing Continuation of Lines, Special Remarks, and Actions

the word SECTION, and then a period. In the PROCEDURE DIVISION, an option may be exercised by which the word SECTION would be followed by a space followed by a priority number. As above, the priority number would be followed by a period. No other text may appear on the same line as the Section Header except in the declarative portion of the PROCEDURE DIVISION. In this case, the USE and COPY sentences may begin on the same line as the Section Header. A section consists of paragraphs in the ENVIRONMENT and PROCEDURE DIVISIONs and data description entries in the DATA DIVISION. Paragraph names, but no section names, are permitted in the IDENTIFICATION DIVISION.

PARAGRAPH NAMES AND PARAGRAPHS.
The name of a paragraph starts in Area A of any line following the first line of a division coding format and ends with a period. A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins in Area B of either the same line as the paragraph name or any succeeding line. Successive sentences either begin in Area B of the same line as the preceding sentence or in Area B of the next line. A sentence consists of one or more statements, followed by a period.

DATA DIVISION ENTRIES.
Each DATA DIVISION entry begins with a level indicator or a level number, followed by one or more spaces, followed by the name of a data item, followed by a sequence of independent clauses described in the DATA DIVISION. Each clause, except the last clause of an entry, may be terminated by a semicolon or comma. This last clause is always terminated by a period.

There are two types of DATA DIVISION entries: those which begin with a level indicator and those which begin with a level number. A level indicator is an FD. In those DATA DIVISION entries which begin with the level indicator FD, the level indicator begins in Area A followed by a space and then by its associated file name and appropriate descriptive information and terminated with a period.

DATA DIVISION entries that begin with level numbers are called data description entries. A level number may be one of the following set: 01 through 49, 66, 77, and 88. Level numbers are written either as a space followed by a digit or a zero followed by a digit. At least one space must separate the level number from the word that follows it.

Level numbers 01, 66 and 77 should be coded in Area A. Other level numbers should be coded in Area B. Each successively higher level number should be indented four positions. This makes the coding easier to follow, and structure is readily apparent. Using odd numbered level numbers permits easy patching of record descriptions.

Coding repetitive information in the same columns makes keypunching easier; such as, PIC in columns 36-37, VALUE columns 52-56.

For example:

```
01  INPUT-RECORD.
    03  AMOUNT              PC 9(5)V99.
    03  AMOUNT-OUT          PC 9(5)V99.
    03  FACTORS             PC 9(3)V9(5).
    03  PERCNT              PC V999.
    03  NAME-CITY           PC X(10).
    03  CODR                PC XX.
    03  DATER.
        05  MONTH           PC 99.
        05  DAY             PC 99.
        05  YEAR            PC 99.
            88  CUR-DECADE                 VA
                60 THRU 69.
    03  FILLER              PC X(33).
66  IN-DATE RENAMES MONTH THRU YEAR.
```

NOTE

The above 88 level is continued on the following line, however, a dash in column 7 must not appear, inasmuch as the compiler continues to scan the following line in an effort to satisfy the VAlue requirement.

DECLARATIVES.

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow the Declaratives portion of the PROCEDURE DIVISION, respectively, must each appear on a line by itself. Each must begin in Area A and be followed by a period.

PUNCTUATION.

The following rules of punctuation apply to the writing of COBOL programs for this system.

a. A sentence is terminated by a period. A period may not appear within a sentence unless it is within a non-numeric literal or is a decimal point in a numeric literal or is in a PICTURE.

b. Two or more names in a series must be separated by a space or a comma.

c. Semicolons are used for readability and are never required. The semicolon is used for separating statements within a sentence or clauses within data description entries.

d. The reserved word THEN is also used for readability and can be used to separate two statements within a sentence. It can also be used between the condition and the first statement within an IF statement. For example:

```
IF .... THEN .... THEN .... ELSE ....
```

e. A space must never be imbedded in a name; hyphens may be used instead. However, a hyphen may not start or terminate a name. For example:

PRODUCTION-PERIOD is a good data-name, section-name, or paragraph-name.

-PRODUCTION-PERIOD or -PRODUCTION-PERIOD- or PRODUCTION-PERIOD- are all bad entries.

SECTION 8

COBOL COMPILER CONTROL

GENERAL.

The COBOL Compiler, in conjunction with the Master Control Program, allows for various types of actions during compilation and is explained in the text that follows.

COMPILATION CARD DECK.

Control of the COBOL Source Language input is dervied from presenting the Compilation Card Deck, illustrated in figure 8-1, to the MCP.

