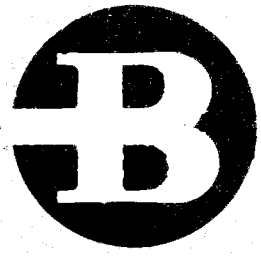


Burroughs L/TC

COBOL

REFERENCE MANUAL

Burroughs

L/TC

COBOL

REFERENCE MANUAL

$5.00

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

INTRODUCTION

This manual provides a complete description of COBOL (COMMON BUSINESS ORIENTED LANGUAGE) as implemented for use on the Burroughs Series L/TC Systems.

COBOL's advantages are derived chiefly from its intrinsic quality of permitting the programmer to state the problem solution in English. The programing language reads much like ordinary English prose, and can therefore provide its documentation automatically.

The Burroughs L/TC COBOL offers the following major advantages:

1. Accelerated program implementation.
2. Accelerated programmer training and simplified retraining requirements.
3. Significant ease of program modification.
4. Standardized documentation.
5. Self-documentation which facilitates non-technical management participation in data processing activities.
6. A comprehensive source program diagnostic capability.

It is assumed that the reader has a thorough understanding of the hardware of the Series L/TC systems. Knowledge of the General Purpose 300 (GP 300) Assembler Language is helpful although not necessary.

A program written in L/TC COBOL, called a source program, is accepted as input by the Series L/TC COBOL Compiler. The compiler verifies that all rules outlined in this manual are satisfied and generates the symbolic input to the Series L/TC Assembler. The Assembler, in turn, produces an object program capable of operating on a Series L/TC system. Program changes may be made either on the source (COBOL) or symbolic (GP 300) level. The source deck, therefore, reflects the operating object program.

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 memory allocation for those hardware devices.

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

The PROCEDURE DIVISION specifies the program steps necessary to accomplish the task desired by operating on data defined in the DATA DIVISION.

SECTION 1
CODING FORM

GENERAL

The coding form, which provides a standard method for describing and organizing COBOL source programs, has been defined by CODASYL specifications and common usage. The Series L/TC 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.

CODING FORM DESIGNATION

The coding format for a line is represented in Figure 1-1.

AREA	COLUMNS	BEGINNING MARGIN	COLUMN
Sequence Number	1-6	L	1
Continuation	7	C	7
A	8-11	A	8
B	12-72	B	12
Identification	73-80	R	73

Figure 1-1 Coding Format

MARGIN L

Margin L designates the leftmost character position of a line.

The Sequence Number Area occupies the first six character positions of a line beginning at Margin L. The sequence number field may be used to sequence the source program cards. Six-digit sequence numbers are usually used. Normally, the first three digits of the sequence number contain the Coding Form page number and the last three digits contain the line number on that particular page. Succeeding line numbers are usually incremented by 10 (e.g., the sequence number for the fifth line of page seven would be 007050). The compiler generates a warning message during compilation time if a sequence error (other than ascending) occurs. Other than providing this warning, the compiler does not use the sequence numbers. This is true only when compiling from a card source; when patching tape or disk, a sequence error is treated as a true error.

Burroughs COBOL CODING FORM

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

LINE NO. 4 6 | C 7 | A 8 11 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 | Z

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20

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

Printed in U. S. America

Form 1020716

Figure 1-2 A photographic reduction of Burroughs COBOL CODING FORM

MARGIN C

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

The Continuation Area occupies one character position beginning at Margin C. 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 on the previous line. If a hyphen does not appear, 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 or line by a space.

An asterisk (*) in column seven indicates that the source line is for documentational purposes only and can thus be used anywhere within the source program. Documentation may be continued by placing an asterisk (*) on each subsequent line or documentation. All entries of this type are free form from Area A through Area B.

MARGIN A

Margin A designates the eighth character position relative to Margin L. Area A occupies four character positions beginning at Margin A. (Columns 8-11, inclusive.)

The following items must begin in Area A (column 8):

1. Division – Name
2. Section – Name
3. Paragraph – Name
4. FD entry and other unindented entries
5. Record descriptions
6. Declaratives and End Declaratives

MARGIN B

Margin B designates the twelfth character position relative to Margin L. Area B occupies 61 character positions beginning at Margin B. (Columns 12-72, inclusive.)

The following items must begin in Area B (columns 12-72):

1. All sentences.
2. Continued sentences, words, or literals.
3. Any coding which has not been specified to begin in any other area.

MARGIN R

Margin R designates the rightmost character position of a line. The Identification Area occupies eight character positions beginning with Margin R leftwards (columns 73-80, inclusive). These columns may be used for any purpose required by the programmer, although they are normally used for card deck identification. Information in these columns will be ignored by the processor in creating the object program, although they will be printed on the compiler listing on the line corresponding to the card.

SECTION 2

COBOL LANGUAGE ELEMENTS

GENERAL

It has been stated that COBOL is a language based on English to the extent that the language is composed of words, statements, sentences, paragraphs, etc. This section defines the rules to be followed in the manipulation 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 Series L/TC COBOL character set consists of all characters on the L/TC print ball (Commercial and USASCII combined). See Character Set (Table A-5 in Appendix A).

CHARACTERS USED FOR WORDS

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

0-9

A-Z

- (hyphen or minus sign)

PUNCTUATION CHARACTERS

The following characters may be used for punctuation:

,	comma
.	period
;	semicolon

CHARACTERS USED IN EDITING

The L/TC COBOL Compiler accepts the following characters in editing (masking):

$	dollar sign
,	comma
.	decimal point
9	numeric digit
J	punctuation suppress
K	print compress
I	ignore digit
+	plus sign
-	minus sign
Z	zero suppress
P	punch zero suppress
X	alpha character

CHARACTERS USED IN RELATIONS

The L/TC COBOL Compiler accepts the following characters in conditional relationships:

= equal

< less than

> greater than

DEFINITION 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

L/TC COBOL contains three types of words. These word types are:

1. Nouns.
2. Verbs.
3. Reserved words.

NOUNS

Nouns are divided into nine special categories:

1. File-names
2. Record-names
3. Data-names
4. Procedure-names
5. Literals
6. Figurative constants
7. Special-names
8. Paragraph-names
9. Table-names

Since a noun is a word, its length may not exceed thirty characters, and must contain at least one alphabetic character. (Exception: See literals, page 2-3.) For purposes of readability, a noun may contain a hyphen, but 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 associated with a particular input-output device. 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 subdivided into several data items, each of which is distinguishable by a data-name.

Data-Name

A data-name is a noun used to refer to an item of data, or to a defined area containing the data. Some valid data-names are:

MONTHLY-EARNINGS

BALANCE-OF-PAYMENTS

K3457

Procedure-Name

A procedure-name is used to define the name of a subroutine to be PERFORMED. Procedure-names are declared only in the DECLARATIVES section.

Literal

A literal is an item of data whose value is identical to the characters contained within the item. There are two types of literals: numeric and non-numeric.

NUMERIC LITERAL. A numeric literal in the PROCEDURE DIVISION is an item composed of characters chosen from the digits 0 through 9, the plus sign (+) or minus sign (-). Numeric literals are also used in conjunction with VALUE statements in the DATA DIVISION; in this case the decimal point character is also allowable. The rules for formation of a numeric literal are:

1. Only one sign character and/or one decimal point may be contained in a numeric literal.
2. There must be at least one digit in a numeric literal.
3. The sign of the literal must appear as the leftmost character. If no sign is present, the literal is defined as positive.
4. The decimal point may appear anywhere within the literal except for the rightmost character of a numeric literal. Absence of a decimal point denotes an integer quantity.
5. A numeric literal cannot exceed 15 signed digits.

CONTINUATION OF A NUMERIC LITERAL. When a numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area (column 7) 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.

NON-NUMERIC LITERAL. A non-numeric literal may be composed of any print graphic character allowable for the L/TC. The beginning and end of a non-numeric literal is denoted by a quotation mark. All spaces and characters enclosed by quotation marks are considered part of the literal. A quotation mark may not be imbedded in a literal string; it must be handled separately with the figurative constant QUOTE. A non-numeric literal cannot exceed 99 characters. Examples of non-numeric literals are:

LITERAL ON SOURCE PROGRAM LEVEL	LITERAL STORED BY COMPILER
"ANNUAL DUES"	ANNUAL DUES
"-1234.789"	-1234.789
"QUOTE 98.6 QUOTE"	"98.6"

Note: Literals that are to be used for arithmetic computation must be expressed as numeric literals and must not be enclosed in quotation marks.

CONTINUATION OF NON-NUMERIC LITERAL. When a non-numeric literal is continued from one line to another, a hyphen is placed in column 7 of the continuation line and a quotation mark is placed in the first character position of Area B (column 12). The continuation of the non-numeric literal commences in column 13 of Area B. 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.

Figurative Constant

A figurative constant is a particular numeric or alpha value that has been assigned a fixed data-name. When a figurative constant is used to represent the corresponding value, it must never be enclosed in quotation marks. This does not preclude the use of a figurative constant as part of a non-numeric literal at which time it is enclosed in quotation marks. The figurative constant names and their meanings are:

ZERO ZEROS ZEROES	Represent the value 0 (or 00,000, etc.).
QUOTE	Represents "
ONE	Represents the value 1

Special-Names

The SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION facilitates association between certain hardware equipment names and problem-oriented mnemonics that the programmer may wish to use. SPECIAL-NAMES should be used for frequently referenced values such as print positions or line numbers.

Paragraph-Name

A PARAGRAPH-NAME is used to assign a label to a paragraph (defined on page 2-5). A PARAGRAPH-NAME consists of a word followed by a period and names the paragraph which immediately follows. This paragraph is terminated by the next PARAGRAPH-NAME.

Table-Name

A TABLE-NAME is used to define a Program Key table (PK table). It appears only in the DECLARATIVES section of the PROCEDURE DIVISION.

VERBS

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

RESERVED WORDS

The third type of COBOL word is the reserved word (includes verbs). 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. A complete list of reserved words in Series L/TC COBOL for the compiler is included in Appendix D. Reserved words are for syntactical purposes and can be divided into three categories:

1. Optional words
2. Key words
3. Connectives

Optional Word

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 Word

Another type of reserved word is the 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 preceding paragraph, the words IF and GREATER are key words.

Connective

The third type of reserved word is the connectives. Connectives are used to indicate the presence of a qualifier or to form compound conditional statements. AND, NOT and OR are used as logical connectives in conditional statements.

STATEMENT AND SENTENCE FORMATION

Statements are formed by the completion of the various verb constructs and other entries 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, also forms a sentence. An example of a sentence made up of three statements would be MOVE A TO B, ADD 1 TO COUNTER, DISPLAY SUMMARY UPON PRNTR.

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. On a coding form the name of a paragraph starts in Area A of any line following the Division Heading and ends with a period. The last statement of any paragraph must be terminated with a period. 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 somewhere in Area B of the same line as the preceding sentence or in Area B of the next line.

A paragraph is formed (i.e., a paragraph-name is assigned) in the PROCEDURE DIVISION whenever it is necessary that program control be transferred to a program segment by means of the GO TO verb (discussed later).

On a coding form the name of a section starts in Area A (at column 8) of any line except the same line as the Division Heading. It is followed by a space, then the word SECTION and then a period (e.g., WORKING-STORAGE SECTION.). Other text may appear on the same line as the Section Header. A section consists of data description entries in the DATA DIVISION.