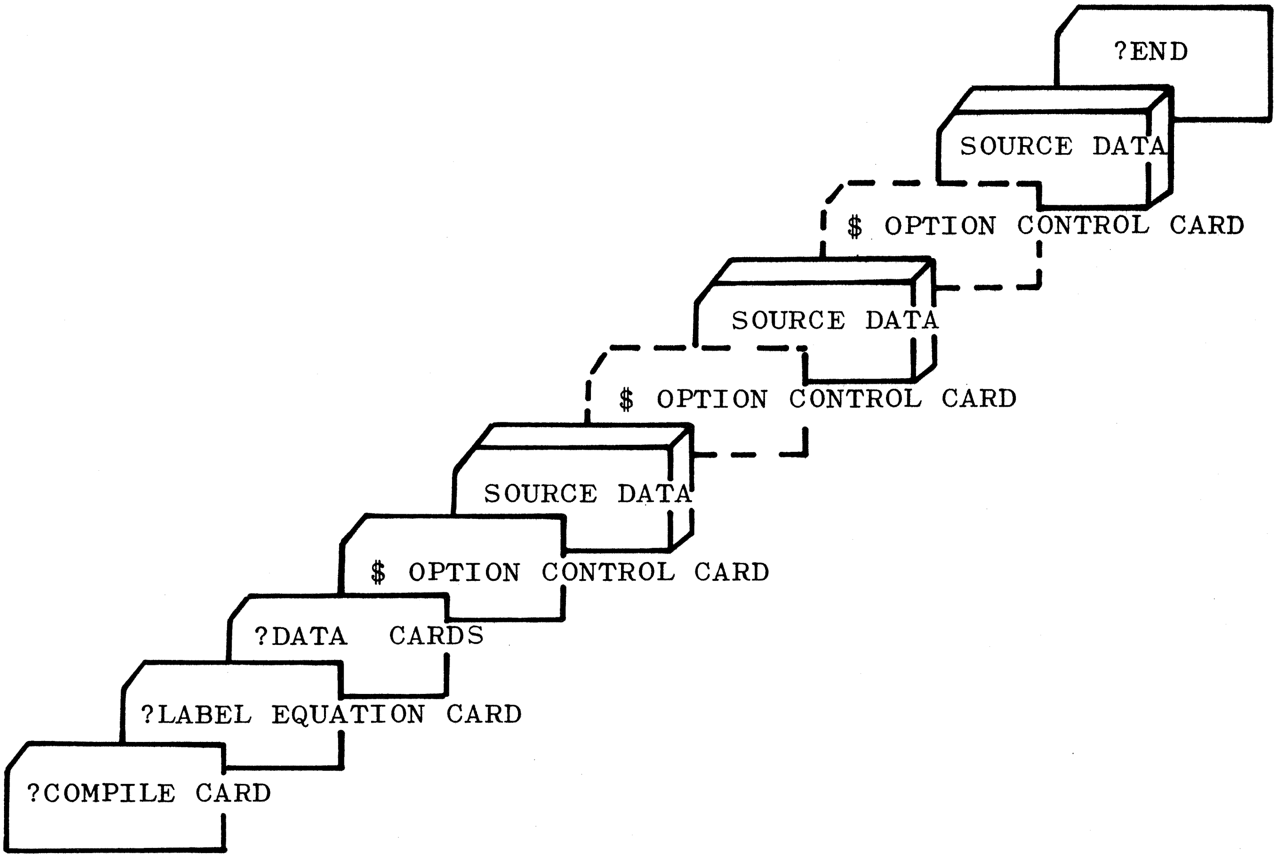


Figure 8-1. Compilation Card Deck

The Compilation Card Deck is comprised of several cards; these cards along with a detailed discussion of their function are presented in the paragraphs that follow.

?COMPILE CARD.

The first input control card instructs the MCP to call-out the COBOL Compiler and to compile the indicated program-name (P-N) using one of the following options:

a. To compile and run the resultant object program, the card is coded:

```
?COMPILE P-N WITH COBOL
```

b. To compile for a syntax check only, the card is coded:

```
?COMPILE P-N WITH COBOL SYNTAX
```

c. To compile and place the resultant object code into the Systems Library, the card is coded:

```
?COMPILE P-N WITH COBOL LIBRARY
```

d. To compile and place the resultant object code into the Systems Library, and then run the object program, the card is coded:

```
?COMPILE P-N WITH COBOL SAVE
```

NOTE

The word WITH is for readability only and may be excluded from the above statements.

The absence of the ?COMPILE Card will cause the System Operator to manually execute one of the above options through the SPO, using the MCP's CC notation in place of the invalid character (?).

MCP LABEL CARD.

The second control card, excluding Label Equation Cards, is the MCP LABEL Card and is formatted in the following form:

?DATA CARDS (indicates EBCDIC or BCD source language input).

The absence of the MCP LABEL Card will cause the message,

```
**NO FILE file-name program-name = mix-index
```

to be displayed on the SPO. The System Operator will not know the proper IL message to give the MCP (because of the options involved), without being given specific instructions by the programmer.

$ OPTION CONTROL CARD.

The third card, excluding Label Equation Cards, is the COBOL Compiler Option Control Card ($ sign in column 7). This card is used to notify the compiler as to which options are required during the compilation. If this card is omitted, $ CARD LIST CHECK SINGLE will be assumed. There must be at least one space between each item on the control card. The options may be in any order. Columns 1 thru 6 of the $-Card are used for sequence numbers. Any number of $-Cards may be used and may appear anywhere in the source deck. The options specified will become either active or inactive from that point on. The format of the Compiler Option Control Card is as follows:

```
$ option  option  ...
```

The options available for the COBOL Compiler Option Control cards are as follows:

a. CARD - input is from the Source Language Cards or paper tape. This option is for documentation only.

b. LIST - creates a single-spaced output listing of the source language input, with error and/or warning messages, where required.

c. SINGLE - causes the output listing to be printed in a single-spaced format.

d. DOUBLE - causes the output listing to be printed in a double-spaced format.

e. CODE - list object code following each line of source code from the point of insertion.

f. HEX CODE - all addresses on the CODE listing will be in Hexadecimal format. If this option is omitted the addresses will be in decimal format.

g. MERGE - primary input is from a source other than a card reader and may be merged with a patch deck in the card reader. It is assumed to be from a disk file, with a file-ID of COBOLW/SOURCE, by default. If it is desirable to change the input file-ID or change the input device from disk to tape, a LABEL EQUATION CARD must be used. The NEW option may be used with the MERGE option to create a new output source file plus changes.

h. NEW - creates a NEW output source file with changes, if any, entered through the use of the MERGE option, but does not include the Compiler Option Cards, if any, which must be merged in from the card reader when compiling from disk or tape. The output file will be created on disk by default with the file-ID of COBOLW/SOURCE. If it is desirable to change the output file-ID device from disk to tape, a LABEL EQUATION CARD must be used.

i. CHECK - this option will cause the compiler to check for sequence errors and print a warning message for each sequence error. The CHECK option is set on by default at the beginning of each compile, but may be terminated with the NO option.

j. SUPPRESS - suppresses all warning messages except sequence error messages. The sequence error message can be suppressed with the NO CHECK option.

k. SPEC - If syntax ERRORS occur this option negates the control and LIST option and causes only the syntax errors and associated source code to be printed. Otherwise the CONTROL and

LIST options remain in effect.

l. "Non-numeric literal" - is inserted in columns 73-80 of all following card images when creating a new source file and/or listing. This option can be turned off or changed by a subsequent control card with the area between the quote marks containing blank characters.

m. SEQ - starts re-sequencing, the output listing and the new source file if applicable, from the last sequence number read in and increments the sequence number by ten or by last increment presented in a previous $-Option Card. When re-sequencing starts at the beginning of the program source statements, the sequence will start with 000010.

n. SEQ nnnnnn - starts re-sequencing the output listing and new source file if applicable, from the sequence number specified by nnnnnn and increments the sequence numbers by ten.

o. SEQ +nnnnnn - starts re-sequencing the output listing and new source file if applicable, from the last sequence number read in and increments by the number specified by +nnnnnn. When re-sequencing starts at the beginning of the program source statements, the sequence will start with 000010.

p. SEQ nnnnnn+nnnnnn - starts re-sequencing the output listing and new source file if applicable, from the sequence number specified by nnnnnn and increments by the value of +nnnnnn.

q. NO SEQ - terminates the SEQ option and resumes using the sequence number in the source statement as it is read in.

r. CONTROL - prints the $ Option Control Cards on the output listing. The LIST option must be on.

s. NO - when the NO option precedes one of the above options with the exception of MERGE which cannot be terminated it will terminate the function of that option.

The NEW option does not have to be included when operating with a tape or disk source input, thus allowing temporary source language alterations without creating a new source output file.

The MERGE option without the NEW option allows a disk or tape input file to be referenced and to have external source images included from the card reader on the output listing and in the object program. A new output file will not be created.

Columns 1-6 of the Compiler Option Control Card may be left blank when compiling from cards. A sequence number is required when compiling from tape or disk when the insertion of the $ option is requested within the source input.

SOURCE DATA CARD.