NOTATION USED IN VERB AND OTHER ENTRY FORMATS

The notation conventions that follow enable the reader to interpret the Series L/TC COBOL syntax present 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 causes error conditions at compilation time. An example of key words is as follows:

IF data-name IS { ZERO / NEGATIVE }

The key words are: IF, ZERO, and NEGATIVE.

OPTIONAL WORDS

All upper case words not underlined are optional words and are 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 is a generic term in the following example:

OCCURS integer TIMES

BRACES

When words or phrases are enclosed in braces { }, a choice of one of the entries must be made. In the following example, either the word ZERO or NEGATIVE must be included.

IF data-name IS { ZERO / NEGATIVE }

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 the following example, the format enclosed in brackets is optional.

ENABLE [table-name] PK1 PK2 . . . .

CONSECUTIVE PERIODS

The presence of three consecutive periods (...) within any format indicates that the data immediately preceding the periods may be successively repeated, depending upon the requirements of problem solving. In the following example any or all of the conditions SW1, SW2, SW3, or SW4 may be tested.

IF { SW1 / SW2 / SW3 / SW4 } [OR { SW1 / SW2 / SW3 / SW4 } . . .]

In actual use, it might be:

IF SW2 OR SW3 OR SW4

PERIOD

When a single period is shown in the COBOL syntax discussed in this manual, it must appear in the same position whenever the source program calls for the use of that particular statement.

DATA NAMES

The normal convention of using "data-name" to indicate user defined items is expanded for the purpose of clarity as follows:

1. data-name. Refers to alpha or numeric data.
2. alpha-data-name. Refers to alpha data only.
3. numeric-data-name. Refers to numeric data only.
4. alpha-file-data-name. Refers to alpha data located only in a file.
5. numeric-file-data-name. Refers to numeric data located only in a file.

PUNCTUATION

The following rules of punctuation apply to the writing of Series L/TC COBOL programs:

1. A sentence is terminated by a period. A period may not appear within a sentence unless it is within a non-numeric literal (within " "s).
2. Two or more names in a series may be separated by a space or a comma.
3. 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. It is used the same way as a space or comma.
4. 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 ...
5. Data-names or paragraph-names must always be one continuous string of characters. A space must never be imbedded in a name; hyphens must be used instead. However, a hyphen may not start or terminate a name.

   Example:

   GRAND-TOTAL is a valid data-name or paragraph-name

   -GRAND-TOTAL or
   -GRAND-TOTAL- or
   GRAND-TOTAL- or
   GRAND TOTAL are all invalid entries.

SECTION 3
IDENTIFICATION DIVISION

GENERAL

The first part or DIVISION of the source program is the IDENTIFICATION DIVISION. This DIVISION identifies the source program and the output of a compilation. In addition, the date the program is written, the date of the compilation, and other information desired may be included in the IDENTIFICATION DIVISION.

The structure of this division is as follows:

IDENTIFICATION DIVISION.
[PROGRAM-ID. Any L/TC COBOL word.]
[AUTHOR. Any entry.]
[INSTALLATION. Any entry.]
[DATE-WRITTEN. Any entry.]
[DATE-COMPILED. Any entry – replaced by the current date as maintained by the MCP.]
[SECURITY. Any entry.]
[REMARKS. Any entry. Continuation lines must be coded in Area B of the coding form.]

```
4 - 6  7  8   11  12   16   20   24   28   32   36   40   44   48   52   56   60
01        IDENTIFICATION DIVISION.
02        PROGRAM-ID. ACTIVITY-REPORT.
03        AUTHOR. JOHN DOE.
04        DATE-WRITTEN. 12/10/69.
05        DATE-COMPILED.
06        SECURITY. COMPANY CONFIDENTIAL.
07        REMARKS. THIS PROGRAM ACCEPTS ACTIVITY DATA
08                 FROM PAPER TAPE OR KEYBOARD, PRINTS
09                 THE ACTIVITY REPORT, AND OPTIONALLY
10                 PUNCHES THE DATA ON PAPER TAPE.
```

Figure 3-1 Sample Coding for Identification Division

SYNTAX RULES

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

1. The IDENTIFICATION DIVISION must begin with the reserved words IDENTIFICATION DIVISION, followed by a period.
2. The entries must conform to the rules for the creation of words in Series L/TC COBOL.
3. The heading and paragraph-names must all begin at Margin A.

Notes:

When DATE-COMPILED is included, the compiler automatically inserts 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.

The first six characters of the PROGRAM-ID will appear in the heading of each page of the assembler listing, and in the summary information at the end of the compiler listing.

CODING THE IDENTIFICATION DIVISION

Figure 3-1 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.

SECTION 4

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 to the compiler information about storage files defined within the program. In addition, this DIVISION can also be used to specify the input-output procedures to be utilized. A coded example is provided by Figure 4-1 at the end of this section.

ORGANIZATION

The ENVIRONMENT DIVISION consists of two sections. The Configuration Section contains the overall specifications of the computer. The Input-Output Section deals with files to be used in the object program.

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:

1. The ENVIRONMENT DIVISION must begin with the reserved words ENVIRONMENT DIVISION followed by a period.
2. All paragraphs within the ENVIRONMENT DIVISION are optional and except for SPECIAL -NAMES, FILE-CONTROL, and I-O-CONTROL are for documentational purposes only. However, if a paragraph is used, its corresponding paragraph name and section name must also be included. For example, if a FILE-CONTROL entry is used, then INPUT-OUTPUT SECTION and FILE-CONTROL must appear first.

The specific definitions of the entries for the paragraphs shown are given below.

SOURCE-COMPUTER
OBJECT-COMPUTER
SPECIAL-NAMES

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:

```
[SOURCE-COMPUTER. B-3500.]
```

This paragraph is for documentation only. The actual machine style is optional; any entry may be made in this area.

OBJECT-COMPUTER

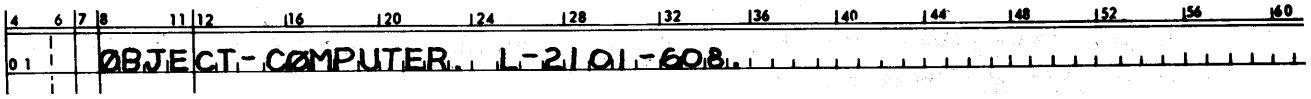
The function of this paragraph is to allow documentation of the configuration used for the object program.

The construct of this paragraph is:

```
[OBJECT-COMPUTER. {TC-500 }]
                  {TC-700 }  .
                  {L-2000 }
```

This paragraph is for documentation only. The actual machine style is optional; any entry may be made.

Example:



SPECIAL-NAMES

The function of this paragraph is to allow the programmer to assign a data-name to line numbers, print positions, and punch card columns which are frequently referenced. This allows simplicity for later changes to the program by changing only the single entry for the line number, print position, or card column, defined with the data-name, rather than all entries in the program using that reference.

The construct of this paragraph is:

```
[SPECIAL-NAMES.
    [data-name IS {POSITION} integer.]
                  {POS     }
    [data-name IS LINE integer.]
    [data-name IS {COLUMN} integer.]
                  {COL   }                ]
```

Example:

```
02
03  SPECIAL-NAMES.
04      SOLD-TO-LINE IS LINE 8.
```

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 specify the particular input-output hardware assignment, and to specify alternate work areas and buffers for input and output data.

The construct of this paragraph is:

```
[FILE-CONTROL.
    SELECT file-name ASSIGN TO {CARD-READER} {IN }
                               {DATA-COMM  } {OUT}
  [RESERVE ALTERNATE AREA] [ACCESS MODE IS SEQUENTIAL] {USE} WORK-AREA.]
                                                       {NO }
```

Note

The braces around DATA-COMM and IN and OUT are used to indicate that if DATA-COMM is ASSIGNED, either IN or OUT must be specified.

Files used in a program must be the subject of only one SELECT statement. That is, a given file-name may not be ASSIGNed to more than one hardware device. The nature of the Paper Tape Reader, Paper Tape Punch, and Card Punch peripheral devices exclude them from file control.

The RESERVE ALTERNATE AREA option will cause a working record area to be used for DATA-COMM IN or DATA-COMM OUT.

The ACCESS MODE IS SEQUENTIAL clause is used to specify that a DATA-COMM IN file will be accessed out of the data communications buffer (or alternate record area if the RESERVE ALTERNATE AREA clause was used) in a serial manner. This clause can be used only if the fields of the buffer are all fixed length. It will prevent the compiler from generating field pointers and therefore save code. The NO WORK-AREA clause must be used when using this option.

The NO WORK-AREA option is used to specify that the data will be accessed directly into and out of the data communications buffer or out of the punch card-in buffer.

The USE WORK-AREA option is used to specify that the input and output data will be accessed in individual main memory fields. The FILL verb (discussed in PROCEDURE DIVISION) is used to accomplish the transfer to the WORK-AREA. The WORK-AREA would be defined in the FILE SECTION of the DATA DIVISION.

Note

Only one file of a given hardware type may be SELECTed per program.

Example:

```
05
06      FILE-CONTROL.
07          SELECT LINE-DATA-IN ASSIGN TO DATA-COMM IN
08              ACCESS MODE IS SEQUENTIAL
09              NO WORK-AREA.

11
12          SELECT CARD-DATA-IN
13              ASSIGN TO CARD-READER
14              USE WORK-AREA.
```

The first example above ASSIGNS the file-name LINE-DATA-IN to the data communications message receive record area. The data will still be in character mode since the ACCESS MODE IS SEQUENTIAL and NO WORK-AREA clauses are used.

The second example ASSIGNS the file-name CARD-DATA-IN to the card reader buffer area in main memory. The card data will be moved to the WORK-AREA when the FILL verb is used in the PROCEDURE DIVISION.

I-O-CONTROL

The function of this paragraph is to specify the memory area to be shared by different files during processing.

```
[ I-O-CONTROL.
    [SAME AREA FOR file-name-1, file-name-2. ]
    [SAME WORK-AREA FOR file-name-1, file-name-2, . . . .] ]
```

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

The SAME AREA clause is used only to specify that DATA-COMM-IN and DATA-COMM-OUT alternate record area (declared by the RESERVE ALTERNATE AREA clause) will use the same main memory area.

The SAME WORK-AREA clause is used to specify that the work areas (declared by the USE WORK-AREA clause) will use the same main memory area. However, if two or more files share the same area, only one file may access the area at any given time, and processing of one record must be complete before attempting to use the area for another record from either file.

Note

The file requiring the largest memory area (work area) must be declared first in the DATA DIVISION.

CODING THE ENVIRONMENT DIVISION

An example of the ENVIRONMENT DIVISION coding is provided in Figure 4-1.

```
ENVIRØNMENT DIVISIØN.
CØNFIGURATIØN SECTIØN.
SØURCE-CØMPUTER. B-3500.
ØBJECT-CØMPUTER. L-2101.
SPECIAL-NAMES.
    TØTAL-PØS IS PØSITIØN 99.
    REMARKS-CØL IS CØLUMN 68.
    TØTAL-LINE IS LINE 35.
INPUT-ØUTPUT SECTIØN.
FILE-CØNTRØL.
    SELECT ØN-LINE-IN ASSIGN TØ DATA-CØMM IN
        RESERVE ALTERNATE AREA USE WØRK-AREA.
    SELECT ØN-LINE-ØUT ASSIGN TØ DATA-CØMM ØUT.
    SELECT CARD-DATA-IN ASSIGN TØ CARD-READER
        USE WØRK-AREA.
I-Ø-CØNTRØL.
    SAME AREA FØR ØN-LINE-IN, ØN-LINE-ØUT.
    SAME WØRK AREA FØR ØN-LINE-IN, ØN-LINE-ØUT,
        CARD-DATA-IN.
```

Figure 4-1 Sample Coding for Environment Division

SECTION 5
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:

1. Data which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.
2. Data which is developed internally and placed into intermediate or WORKING-STORAGE (defined below), or placed into specific format for output reporting purposes.
3. Constants which are defined by the programmer.