These cards follow the $ Option Control Cards. The following source cards are used to create an updated version of the source input file or cause temporary changes to the tape or disk source language input:

a. VOID nnnnnn Patch Card. The punch sequence number in card columns 1-6 is followed by a $ in column 7 then the word VOID. This will delete the source records from the sequence number in the first six positions of the VOID Card through the sequence number specified by nnnnnn. If "N" is left blank only the source record identified by the sequence number in the VOID Card will be deleted from the compilation and the output listing, tape or disk files.

b. Change or Addition Patch Card. Punch sequence number in card columns 1-6 and changed or added source language data in applicable card columns. These cards must be in the proper sequence for the source input file in order to be properly MERGED into that file.

The COBOL Compiler has the capability of merging inputs from two sources (punched cards or paper tape, either of which may be merged with magnetic tape or disk) on the basis of the sequence numbers.

When merging inputs, the output compilation listing will indicate all inserts and/or replacements.

All of the $ options may be inserted at any point within the source language input data. Once an option has been turned on it will remain on until turned off with the "NO" option in another $ Option Card. In the case of the non-numeric literal it must be turned off by coding a non-numeric literal with blanks.

LABEL EQUATION CARD.

This card may be used to change a compiler file-name in order to avoid duplication of file-names when operating in a multiprogramming environment.

The Label Equation Card must be used in conjunction with the MERGE and NEW options when the primary input or output is from magnetic tape, the input disk file does not have a file-ID of SOURCE, or when a file-ID other than SOURCE is desired for the new disk output file.

The format for the LABEL EQUATION CARD is:

```
?COBOL FILE internal file-name =
users choice of file-IDs, file-attributes ...
```

The Label Equation Card (or cards), if used, must immediately follow the ?COMPILE ... Control Card and precede the MCP LABEL Control Card (refer to figure 8-1).

The COBOL Compilers internal file-names and external file-IDs for use in label equation are as follows:

Internal File-Name	External File-ID	Description
CARDS	CARDS	Input file from the card reader. If $ MERGE is used this file will be merged with the input file on disk or tape. The default input is from the card reader.
SOURCE	COBOLW/SOURCE	Input file from disk or tape when the MERGE option is used. The default input is from disk.
NEWSOURCE	COBOLW/SOURCE	Output file to disk or tape for a NEW source file when the NEW option is used. The default output is to disk.
LINE	LINE	Source output listing to the line printer.

The following are some examples of the LABEL EQUATION uses.

Example 1:

To compile a COBOL Program from the card reader and create a copy of the source program blocked five on a disk file with the file-ID of COBOL/TEST1 the following Label Equation (FILE) Cards could be used:

```
? COMPILE P-N WITH COBOL SYNTAX
? COBOL FILE NEWSOURCE = COBOL/TEST1,DISK,BLOCK 5
? DATA CARDS
  $ CARD LIST DOUBLE NEW
... SOURCE PROGRAM DECK ...
? END
```

To create the same program file on magnetic tape use the following FILE Card.

```
? COBOL FILE NEWSOURCE = COBOL/TEST1,MAGTAPE,BLOCK 5
```

Example 2:

To compile a COBOL Program from a disk file which had been created by the default option of the $ NEW option and create a new source file on disk with the file-ID of TEST2 the following LABEL EQUATION Card could be used:

```
? COMPILE P-N WITH COBOL SYNTAX
? COBOL FILE NEWSOURCE = TEST2,DISK
? DATA CARDS
  $ MERGE NEW
... PATCH CARDS IF ANY ...
? END
```

If the input file had a file-ID of COBOL/TEST1 in place of of the default file-ID of SOURCE the following FILE Card should have also been used in the above example.

```
? COBOL FILE SOURCE = COBOL/TEST1,DISK
```

APPENDIX A

COBOL RESERVED WORDS(*3)

ABOUT(*4)
ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS(*4)
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
ALTERNATING
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATT-8A1
AUTHOR
AUXILIARY

B-500
B-1710
B-1712
B-2500
B-3500
B-9352
B-9353
BACKUP
BATCH-COUNT
BEFORE
BEGINNING
BLANK
BLOCK
BREAK
BY
BZ

CARD96
CF(*4)
CH(*4)
CHANNEL
CHARACTERS
CLOSE
CMP
CMP-1
CMP-3
COBOL
CODE(*4)
COLUMN(*4)
COMMA
COMP
COMP-1
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROL-1

[3] See special instructions, page 1-10.

[4] These reserved words may appear in a future compiler.

CONTROL-2
CONTROLS(*4)
COPY
CORR
CORRESPONDING
CURRENCY

DATA
DATE (SPECIAL REGISTER)
DATE-COMPILED
DATE-WRITTEN
DCT-2000
DE(*4)
DECIMAL-POINT
DECLARATIVES
DEMAND
DEPENDING
DESCENDING
DETAIL(*4)
DISC
DISK
DISK-PACK
DISPLAY
DISPLAY-UNIT
DIVIDE
DIVISION
DOWN

ELSE
END
ENDING
END-OF-JOB
END-TRANSIT
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILL
FILLER
FINAL(*4)
FIRST
FLOW
FOOTING(*4)
FOR
FORMAT
FORM
FROM

GENERATE(*4)
GIVING
GO
GREATER
GROUP(*4)

HEADING(*4)
HIGH-VALUE
HIGH-VALUES
HOLD(*4)

[4] These reserved words may appear in a future compiler.

IBM-1030
IBM-1050
ID
IDENTIFICATION
IF
IGNORE
I-O
I-O-CONTROL
IN
INDEX
INDEXED
INDICATE(*4)
INITIATE(*4)
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS

JS
JUST
JUSTIFIED

KEY

LABEL
LAST(*4)
LEADING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINE(*4)
LINE-COUNTER(*4)
LINES
LISTER
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MICR
MICR-EDIT
MICT-OCT
MOD
MODE
MODULES
MONITOR
MOVE
MFCU

NEGATIVE
NEXT
NO
NO-DATA
NO-ERRORS
NO-FORMAT
NON-STANDARD
NOT
NOT-READY
NOTE
NUMBER(*4)
NUMERIC

4 These reserved words may appear in a future compiler.

OBJECT-COMPUTER
OC
OCCURS
OCR
OF
OFF
O-I
O-L-BANKING
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER(*4)
PC
PERFORM
PF(*4)
PH(*4)
PIC
PICTURE
PLUS(*4)
POCKET-LIGHT
POLL
POSITION
POSITIVE
PRINTER
PRIORITY
PROCEDURE
PROCEED
PROCESS(*4)
PROCESSING
PROCESSOR
PROGRAM-ID
PT-PUNCH
PT-READER
PUNCH
PURGE

QUOTE
QUOTES

RANDOM
RD(*4)
READ
READER
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
REPLACING
REPORT(*4)
REPORTING(*4)
REPORTS(*4)
RERUN
RESET(*4)
RESERVE
RETURN

[4] These reserved words may appear in a future compiler.

REVERSED
REWIND
RF(*4)
RH(*4)
RIGHT
ROUNDED
RUN

SAME
SAVE
SAVE-FACTOR
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SENTINEL
SEQUENTIAL
SET
SIGN
SIGNED
SIZE
SORT
SORTER
SOURCE(*4)
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
SPO
STACKER
STANDARD
START-TEXT
START-FLOW
STATUS(*4)
STOP
STOP-FLOW
STREAM
SUBTRACT
SUM(*4)
SUPERVISOR
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SY
SYMBOLIC
SYNC
SYNCHRONIZED

TALLY
TALLYING
TAPE
TAPE-7
TAPE-9
TC-500
TERMINATE(*4)
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TODAYS-DATE (SPECIAL REGISTER)

[4] These reserved words may appear in a future compiler.

TONE
TOUCH-TONE
TRACE
TRANSLATION
TT-28
TWX
TYPE(*4)

UNIT(*4)
UNTIL
UP
UPON
USAGE
USASI
USE
USING

VA
VALUE
VARYING
VOICE

WAIT
WHEN
WITH
WORDS
WORK
WORKING-STORAGE
WRITE
WRITE-READ
WRITE-READ-TRANS
WRITE-TRANS-READ

ZERO
ZEROS
ZEROES
ZIP

[4] These reserved words may appear in a future compiler.

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: B 1700 SYSTEMS
COBOL Reference Manual

FORM: 1057197
DATE: 10-72

CHECK TYPE OF SUGGESTION:

□ ADDITION □ DELETION □ REVISION □ ERROR

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME ______________ DATE ______________
TITLE ______________
COMPANY ______________
ADDRESS ______________
____

cut along dotted line

STAPLE

FOLD DOWN SECOND FOLD DOWN

Postage
Will Be Paid
by
Addressee

No
Postage Stamp
Necessary
If Mailed in the
United States

BUSINESS REPLY MAIL
First Class Permit No. 817, Detroit, Mich. 48232

Burroughs Corporation
6071 Second Avenue
Detroit, Michigan 48232

attn: Sales Technical Services
Systems Documentation

FOLD UP FIRST FOLD UP

Wherever There's
Business There's Burroughs

Printed in U.S. America