DATA DIVISION ORGANIZATION

The DATA DIVISION is subdivided into two sections:

1. The FILE SECTION which defines the contents of data files which are to be created or used by an external medium such as punch card. Each file is defined by a file description, followed by a record description or a series of record descriptions.
2. The WORKING-STORAGE SECTION which 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 . . . ] .
etc.
[FD file-name-2 . . . ] .
WORKING-STORAGE SECTION.
[77 data-name-3 . . . ] .
[77 data-name-4 . . . ] .
[01 record-name-2 . . . ] .
     [02 data-name-5 . . . ] .
     [02 data-name-6 . . . ] .
etc.
[01 record-name-3 . . . ] .
etc.
```

RECORD DESCRIPTION STRUCTURE

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

LEVEL-NUMBER CONCEPT

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for non-contiguous constants, and WORKING-STORAGE items. Each record of a file begins with the level-number 01. This number is used in the FILE-SECTION 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 go as high as 10. Figure 5-1 illustrates the use of level within a record.

Multiple level 01 entries of a given File Description of the FILE SECTION represent implicit redefinition of the same area. That is, both record-names will refer to the same file.

For an item to be elementary, it can not have subordinate levels. Therefore, the smallest element of a data description is called an elementary item. In Figure 5-1, MONTH, DAY, YEAR, MILLING, FINISHING, ITEM-NO, LOT-NO, STANDARD-COST, ASSEMBLY, INSPECTION, and WARRANTY-CODE are elementary items. A level that has further subdivisions is called a group item. In Figure 5-1, 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 5-1, group item PRODUCTION-CODE ends with INSPECTION. A group item can consist only of a level-number and a data-name followed by a period.

One additional level-number exists in Series L/TC COBOL. This number is 77 and is used for non-contiguous or single items within WORKING-STORAGE only. All 77 items must precede any other item.

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 10, plus the special number 77.

```
01  01  PRØDUCTIØN-RECØRD.
02      03  ITEM-NØ PICTURE 99999.
03      03  LØT-NØ PICTURE 999999.
04      03  ITEM-DATE.
05          05  MØNTH PICTURE 99.
06          05  DAY   PICTURE 99.
07          05  YEAR  PC 99.
08      03  STANDARD-CØST PICTURE ZZZZ.99.
09      03  PRØDUCTIØN-CØDE.
10          05  MACHINE-SHØP.
11              06  MILLING PICTURE 999.
12              06  FINISHING PIC  99.
13          05  ASSEMBLY PC 9999.
14          05  INSPECTIØN  PICTURE X(5).
15      03  WARRANTY-CØDE PIC 99.
```

Figure 5-1 Sample Coding for Data Division

FILE DESCRIPTION

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

The construct of this paragraph is:

FD file-name [DATA {RECORD / RECORDS} IS / ARE record-name-1, record-name-2, . . .] .

The level indicator FD identifies the beginning of a File Description and must precede the file-name. The DATA RECORD(S) clause is used for documentation only.

Only one period is allowed in the FD entry and it must follow the last used clause.

Example:

```
FD  INVENTORY-BAY-1
    DATA RECORDS ARE TUBES, DIODES, TRANSISTORS.
```

The above File Description example lists the record-names for INVENTORY-BAY-1. Following this sentence each record-name would be defined on an 01 level with whatever data-name breakdown is necessary for each. Additional 01 record descriptions for the records DIODES and TRANSISTORS would represent implicit redefines of the record area labeled TUBES. Since the File Description is for documentation only, the implicit redefinition is not accomplished until additional 01 record designations are made.

FORMAT

The function of this clause is to describe size and scaling factor for numeric input and arithmetic operations. It specifies the allowable number of digits to the right and to the left of the decimal point and provides for the enabling of the C, M, and RE keys.

The construct of this clause is:

{FMT / FORMAT} IS (any allowable format characters not to exceed 15 digits)

The FORMAT clause may appear in addition to the PICTURE clause and will determine the decimal alignment and keyboard entry factors. It may only be used with elementary items.

The FORMAT clause may be used in place of a PICTURE clause if printing of the item will not be required.

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

1. The letter S in the leftmost position indicate that the "RE" key may be used to enter signed data through the keyboard.
2. The letter V indicates the decimal position for decimal alignment and keyboard entry parameters.
3. The number 9 indicates a numeric digit.
4. The letters CM in the rightmost position indicates that the "C" or "M" key may be used when entering data through the keyboard.

Example:

```
    01  QUANTITY FORMAT IS S9999V99.
    01  PRICE FMT 9999V99CM.
    01  ID-NO FMT IS 999999.
```

In the example above, the first clause provides for the entry of 4 numbers to the left of the decimal, the decimal key, two numbers to the right and the RE key. The second allows the decimal key with 4 numbers to the left and 2 to the right as well as the C and M keys. The third clause allows only 6 digits to be entered on the numeric keyboard. The coding of the FORMAT clause may be shortened by the use of the 9 (integer) notation. The integer represents the number of numeric digits in a continuous string of digits.

Example:

```
4  6 7  8    11 12   16   20   24   28   32   36   40   44   48   52   56   60
01    01  FACTOR-1  FORMAT IS S99999999V999CM.
02    01  FACTOR-1 FMT S9(8)V999CM.
03
04    01  AMOUNT  FMT  S9(8).
05
06    01  FACTOR-2  FMT  9(6)V9(5).
```

In the above example FACTOR-1 has equivalent FORMAT in both statements.

The decimal alignment specified in the FORMAT determines the scaling factor when arithmetic is done with the data item.

OCCURS

The function of this clause is to define a sequence of contiguous data-items and to define a subscripted item (see page 5-9 for a discussion of subscripting).

The construct of this clause is:

{OC / OCCURS}	integer TIMES

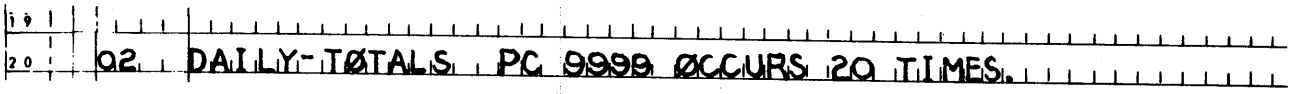
This clause cannot be used in any data description entry whose level-number is 01, and it can be used only with elementary items.

The integer may not be zero and must not exceed 256. The OCCURS clause may not appear with a VALUE clause or with an item which is subsequently REDEFINED.

Every item which is being defined by a given OCCURS clause will have the same FORMAT (discussed on page 5-3) or PICTURE (discussed on page 5-5).

The following illustrates the use of the OCCURS clause.

Example:



In the above example, a 20-word sequential storage area is defined, each of which has a format of 9999. An item in this area must be referenced by appropriately subscripting the data-name DAILY-TOTALS.

PICTURE

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

The construct of this clause is:

$$\left\{\begin{matrix}\underline{\text{PC}}\\ \underline{\text{PIC}}\\ \underline{\text{PICTURE}}\end{matrix}\right\}\ \text{IS (any allowable character string to describe the data).}$$

A maximum of 24 characters and symbols may be used to describe a numeric item, which cannot exceed 15 digits. An alpha item may not exceed 99 characters.

The PICTURE clause determines the print specifications. It also determines the decimal alignment, and keyboard entry factors unless a FORMAT clause is used in addition to the PICTURE clause in which case the FORMAT clause will determine the decimal alignment (for scaling) and keyboard entry factors.

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

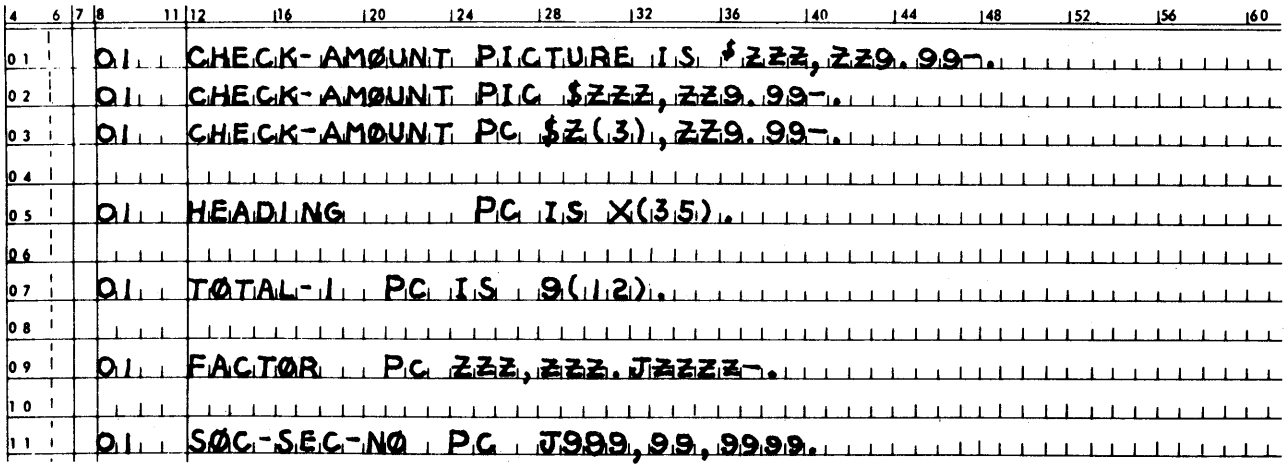
1. The letter J in the MSD position of a picture indicates suppress punctuation.
2. The letter J to the left of the decimal but not in the MSD position indicates reinitiate zero suppression and suppress for one digit (single digit zero suppress). The single digit zero suppress may be continued to the right with the letter Z for each additional character.
3. The letter J to the right of the decimal point indicates initiation of trailing zero suppression. Zero suppression is re-initiated and may be continued to the right with the letter Z for each additional character.
4. The letter K coded in the MSD position indicates "print compress" (TC 700 only).

5. The letter P coded in the MSD position indicates "punch zero suppress".
6. The letter X indicates a single alpha character.
7. The letter Z indicates zero suppression of number data.
8. The number 9 indicates numeric data with no zero suppression.
9. The special character $ indicates floating dollar protection.
10. The special character comma (,) indicates insertion of a comma.
11. The special character period (.) indicates insertion of a period and the decimal or scaling factor.
12. The special character plus (+) in the LSD position indicates print with the ribbon reversed if plus.
13. The special character minus (-) in the LSD position indicates print with the ribbon reversed if minus.
14. The character I indicates "Ignore digit".

Items 2, 3, 6, 7, and 8 above are counted in the length of the data item.

Alpha PICTURES may appear as X (integer) where the integer may have a value of 1-99. This is also true for numerics, 9 (integer). Likewise, the Z code can be Z (integer).

Example:



This example illustrates the coding of the PICTURE clause.

Example:

DATA	PICTURE	PRINTED OUTPUT
0000000377431334	J999,99,9999	377 43 1334
0000000008970045	ZZZJZZZ	897 045
NAME	XXXX	NAME
CUSTOMER	X(8)	CUSTOMER
0000000004891723	$ZZZ,ZZZ.99	$48,917.23
0000000000012345	999999	012345

A data-name may have both a PICTURE and a FORMAT as illustrated in the following example.

Example:

```
02
03
04   01  TØTALS FØRMAT IS 999999 PICTURE IS Z,ZZZ.99.
```

It should be noted that although both a PICTURE clause and a FORMAT clause may be used to determine decimal alignment and keyboard entry factors, the PICTURE clause may generate a print mask which will occupy one word of memory. Thus, if it is essential that memory space be conserved and printing of a numeric data item is not to take place, a FORMAT clause without a PICTURE clause is recommended. See Table 5-1 for the FORMAT and PICTURE combinations.

A PICTURE or a FORMAT or both must appear for every elementary numeric item level entry and cannot be used at group levels. Elementary alpha items must have a PICTURE and may not have a FORMAT.

Possible Combinations of Format and Picture Clauses

1. FORMAT – NO PICTURE
   a. Item is numeric.
   b. Scale factor from format.
   c. Input parameters from format.
   d. Item can be accepted (numeric keyboard).
   e. Item cannot be displayed.
2. FORMAT – ALPHA PICTURE
   a. Error.
3. FORMAT – NUMERIC PICTURE
   a. Item is numeric.
   b. Scale factor from format.
   c. Input parameters from format.
   d. Mask from picture.
   e. Item can be accepted (numeric keyboard).
   f. Item can be displayed.
4. NO FORMAT – ALPHA PICTURE
   a. Item is Alpha.
   b. Input parameters from picture.
   c. Item can be accepted (Alpha keyboard).
   d. Item can be displayed.
5. NO FORMAT – NUMERIC PICTURE
   a. Item is numeric.
   b. Scale factor from picture.
   c. Mask from picture.
   d. Item can be accepted.
   e. Item can be displayed.

Table 5-1

REDEFINES

The function of this clause is to allow an area of memory to be referred to by more than one data-name with the possibility of different FORMATS and PICTURES. Redefinition here is explicit as opposed to implicit redefinition of data-records in the FILE-SECTION (see page 5-3).

USE

The construct of this clause is:

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

The REDEFINES clause must be in the WORKING-STORAGE SECTION only.

The area being redefined (data-name-2) must be the previous item with the same level number (as data-name-1) which was not redefined.

The redefining name's (data-name-1) size must be equal to or less than the word size of the area which is being redefined (data-name-2).

There must be no VALUE clauses (discussed on page 5-9) within the redefined area.

Example:

```
06
07  01  TAX-CALCULATE PICTURE .9999.
08  01  TAX-PRINT REDEFINES TAX-CALCULATE PICTURE Z9.99.

10
11  01  ACTIVITIES.
12      02 ACT-1 PIC X(8) VALUE "ASSEMBLE".
13      02 ACT-2 PIC X(8) VALUE "PROCESS ".
14      02 ACT-3 PIC X(8) VALUE "TEST    ".
15  01  ACTIVITIES-TABLE REDEFINES ACTIVITIES.
16      02 ACT-LIST PICTURE X(8) OCCURS 3 TIMES.
```

In the first example above, the REDEFINES is used to provide for a printing decimal alignment different from the arithmetic scaling alignment. In the second example, ACTIVITIES and ACTIVITIES-TAB each define the same area of memory. The purpose of redefinition here is to allow a VALUE to be assigned to data which will be referenced (in PROCEDURE DIVISION) by subscripting.

USE

The function of this clause is to specify that a Data Comm field is of variable length. It also specifies the delimiter to use for Data Comm out.

The construct of this clause is:

```
[USE @ab@ FOR DELIMITER.]
 a, b may be 0 through F
```

The delimiter needed can be found on the USASCII Table in Appendix A. The "a" and "b" in the construct refer to the column number and row number from this table respectively. Since "b" may be only one digit, the row numbers 10-15 are designated with the letters A-F (10=A, 11=B, 12=C, 13=D, 14=E and 15=F).

This clause is optional. However, if it is used, the DELIMITER specified will be inserted after every field transferred to the transmit buffer. An alternative method of inserting DELIMITER codes is discussed with the MOVE verb in the Data Communications part of the PROCEDURE DIVISION (Section 6).

Example:

```
03
04    01   INVØICE-TØT  PC PZZZ,ZZZ.99-
05         USE @120 FØR DELIMITER
```

From the USASCII Table in Appendix A, it may be verified that the above clause will provide for the insertion of a DC2 (which corresponds to the K1 flag) after each field transferred to the transmit buffer.

VALUE

The function of this clause is to define the initial value of WORKING-STORAGE items.

The construct of this clause is:

$$\left\{\begin{array}{l}\underline{\text{VA}}\\ \underline{\text{VALUE}}\end{array}\right\} \text{ IS } \left\{\begin{array}{l}\text{up to 15 numeric digits}\\ \text{"up to 99 alpha characters}\\ \text{enclosed in quotes"}\end{array}\right\}$$

The VALUE clause must be only on elementary items and cannot be assigned to a data-name with an OCCURS clause.

Also the VALUE clause may not be used with the REDEFINES clause.

When defining a data-name, the PICTURE or FORMAT must be specified before the VALUE is assigned and the value must not exceed the size limitation specified by the PICTURE or FORMAT.

An alpha VALUE must be equal to the number of characters specified by the PICTURE.

It is important to note that, unless a VALUE is assigned, the contents of the memory area for any given data-name is unknown. That is, the memory area will be clear when the program is loaded only if a VALUE of 0 is assigned.
Example:

```
18
19    01   PAGE-DESCRIPTIØN PIC X(7) VALUE IS "PAGE NØ".
20    01   TAX-RATE PC Z9.99 VA 4.25.
```

SUBSCRIPTING

When a data-name is declared with an OCCURS clause, indicating an array, the particular element desired within the array is referred to by using subscripts. The subscripts follow the data-name

representing the array in a COBOL statement. A subscript may be either a numeric literal or a data-name and must be bounded by parentheses. A data-name being used as a subscript may not be subscripted and must define a numeric item. It is assumed to be right justified.

It may sometimes be necessary to assign data-names to each item in a group of data items for individual reference and then REDEFINE the group of data items using the OCCURS clause for subscripting.

When a data-name is subscripted, the subscript value must be greater than zero but not greater than the value shown in the corresponding OCCURS clause. The value subscripted will not be checked by the compiler.

When the option INDEXED BY appears in constructs discussed in the PROCEDURE DIVISION, the entry immediately following the INDEXED BY clause provides the subscripting integer value.

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. The example below shows how the OCCURS clause may be used in order to have the compiler build a table of twelve 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 that of subscripting (discussed above).

Example:

```
01    02  MONTHLY-TOTALS PIC ZZZ,ZZZ.99 OCCURS 12 TIMES.
```

The two-part example below will illustrate the use of tables and subscripting.

```
01  WORKING-STORAGE SECTION.
02  01  AMOUNTS.
03      02 AMOUNT PC ZZZ,ZZZ.99-
04         FORMAT IS S9(8), OCCURS 3 TIMES.
05
06  01  TOTALS.
07      02 TOTAL-1 PC 9(6), OCCURS 99 TIMES.
```

Part 1 above appears in the DATA DIVISION. The data-name AMOUNT appears with the OCCURS clause. In this case a table AMOUNT(1)) AMOUNT(2) AMOUNT (3) is set into memory. A 99 member table is developed for TOTAL-1.

Part 2 below illustrates the technique of subscripting. The value in AMOUNT(1) is added to the value in TOTAL-1 indexed by DISTRIBUTION-CODE. If DISTRIBUTION-CODE equals 25 then add AMOUNT(1) to TOTAL-1(25).

```
10
11      ADD AMOUNT(1) TO TOTAL-1(DISTRIBUTION-CODE).
12      DISPLAY TOTAL-1(COUNTER) UPON PRNTR.
```

FILLER

The reserved data-name FILLER is used by the programmer to declare areas which will never be directly referenced in the program.

Example: This example declares a table whose elements have an initial value of zero.

```
01   02 AREA-1.
02   03 FILLER  9(5)  VALUE IS 0.
03   03 FILLER  9(5)  VALUE IS 0.
04   03 FILLER  9(5)  VALUE IS 0.
05   02 TOTALS REDEFINES AREA-1  OCCURS 3 TIMES.
```

CHECK-DIGIT TABLE

This appears in the WORKING-STORAGE section. It functions to locate the CHECK-DIGIT TABLE in memory.

Example:

```
06
07   01  TABLE-NAME
08       02 CD-TABLE  999999999999999.
```

SECTION 6

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.

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, within the PROCEDURE DIVISION. The sentence structure is not governed by the rules of English grammar, but dictated by the rules and formats outlined in this manual.

STATEMENTS

There are two types of statements: imperative statements and conditional statements.

IMPERATIVE STATEMENTS

An imperative statement is any statement that is an unconditional execution command to the computer. The term imperative statement is also used to refer to a sequence of such statements, each separated from the next by a separator (period, comma, space, or semicolon). A single imperative statement is made up of a verb followed by its operand.

CONDITIONAL STATEMENTS

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is contingent upon this truth value.

SENTENCES

There are two types of sentences: imperative sentences and conditional 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 consists of one or more imperative statements 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.

Example:

```
ADD MONTHLY-SALES TO TOTAL SALES.
        or
SUBTRACT MINUSES FROM PLUSES, ADD NEW-PLUSES TO PLUSES THEN STOP RUN.
```

CONDITIONAL SENTENCES

A conditional sentence is a conditional statement terminated by a period. It may be optionally preceded by an imperative statement.

Example:

IF SALES EQUAL TO QUOTA THEN ADD NEW TO OLD, ELSE SUBTRACT NEW FROM MINUSES.

SENTENCE PUNCTUATION

The following rules apply to the punctuation of sentences:

1. A sentence is always terminated by a period.
2. A separator is a word or character used for the purpose of enhancing readability. The use of a separator is optional. The allowable separators are the semicolon(;), the comma (,), and the reserved word THEN.
3. Separators may be used in the following places:
   a. Between statements.
   b. In a conditional statement.
      (1) Between the condition and statement-1.
      (2) Between statement-1 and ELSE.
4. A separator must be followed by at least one space.

EXECUTION OF IMPERATIVE SENTENCES

An imperative sentence is executed in its entirety and control (sequence of program execution) is passed to the next applicable procedural sentence.

EXECUTION OF CONDITIONAL SENTENCES

In the conditional sentence:

IF condition THEN { statement-1 / NEXT SENTENCE } [ELSE 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 after statement-2.

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

statement-3 ELSE statement-4 ELSE statement-2.

If condition-1 is TRUE, imperative-statement-1 is executed. Then 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 the NEXT SENTENCE clause causes transfer of control to the next sentence written in order. However, if NEXT SENTENCE is used in the last sentence of a subroutine being PERFORMed (see PERFORM verb, page 6-29), control is transferred to the sentence in the mainline program to which control would otherwise be passed when the execution of the subroutine is complete.

PARAGRAPHS

So that the source programmer may group several sentences to convey one idea, paragraphs have been included in COBOL. The source programmer begins a paragraph with a name. The name consists of a COBOL word followed by a period, which precedes the paragraph it names. It is necessary that the statement which immediately precedes a paragraph name end in a period. 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 special paragraph-name END-OF-JOB which must be the last statement in the source program.

CONTROL RELATIONSHIP BETWEEN PARAGRAPHS

In COBOL, imperative and conditional sentences describe the function that is to be accomplished. The sentences are written successively, according to the rules stated in Section 2 to establish the sequence in which the object program is to execute. In the PROCEDURE DIVISION, names are used so that one paragraph can reference another by naming the paragraph 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 paragraph.

In executing paragraphs, control is transferred only to the beginning of a paragraph. Control is passed to a sentence within a paragraph only from the sentence written immediately preceding it. If a paragraph is named, control can be passed to it from any sentence which contains a GO TO, followed by the name of the paragraph to which control is to be transferred.

Example:

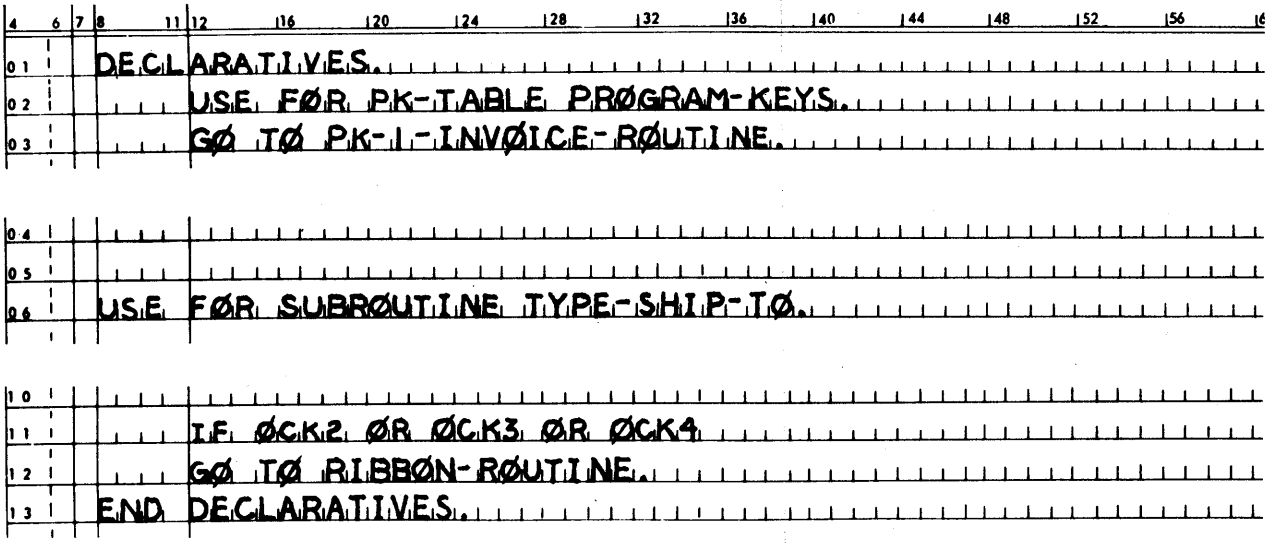
```
4   6 7 8   11 12      16      20      24      28      32      36      40      44      48      52      56      60
01             GØ TØ LINE-ITEM.
02         CLEAR-TØTALS.
03             MØVE ZERØS TØ GRAND-TØTALS.
04             GØ TØ LØAD-DATE.
05         TØTAL-ØUT.
06             ADVANCE LEFT TØ 5T.
07             PØSITIØN TØ 30.
08
```

DECLARATIVES

The DECLARATIVES part of the PROCEDURE DIVISION contains the subroutines and PK-Tables. 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 L/TC COBOL Compiler. These are the USE FOR PK-TABLE and USE FOR SUBROUTINE clauses (see page 6-32 for a complete discussion). In declaring the PK Tables and subroutines, the PK Tables must be declared first.

PROCEDURES

Paragraphs which are grouped together in order to accomplish a subroutine are termed a procedure.



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 statements. 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 6-1 indicates the relationships between the logical operators and conditions A and B.

Relationship of Conditions, Logical Operators, and Truth Values

Condition			Condition and Value	
A	B		A and B	A or B
TRUE	TRUE		TRUE	TRUE
TRUE	FALSE		FALSE	TRUE
FALSE	TRUE		FALSE	TRUE
FALSE	FALSE		FALSE	FALSE

Table 6-1

An example of the use of this table would be: Suppose in a program, a test is required to determine if the SW1 and SW2 flags are set. The following construct could possibly be used:

IF SW1 AND SW2 THEN ADD 1 TO COUNTER ELSE GO TO NUVAL.

In this case, 1 would be added to counter only if both switches are set. The phrase SW1 and SW2 (A and B column of table) is true only when both switches are set (that is, true).

RELATIONAL OPERATORS

The relational operators specify the type of comparison to be made in a relation condition (discussed below). The relational operators must be preceded by a space and followed by a space. The relational operators are:

1. IS EQUAL TO
2. IS =
3. IS NOT EQUAL TO
4. IS NOT =
5. IS GREATER THAN
6. IS >
7. IS LESS THAN
8. IS <

RELATION CONDITION

A relation condition causes comparison of two operands, each of which may be a data-name or a literal. The general format for a relation condition is as follows:

$$\left\{\begin{matrix}\text{data-name-1}\\ \text{literal-1}\end{matrix}\right\} \text{ relational-operator } \left\{\begin{matrix}\text{data-name-2}\\ \text{literal-2}\end{matrix}\right\}$$

The first operand, data-name-1 or literal-1, is called the subject of the condition. The second operand, data-name-2 or literal-2, is called the object of the condition.

COMPARISON OF OPERANDS

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

NON-NUMERIC

For non-numeric operands, a comparison will result in the determination that one operand is less than, equal to, or greater than the other with respect to a specified internal coding sequence of characters (see Table A-5 in Appendix A). The size of an operand is the total number of characters or digits in the operand. Relative tests of non-numeric operands require that their sizes (lengths) be the same. When being tested, characters or digits in corresponding character or digit positions of the two operands are compared starting from the high-order end through the low-order end. If all pairs of characters or digits compare equally through the last pair, the operands are considered equal when the low-order end is reached. The first pair of unequal characters or digits to be encountered is compared to determine their respective relationship. The operand that contains the character or digit that is positioned higher in the internal coding sequence (higher value in collating sequence) is considered to be the greater operand.

INTERNAL PROGRAM SWITCHES

Every program written in Series L/TC COBOL has up to eight general purpose (programable) switches (often called flags). Moving a numeric 1 to the switch will set it, and moving a numeric 0 to the switch will reset it. 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 command.

Example:

MOVE 0 TO SW4: Resets Switch 4

MOVE 1 TO SW2: Sets Switch 2

The switches will initially be reset (turned off) in the initialization routine of each program which is automatically provided by the compiler.

ARITHMETIC

SCALE FACTORS FOR DECIMAL ALIGNMENT

The verbs used for arithmetic are the ADD, SUBTRACT, MULTIPLY, and DIVIDE verbs. The respective PICTURES or FORMATS of the operands will automatically determine the manipulation necessary to provide the desired decimal alignment for the answer. If both PICTURE and FORMAT are present, the FORMAT will determine decimal alignment. For example, if a 2 decimal factor is multiplied by a 4 decimal factor and if the product must have 2 decimal places, the necessary shift command is provided by the compiler. However, if one of the operands is the ACCUMULATOR, the scale factor is unknown and the shifting must be provided by the source programmer.

ROUND

When it is desirable to have the result of an arithmetic operation rounded, the ROUNDED option may be used to accomplish this. This is a true arithmetic rounding, and is only applicable when the receiving field (result field) has fewer decimal places than the source fields.

SIZE ERROR

Whenever the magnitude of the calculated result exceeds 15 digits, a size error condition arises. The testing for the size error condition occurs only when the ON SIZE ERROR option has been specified. 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.

1. In the event that ON SIZE ERROR is not specified and size error conditions arise, the value of the resultant data-name is unpredictable.
2. 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 imperative statement associated with (immediately following) the ON SIZE ERROR option will be executed. If an error occurs when the ACCUMULATOR is one of the operands in an arithmetic statement with an "ON SIZE ERROR" clause, the ACCUMULATOR will contain undeterminable results.

SPECIAL HARDWARE NAMES

Some verbs contain certain Special Hardware Names. These special names and their meanings are:

KEYBOARD	REFERS TO ENTERING DATA THROUGH THE KEYBOARD, BUT NOT PRINTING.
KEYBOARD-PCH	REFERS TO ENTERING DATA THROUGH THE KEYBOARD, PUNCHING SAME, BUT NOT PRINTING.
KEYBOARD-PRNTR	REFERS TO ENTERING DATA THROUGH THE KEYBOARD AND PRINTING.
PCH	REFERS TO PUNCHING OUT DATA.
PRNTR	REFERS TO PRINTING OUT DATA.
PRNTR-PCH	REFERS TO PRINTING AND PUNCHING OUT DATA.
RDR	REFERS TO READING OF PAPER TAPE.
RDR-PCH	REFERS TO READING AND PUNCHING OF PAPER TAPE.
RDR-PRNTR	REFERS TO READING AND PRINTING FROM PAPER TAPE.
RDR-PRNTR-PCH	REFERS TO READING, PRINTING, AND PUNCHING OF PAPER TAPE.
KYBRD-PRNTR-PCH	REFERS TO ENTERING DATA THROUGH THE KEYBOARD, PRINTING, AND PUNCHING SAME.

Note: In cases where the hardware name is optional, the following is assumed if no hardware device is coded: KEYBOARD for the ACCEPT verb and PRNTR for the DISPLAY verb.

PROCEDURAL CONSTRUCTS

The specific verbs and other constructs, together with a detailed discussion of the restrictions and limitations associated with their use, appear on the following pages.

PART A: PROCESSING; PAPER TAPE I/O; 80-COLUMN CARD I/O

This part of the Procedural Constructs discusses the L/TC COBOL constructs used in processing and printing data, reading paper tape and 80-column card, and punching paper tape and 80-column card. Whenever paper tape is referenced, it is understood that the same applies to edge punched card.

Accept

The use of this construct is to allow entry of data through the keyboard or the paper tape reader. The basic function is to enter data into memory; however, through the use of special hardware names, the multi-purpose hardware instructions have been included.

The construct has four options:

OPTION 1

```
ACCEPT { data-name                          } [ FROM { KEYBOARD          } ]
       { INTO ACCUMULATOR data-name         }        { KEYBOARD-PRNTR    }
                                                     { KEYBOARD-PCH      }
                                                     { RDR               }
                                                     { RDR-PRNTR         }
                                                     { RDR-PCH           }
                                                     { RDR-PRNTR-PCH     }
                                                     { KYBRD-PRNTR-PCH   }
```

Option 1 is used to ACCEPT alpha or numeric data from one of the hardware devices into the ACCUMULATOR and store it in memory if specified. If the INTO ACCUMULATOR clause is used, it must be followed by the data-name associated with the appropriate input FORMAT. In this case only numeric data will be accepted.

Example:

```
06
07      ACCEPT STR-NUM FROM KEYBOARD-PRNTR.
08      ACCEPT INTO ACCUMULATOR INVOICE-NUM.
```

OPTION 2

```
ACCEPT integer CHARACTERS [ FROM { KEYBOARD-PRNTR    } ]
                                 { RDR-PRNTR         }
                                 { RDR-PRNTR-PCH     }
                                 { KYBRD-PRNTR-PCH   }
```

Option 2 is used to ACCEPT an integer number of alpha characters but not store it into memory. The integer may have any value between 1 and 99, inclusive.

Example:

```
10
11      ACCEPT 8 CHARACTERS FROM KYBRD-PRNTR-PCH.
12
```

In this example, a maximum of 8 alpha characters may be typed, printed, and punched.

OPTION 3

```
ACCEPT FROM KEYS
```

Option 3 is used to halt the machine so a PK or OCK selection may be made. No data (alpha or numeric) may be entered.

Example:

```
13
14      ENABLE PK1 PK4 PK6.
15      ACCEPT FROM KEYS
```

In the above example, the machine will halt with PK 1, 4, and 6 enabled.

OPTION 4

```
ACCEPT INTO ACCUMULATOR FROM { KEYBOARD }
                             { RDR      }
```

Option 4 is used to ACCEPT numeric data directly into the ACCUMULATOR when there is no data-name associated with the data.

Example:

```
17
18     ACCEPT INTØ ACCUMULATØR FRØM RDR
```

Note

When accepting a numeric data-name from the keyboard and the keyboard mode is to be terminated by depression of a PK key, care must be taken not to use a PK which is programed with a GO TO or a PERFORM. Incorrect results will occur unless the ACCUMULATOR is being addressed (as in Option 1 or 4). That is, the GO TO or PERFORM will cause program control to be transferred before the data is placed into the memory location specified by the data-name.

Add

The use of this construct is to add two numeric data items together. Refer to Page 6-6 for a discussion of arithmetic. The construct has three options: The ACCUMULATOR cannot be added to itself.

OPTION 1

```
ADD { data-name-1 } TO { data-name-2 } [ON SIZE ERROR statements]
    { integer     }    { ACCUMULATOR }
    { ACCUMULATOR }
```

If Option 1 is used, the two operands will be added together and stored in the second operand. See Page 6-6 for an explanation of ON SIZE ERROR.

Example:

```
4  6 7 8   11 12   16   20   24   28   32   36   40   44   48   52   56   60
01
02     ADD INV-TØT TØ GRAND-TØT ØN SIZE ERRØR
03       GØ TØ SHIFT-MØDE-1.
```

OPTION 2

```
ADD { data-name-1 } { data-name-2 } GIVING data-name-3
    { integer     } { ACCUMULATOR }
    { ACCUMULATOR }
[ROUNDED] [ON SIZE ERROR STATEMENTS]
```

ADVANCE

If Option 2 is used, the first two operands will be added together and stored in the third operand. ROUNDED generates correct results with Option 2 only when operand 2 has more decimal places than operand 3 as determined by the respective PICTUREs and/or FORMATs.

Example:

```
01           ADD 14 INCREM-NUM GIVING NEW-INCREM.
```

In both options, automatic decimal alignment is provided if the ACCUMULATOR is not one of the operands. See page 6-6 for a discussion of this.

OPTION 3

ADD digit TO ACCUMULATOR (integer) [ON SIZE ERROR statements]

If Option 3 is used, the "digit" will be added to the "integer" position of the ACCUMULATOR. For example, if the ACCUMULATOR reads 0000000000068345 and an ADD 3 TO ACCUMULATOR (6) instruction is used, the ACCUMULATOR will read 0000000000368345. The ACCUMULATOR digits are labeled, from right to left, as 1 through 16. Hence the "digit" may be one of 1-9 and the "integer" may be one of 1-16.

Example:

```
06           ADD 9 TØ ACCUMULATØR (12) ØN SIZE ERROR
07               ADD 5 TØ ACCUMULATØR.
```

Advance

The use of this construct is to space the forms in the carriage.

ADVANCE [{ LEFT / RIGHT / BOTH }] { TO special-name / TO integer LINE / integer LINES } [INDEXED BY { ACCUMULATOR / data-name }]

If the LEFT, RIGHT, or BOTH option is not used, ADVANCE LEFT is assumed. This construct has no effect on the ACCUMULATOR if the INDEXED BY option is not used. Special names are discussed in the ENVIRONMENT DIVISION (page 4-2).

Example:

```
10           ADVANCE 1 LINE.
11           ADVANCE RIGHT TØ LINE-PØS INDEXED BY ACCUMULATØR
```

Alarm

The use of this construct is to provide the ability to ring the alarm. The construct is:

```
ALARM
```

This construct has no effect on the ACCUMULATOR.

Close

The use of this construct is to provide for closing the forms transport. The construct is as follows:

```
CLOSE CARRIAGE
```

This construct has no effect on the ACCUMULATOR.

Convert

The use of this construct is to provide for the conversion of data to the Sterling monetary system, and to implement the check digit computation and verification commands. This construct has two options:

OPTION 1

```
CONVERT { ACCUM     } TO { SHILLING     }
        { data-name }    { FARTHING     }
                         { SPEC-FARTHING }
                         { POUNDS       }
                         { PENCE        }
                         { SPEC-PENCE   }
```

This construct will convert the data in either the accumulator or data-name as specified to the specified monetary unit.

Example:

```
1 2
1 3     CONVERT DØLLAR-DATA TØ PØUNDS.
```

OPTION 2

```
CONVERT { ACCUM data-name } TO data-name-2 CHECK-DIGIT [ (integer) ]
        { data-name-1     }
```

Option 2 of the CONVERT verb will calculate a check digit with the length obtained from the PICTURE clause to the "data-name" using the table located at the second "data-name" and using the "integer" as the remainder factor. No "integer" will result in an assumed zero remainder. The data-name-1 to be used for check digit computation should be right justified (it will be positioned automatically).

DISPLAY

Example:

```
03
04   COMPUTE ACCOUNT-NO TO CK-TABL-1 CHECK DIGIT.
```

In this example the check digit will automatically be calculated and inserted for ACCOUNT-NO using CK-TABL-1 with 0 remainder.

Display

The use of this construct is to provide for printing and/or punching of data from memory or I/O buffers. The basic function is to print unless specified otherwise by a hardware device.

This construct has six options:

OPTION 1

```
DISPLAY { data-name       } [FROM BUFFER] [ UPON { PRNTR      } ]
        { "literal"       }                       { PCH       }
        { ACCUMULATOR data-name }                 { PRNTR-PCH }
```

When a non-numeric literal is DISPLAYED, a maximum of 99 characters is allowed. If the ACCUMULATOR is DISPLAYED, the PICTURE of the specified data-name is used. The FROM BUFFER portion is valid with alpha data-names located in the card, or data communications input buffers. Data from the card buffer may be displayed on the PRNTR or the PRNTR-PCH. Data from the data communications input buffer may be displayed on the PRNTR only.

Example:

```
15
16   DISPLAY "SUBTOTAL" UPON PRNTR-PCH.
17   DISPLAY TOTAL.
18   DISPLAY REMARKS FROM BUFFER UPON PRNTR.
```

OPTION 2

```
DISPLAY { "character" } [ { PREVIOUS-RIBBON                          } ]
        { QUOTE       }   { { data-name   } { NEGATIVE } }
                          { { ACCUMULATOR } { POSITIVE } }
```

Option 2 will display upon the printer the specified character. If the PREVIOUS-RIBBON clause is used, the character will be printed using the same color ribbon as the last DISPLAY. If the other option is used, printing will occur only if data-name or the accumulator contents is negative or positive. To DISPLAY quotation marks, the figurative constant QUOTE must be used.

Example:

```
01 DISPLAY "/".
02 DISPLAY "*" PREVIOUS-RIBBON.
03 DISPLAY "-" ACCUMULATOR NEGATIVE.
04 DISPLAY QUOTE.
```

OPTION 3

DISPLAY ACCUMULATOR (integer) data-name [UPON { PRNTR / PCH / PRNTR-PCH }]

Option 3 will display the "integer" rightmost digits of the ACCUMULATOR according to the PICTURE for "data-name".

Example:

```
20 DISPLAY ACCUMULATOR (3) INV-NUM UPON PRNTR.
```

In this example only the low order 3 digits of the ACCUMULATOR will be DISPLAYED on the PRNTR using the PICTURE of INV-NUM.

OPTION 4

DISPLAY integer SPROCKET-HOLES [INDEXED BY PCH-REG]

Option 4 of the DISPLAY verb is usually used with edge-punched cards. If the "INDEXED BY PCH-REG" clause is not used, "integer" sprocket holes will be punched. If the "INDEXED BY" clause is used, "integer" sprocket holes minus the number contained in the punch register will be punched. Also, the use of the INDEXED BY clause will result in a routine being generated by the compiler to provide for the feeding out of a variable number of edge punched cards (provided the number of holes punched does not exceed 255). The size of the card is indicated by the integer value in the DISPLAY clause. That is, if an integer value of 70 is used, a routine will be generated to feed out the unpunched portion (70 minus PCH-REG value) of the last card in the string. There is a 3 card maximum for 7" cards and a 2 card maximum for 10" cards.

Example:

```
07 DISPLAY 5 SPROCKET-HOLES.
08 DISPLAY 70 SPROCKET HOLES INDEXED BY PCH-REG.
```

OPTION 5

DISPLAY @ab@ UPON PCH

Option 5 provides for punching into paper tape (or edge punched cards) the bit pattern specified by a and b contained between the @ signs. Any one of 128 possible characters may be punched. The parity channel must also be included in the bit pattern for the desired code to be punched, when the L/TC performs paper tape code translation during input. The USASCII table is given in Appendix A. In using

this table, "a" represents the column number and "b" represents the row number. Since "b" is only 1 digit in length, the row numbers 10-15 must be designated with its hexadecimal equivalent A-F (10=A, 11=B, 12=C, 13=D, 14=E, 15=F). Also included in Appendix A are tables showing the paper tape value of field identifier codes, accumulator flag codes, and codes for Switches 5, 6, 7, 8 (Y-flag group).

Example:

```
09
10    DISPLAY @12@ UPON PCH.
```

The example above will provide for the punching of the code for OCK 1 on paper tape. Reading the wide column of Table A-2 of Appendix A we find the appropriate row; that is, the row where there is a 1 under OCK 1 and zeros under OCK 2, 3, and 4. It is noticed that the code for setting the OCK 1 flag is DC2 and the paper tape value is 1,2. Thus, the @12@ in the example above. The DC2 code will also be found on the USASCII table (Table A-1) under column 1, row 2.

OPTION 6

```
DISPLAY { ACCUM     } { FARTHING     }
        { data-name } { HALF-PENNY   }
```

Option 6 provides for the printing of data in the specified unit of the Sterling monetary system.

Example:

```
11
12    DISPLAY ACCUM HALF-PENNY.
```

Divide

The use of this construct is to provide the ability to divide. The construct is as follows:

```
DIVIDE { data-name-1 } INTO { data-name-2  } [GIVING data-name-3]
       { integer     }      { ACCUMULATOR }
       [ROUNDED] [ON SIZE ERROR statements]
```

Without a "GIVING" option, this construct divides data-name-1 into data-name-2 storing the quotient in data-name-2. With a "GIVING" option the quotient is stored into data-name-3.

Automatic decimal alignment is provided if both operands are data-names.

When dividing "data-name INTO ACCUMULATOR" with an "ON SIZE ERROR" clause, the accumulator will contain undeterminable results if an error does occur.

Example:

```
06
07    DIVIDE HOURS INTO MILES
08          GIVING MPH ROUNDED.
```

A discussion of decimal alignment, ROUNDED, and SIZE ERROR may be found on page 6-6.

Enable

The use of this construct is to provide the ability to enable "PK" keys at the next keyboard operation.

The construct is as follows:

```
ENABLE [table-name] PK1  PK2 . . . . PK8  PK9 . . . . PK16
```

The use of the "table-name" is for activating multiple PK-tables. If only one table is defined, then it need not be coded since the compiler generated initialization routine activates the first table defined. If more than one table is defined, then the "table-name" option should be used at that point in the program when the appropriate table should be activated. The PK Table(s) must be defined in the DECLARATIVES.

This construct has no effect on the ACCUMULATOR.

PK 1 through PK 8 refer to program key group A (A1 through A8). PK 9 through PK 16 refer to program key group B (B1 through B8).

Example:

```
13
14          ENABLE TABL-2 PK8 PK12.
15          ACCEPT FROM KEYS
```

In this example the PK-Table TABL-2 is being referenced. PK 8 and PK 12 will be enabled and the machine will halt on a keyboard instruction. Note that TABL-2 need not be used in subsequent ENABLE clauses unless another table-name is used in between.

End-of-Job

The use of this construct is to notify the compiler that all source statements within a program have been read, and must be the last statement in every program.

The construct is as follows:

```
END-OF-JOB.
```

It does not affect the ACCUMULATOR.

Exit

The use of this construct is to provide the ability to programmatically transfer control from a subroutine instead of taking the automatic EXIT provided by the compiler.

The construct is as follows:

```
EXIT
```

FILL
GO TO
IF

Fill

The use of this construct is to cause the input file to be moved and reformatted into the fields of the designated record.

The format of this construct is as follows:

FILL record-name.

This verb must be used after the READ "file-name" if the USE WORK-AREA clause was used. The buffer may be interrogated prior to using the FILL verb so the appropriate "record-name" may be filled. For further explanation see page 4-4 for a discussion of the USE WORK-AREA clause. See page 6-29 for a discussion of READ.

Go To

The use of this construct is to provide an unconditional break in the sequence of program execution.

The construct is as follows:

GO TO paragraph-name.

The paragraph-name can be any paragraph name, but not a subroutine name. The paragraph could be within a subroutine. However, if execution falls through to the end of the subroutine, the automatic subroutine EXIT would be executed and incorrect results would appear.

This construct has no effect on the ACCUMULATOR.

Example:

```
16
17          GØ TØ TØTALS-RØUTINE.
```

If

The function of this construct is to control the sequence of commands to be executed depending on either a condition, the relative value of two items, of the "ON"-"OFF" condition of the various flags, switches, OCK's, and error conditions.

The conditions are subdivided into nine major categories:

1. Relative tests.
2. Accumulator tests.
3. Error condition tests.
4. Accumulator flag tests.
5. Switch tests.
6. OCK tests.
7. Check Digit test.

8. Sterling test.
9. TC-700 tests.

See the EXECUTION OF CONDITIONAL SENTENCES discussion (page 6-2).

RELATIVE TESTS. This test is used to compare two data-names or literals with regard to their numerical value.

The construct is as follows:

```
IF { data-name-1 } IS { GREATER THAN  }   { data-name-2 }
   { literal-1   }    {    >         }   { literal-2   }
                      { LESS THAN    }
                      {    <         }
                      { EQUAL TO     }
                      {    =         }
                      { NOT EQUAL TO }
                      { NOT =        }

     THEN { statements-1    } [ELSE statements-2].
          { NEXT SENTENCE   }
```

If the condition being tested is true, then the statements-1/NEXT SENTENCE option would be followed. If, however, the condition is not true and the ELSE option is in effect, control is transferred to statements-2. If the condition is not true and the ELSE option is not used, control is automatically transferred to the next sentence. For a discussion of the comparison of non-numeric literals, see page 6-5.

Example:

```
10
11      IF TOTAL IS < MIN-ORDER GO TO REGRET.
12      IF SCORE IS GREATER THAN MASTER NEXT SENTENCE
13          ELSE ADD 1 TO SCORE GO TO START.
```

In the first example, program control is transferred to REGRET only if TOTAL is less than MIN-ORDER. In the second example, the control is transferred to the next sentence only if SCORE is greater than MASTER. If SCORE is equal to MASTER, then 1 will be added to SCORE and control transferred to START.

ACCUMULATOR TESTS. These types of IF statements will test the contents of the ACCUMULATOR for various conditions.

This part of the construct has three options.

OPTION 1

```
IF ACCUMULATOR (integer-1) LESS THAN integer-2 THEN { statements     } [ELSE statements].
                                                    { NEXT SENTENCE }
```

Option 1 will determine if a specific digit of the data in the ACCUMULATOR (specified by integer-1) is less than integer-2 and transfer control accordingly. Integer-1 may be any number from 1 to 15, 1 being

the rightmost digit of the ACCUMULATOR and 15 specifying the leftmost digit of the ACCUMULATOR. Integer-2 may be any integer from 1 to 9.

Example:

```
18
19     IF ACCUMULATØR (3) LESS THAN 1 THEN
20       NEXT SENTENCE ELSE GØ TØ START.
```

OPTION 2

```
IF {data-name   } IS ZERO THEN {statements    } [ELSE statements] .
   {ACCUMULATOR}               {NEXT SENTENCE}
```

This option tests to see if the data-name or the ACCUMULATOR is zero and transfers control accordingly.

Example:

```
4  6 7 8    11 12   16   20   24   28   32   36   40   44   48   52   56   60
01     IF ACCUMULATØR IS ZERØ GØ TØ NUCARD.
```

OPTION 3

```
IF SIZE ERROR THEN statements [ELSE statements] .
```

Option 3 will test to see if a size error condition exists and transfer control accordingly. A discussion of size error may be found of page 6-6.

Example:

```
02
03     IF SIZE ERRØR THEN GØ TØ RECØVERY-RØUTINE.
```

ERROR CONDITION TESTS. This category of the IF verb tests for different error conditions that might exist.

This part of the construct has three options.

OPTION 1

```
IF {RDR-ERR } [{AND} {RDR-ERR }]
   {RDR-COND}  {OR } {RDR-COND}

     THEN {statements    } [ELSE statements] .
          {NEXT SENTENCE}
```

This clause is discussed in conjunction with Option 2 below.

OPTION 2

```
IF { PCH-ERR    }      [ { AND } PCH-ERR    ]
   { PUNCH-OFF  }        { OR  } PUNCH-OFF  ...
   { NO-MEDIA   }               NO-MEDIA
   { LOW-TAPE   }               LOW-TAPE   ]

   THEN { statements    } [ELSE statements] .
        { NEXT SENTENCE }
```

Because of the nature of paper tape, errors in the reading and punching of paper tape can occur. If an error occurs, a flag is set. The RDR-ERR and PCH-ERR options test to see if these flags are set and transfer control accordingly.

If a read or punch paper tape command is used and the tape reader or tape punch is not turned on, then a flag is set. The RDR-cond and PUNCH-OFF options test to see if these flags are set and transfer control accordingly.

When the output media specified in a program is turned off, a MEDIA flag is set. The NO-MEDIA option tests to see if the flag is set and transfers control accordingly. The flag is reset when the condition has been corrected.

When the punch paper tape is nearing depletion (approximately 20 feet of tape remaining), the Punch Tape Supply Flag is set. The LOW-TAPE option tests to see if this flag is set and transfers control accordingly. When the condition has been corrected, the next punch instruction causes the flag to be reset.

Example:

```
07
08      IF RDR-ERR THEN ACCEPT FROM KEYBOARD.
09      IF PUNCH-OFF OR NO-MEDIA OR LOW-TAPE
10          THEN ALARM.
```

OPTION 3

```
IF END-OF-PAGE THEN { statements    } [ELSE statements] .
                    { NEXT SENTENCE }
```

Option 3 will test to see if a forms limit flag has been set and will transfer control accordingly. Although this may not be an error condition, it functions similar to the other clauses of this part and therefore is included here.

ACCUMULATOR FLAG TESTS. This category of the IF verb tests if one or more of the four ACCUMULATOR flags are set. The flags are:

1. N Flag – Reverse Entry Flag
2. S Flag – Special Flag
3. C Flag – Per Hundred Flag
4. M Flag – Per Thousand Flag

```
IF {data-name    } {NFLAG} [{AND} {NFLAG} ...]
   {ACCUMULATOR  } {SFLAG}  {OR } {SFLAG}
                   {CFLAG}       {CFLAG}
                   {MFLAG}       {MFLAG}

THEN {statements    } [ELSE statements] .
     {NEXT SENTENCE }
```

Example:

```
11
12     IF ACCUMULATOR CFLAG OR MFLAG NEXT SENTENCE
13        ELSE GO TO SETUP.
```

SWITCH TESTS. This category of the IF verb tests to see if any of the eight internal program switches (flags) are "on" (set).

The constructs are as follows:

```
IF {SW1} [{AND} {SW1} ...] THEN {statements    } [ELSE statements] .
   {SW2}  {OR } {SW2}           {NEXT SENTENCE }
   {SW3}       {SW3}
   {SW4}       {SW4}
```

```
IF {SW5} [{AND} {SW5} ...] THEN {statements    } [ELSE statements] .
   {SW6}  {OR } {SW6}           {NEXT SENTENCE }
   {SW7}       {SW7}
   {SW8}       {SW8}
```

Example:

```
14
15     IF SW1 AND SW2 THEN ADD 1 TO COUNT
16        ELSE GO TO NAME-IN.
```

In the above example, the condition is true if both SW1 and SW2 are set.

OCK TESTS. This category of the IF verb enables the programmer to test if the OCK flags are set. When an OCK key is depressed (or code read from external media) a flag for that key is set and the flags for the other keys are reset. The depression of a "PK" key resets all OCK flags. It is possible to set multiple OCK flags using the MOVE verb explained on page 6-24. In any flag test AND and OR cannot be mixed in the same statement.

The construct of the OCK test is as follows:

```
IF {OCK1} [{AND} {OCK1} ...] THEN {statements    } [ELSE statements] .
   {OCK2}  {OR } {OCK2}           {NEXT SENTENCE }
   {OCK3}       {OCK3}
   {OCK4}       {OCK4}
```

Example:

```
17
18      IF ØCK1 THEN NEXT SENTENCE ELSE GØ TØ NAME-IN.
```

CHECK DIGIT TEST. This use of the IF verb will verify the check digit using the specified table.

The construct is as follows:

IF { ACCUM data-name / data-name } CHECK-DIGIT [(integer)] [FROM data-name] [TRUNCATED]

THEN statements [ELSE statements] .

This construct will verify the check digit using the table at the second data-name and the "integer" as the remainder factor. No "integer" will result in an assumed zero remainder. The data will be left in the original ACCUMULATOR positions unless the TRUNCATED clause is used, in which case the data will be shifted one place to the right for later arithmetic operations.

STERLING TEST. The use of this variation of the "IF" construct is to provide for the testing of keyboard entered data to see if the Sterling keys were used incorrectly.

The construct is as follows:

IF ACCUMULATOR IS { NON-DECIMAL / NON-STERLING } THEN statements [ELSE statements] .

Example:

```
4  6 7 8  11 12   16   20   24   28   32   36   40   44   48   52   56   60
01        IF ACCUMULATØR IS NØN-DECIMAL
02           THEN ALARM GØ TØ STER-CHECK.
```

Note:

When coding the "statements" following the "THEN" or the "ELSE" in an "IF" statement, the best code will be obtained if routines which are common only to the "IF" construct itself are coded as the "statements" rather than an unconditional branch (GO TO) to a routine which contains these statements.

Relative tests of alpha data names require that the sizes of the two items be the same.

The error condition tests and the switch test have no effect on the ACCUMULATOR.

MOVE

TC-700 FLAG TESTS. These tests are used to check the status of conditions particular to the TC-700.

The construct test whether or not the appropriate flag is set.

IF { TELLER-1 / TELLER-2 / SUPERVISOR / PB-FIRST-LINE / PB-LAST-LINE / PB-FOLD / PB-PRESENT } THEN statements [ELSE statements] .

Example:

```
01
02  IF TELLER-1 THEN GO TO TRANSACTIONS-ROUTINE
03      ELSE GO TO ERROR-ROUTINE.
```

Move

The use of this construct is to transfer data from one area of memory to another area, to load data into and clear memory locations, to set forms control limits and counts, to set or reset switches and flags, and to isolate parts of a word through shifting the ACCUMULATOR.

This construct has thirteen options.

OPTION 1

MOVE { data-name / integer / ZERO / ACCUMULATOR / "literal" } TO { data-name / ACCUMULATOR }

Option 1 is used to move data from operand 1 to operand 2. The contents of operand 1 (in the case of data-name and ACCUMULATOR) remain unchanged.

Example:

```
15
16      MOVE ZERO TO ACCUMULATOR.
```

This will cause the ACCUMULATOR to be cleared.

OPTION 2

MOVE digit TO ACCUMULATOR (integer)

Option 2 will move the "digit" (0-9) to the "integer" position of the ACCUMULATOR (1-15 from right to left).

Example:

```
03
04      MOVE 7 TO ACCUMULATOR (4).
```

If the initial contents of the ACCUMULATOR was 000000000012345, this clause will cause it to become 000000000017345.

OPTION 3

```
MOVE { special-name } TO [ { LEFT  } ] { LIMIT-REG  } [INDEXED BY { ACCUMULATOR } ]
     { integer      }     [ { RIGHT } ] { COUNT-REG  }             { data-name  }
```

Option 3 is used to load the forms control registers. The register will be loaded with the "special name", which must have previously been defined, or "integer". If the "INDEXED BY" option is used, the number in the ACCUMULATOR or data-name specified will be added to the value of the special name or integer and the sum will be moved to the specified register.

Example:

```
05
06      MOVE 3 TO LEFT LIMIT-REG INDEXED BY LIM-IND.
```

If LIM-IND contained 10, this clause will provide for a value of 13 being transferred to the left limit register.

OPTION 4

```
MOVE { 0    } TO { NFLAG } [ { NFLAG } ... ]
     { ZERO }    { SFLAG }   { SFLAG }
     { 1    }    { CFLAG }   { CFLAG }
     { ONE  }    { MFLAG }   { MFLAG }
```

Option 4 is used to set or reset the ACCUMULATOR flags (0 = reset; 1 = set). See Accumulator Flag Tests of the IF verb for a discussion of the accumulator flags (page 6-19).

Example:

```
07
08      MOVE ONE TO NFLAG CFLAG.
```

OPTION 5

```
MOVE { 0    } TO { SW1 } [ { SW1 } ... ]
     { ZERO }    { SW2 }   { SW2 }
     { 1    }    { SW3 }   { SW3 }
     { ONE  }    { SW4 }   { SW4 }
```

Option 5 is used to set or reset internal program switches SW1 through 4 only (0 = reset; 1 = set).

OPTION 6

```
MOVE { 0    } TO { SW5 } [ { SW5 } ... ]
     { ZERO }      { SW6 }   { SW6 }
     { 1    }      { SW7 }   { SW7 }
     { ONE  }      { SW8 }   { SW8 }
```

Option 6 is used to set or reset internal program switches SW5 through 8 only.

Example:

```
MOVE 0 TO SW5 SW7.
```

It is incorrect to combine switches from different groups (1-4, 5-8) in a single clause. For example, SW1 and SW6 may not appear in the same MOVE clause.

OPTION 7

```
MOVE { 0    } TO { OCK1 } [ { OCK1 } ... ]
     { ZERO }      { OCK2 }   { OCK2 }
     { 1    }      { OCK3 }   { OCK3 }
     { ONE  }      { OCK4 }   { OCK4 }
```

Option 7 is used to set and reset the OCK flags. (For a discussion of the OCK flags, see page 6-20.)

Example:

```
MOVE 1 TO OCK4.
```

OPTION 8

```
MOVE { 0    } TO { RDR-ERR }
     { ZERO }      { PCH-ERR }
```

Option 8 is used to reset the paper tape reader error and punch error flags. These flags are discussed under the IF construct (page 6-19).

Example:

```
MOVE ZERO TO RDR-ERR.
```

OPTION 9

```
MOVE ACCUMULATOR (integer-1 [integer-2] ) TO { data-name   [ (integer-3) ] } [WITH SIGN]
                                             { ACCUMULATOR }
```

Option 9 is used to isolate digits in the ACCUMULATOR as specified below and transfer the result to data-name if indicated.

1. If "integer-1" is used by itself, the computer will right justify the "integer-1" digit of the ACCUMULATOR.

Example:

```
15
16      MOVE ACCUMULATOR (5) TO PROD-CODE.
```

If the previous contents of the ACCUMULATOR was 000000000123456, the execution of the construct in this example would result in data-name PROD-CODE having the value 000000000000012.

2. If "integer-1" and "integer-3" are used, the computer will isolate the "integer-1" digit into the "integer-3" position of the ACCUMULATOR.

Example:

```
17
18      MOVE ACCUMULATOR (6) TO ACCUMULATOR (3).
```

If the previous contents of the ACCUMULATOR was 000000001234567, the execution of the construct in this example would cause the ACCUMULATOR to have the value 000000000000200.

3. If "integer-1" and "integer-2" are used, the computer will isolate and right justify "integer-2" digits of the ACCUMULATOR starting at the "integer-1" position.

Example:

```
04
05
06      MOVE ACCUMULATOR (8,2) TO ITEM-CODE.
07
```

If the previous contents of the ACCUMULATOR was 000000123456789, the execution of the construct in this example would cause data-name ITEM-CODE to have as its contents 000000000000023.

4. If all 3 integer options are used, the computer will isolate the first integer-2 digits to the right of and starting at integer-1 and position them starting at and to the right of integer-3.

Example:

```
08
09      MOVE ACCUMULATOR (1,05) TO ACCUMLATOR(7).
```

If the previous contents of the ACCUMULATOR was 003775076951456, the execution of the construct in this example would cause the accumulator to contain 000000005076900.

The ranges for all three integers is 1 through 15 unless the WITH SIGN option is used, in which case the range is 1 through 16.

OPTION 10

MOVE integer TO PCH-REG

Option 10 is used to load the Punch Count Register. The integer may be any value from 0 through 255.

Example:

```
19
20      MØVE 0 TØ PCH-REG.
```

OPTION 11

```
MOVE { ZERO  } TO group-name
     { ZEROES }
     { ZEROS  }
     { 0      }
```

Option 11 is used to clear all data items in the group specified by the group-name. The group name must be declared in the WORKING-STORAGE section. In order to conserve code generated by the compiler, this construct should not be used unless the group-name has 6 or more data-items within it.

OPTION 12

```
MOVE { 1    } TO PB-REQUIRED
     { ONE  }
     { 0    }
     { ZERO }
```

This construct applies to the TC-700.

OPTION 13

```
MOVE REMAINDER TO { ACCUMULATOR }
                  { data-name   }
```

Option 13 provides the programmer with the ability to use the REMAINDER resulting from the division process.

Multiply

The use of this construct is to multiply two items together.

The construct is as follows:

```
MULTIPLY { data-name-1 } BY { data-name-2  } [GIVING data-name-3] [ROUNDED]
         { literal     }    { ACCUMULATOR }

    [ON SIZE ERROR statements]
```

Without the "GIVING" option the product is stored in the second operand. With the "GIVING" option the product is stored in data-name-3.

Automatic decimal alignment is provided if both operands are data-names.

If a size error occurs, the first two data-name operands would be unchanged. However, if the ACCUMULATOR was the first operand, it would contain undeterminable results. A discussion of decimal alignment, ROUNDED, and SIZE ERROR is provided on page 6-6.

Example:

```
02
03     MULTIPLY TOTAL BY 3 GIVING FINAL-RESULT ROUNDED.
```

No-Op

The use of this construct is to provide the ability for the programmer to instruct the compiler to insert a "NO OPERATION" machine code.

The construct is as follows:

```
NO-OP.
```

Example:

```
04
05     USE FOR PK-TABLE PK-SELECT.
06     NO-OP
07     GO TO TOTAL-OUT.
```

As the example may imply, the most common use of the NO-OP is that of a "filler" in a PK Table in the DECLARATIVES.

Note

The use of this construct is to allow the programmer to write explanatory statements in his program which are printed on the program listing, but do not generate object code.

This construct has two options:

OPTION 1

```
NOTE sentence.
```

OPEN

Option 1 is NOTE followed by any comment and terminated by the next period.

Option 1 is coded anywhere in Area B of the coding form.

Example:

```
08
09          NOTE. THIS SENTENCE IS FOR DOCUMENTATION ONLY
10              AND MUST BE TERMINATED BY A PERIOD.
```

OPTION 2

NOTE. paragraph

Option 2 is NOTE followed by any comments or sentences or text. The documentation is terminated by the next paragraph name, and its coding must start in column 8 of the coding form.

Example:

```
11
12          NOTE. THIS ALLOWS THE INSERTION OF AN ENTIRE
13              PARAGRAPH. IT WILL END WITH THE NEXT
14              PARAGRAPH NAME.
```

Open

This construct is used to provide for opening the paper tape media clamp and for opening the forms transport.

The construct is as follows:

```
OPEN { MEDIA-CLAMP | CARRIAGE } [ { integer | special-name } ] [ INDEXED BY { ACCUMULATOR | data-name } ]
```

The MEDIA-CLAMP option will open the paper tape media clamp.

The CARRIAGE option will open the forms transport. If the "integer/special-name" option is used, the carriage will stay open until a print or a programed close is made, at which time the platen will advance the specified number of lines. If the INDEXED BY option is used, the specified value will be added to the number of lines advanced.

Example:

```
15
16          OPEN MEDIA-CLAMP.
17          OPEN CARRIAGE START-LINE.
```

Perform

The use of this construct is to depart from the normal sequence of execution in order to execute a procedure which was declared as a subroutine in the DECLARATIVES. When the subroutine execution is complete, control is automatically returned to the next command in sequence in the mainline program. See discussion of EXIT.

The construct is as follows:

```
PERFORM procedure-name.
```

Only subroutines are allowed to be "PERFORMED". Attempting to "PERFORM" a paragraph-name will result in an error at compilation time.

Subroutines may be performed from within a subroutine. However, a hardware limit of four (4) is in effect. The nested subroutines are "exited" in reverse sequence to that in which they were performed.

This construct has no effect on the ACCUMULATOR.

Example:

```
PERFØRM SUBRT-1.
```

Position

The use of this construct is to provide for the position of the print ball.

The construct is as follows:

```
{POSITION} TO {integer      } [ INDEXED BY {ACCUMULATOR} ]
{POS     }    {special-name}               {data-name  }
```

This construct has no effect on the ACCUMULATOR unless the "INDEXED BY" option is used.

Example:

```
PØSITIØN TØ IO INDEXED BY PØS-MEM-LØC.
```

Read

The use of this construct is to provide for the reading of an 80-column card into the card input buffer.

The construct is as follows:

```
READ file-name
```

The file-name is the name ASSIGNED to the card file in the SELECT clause of the ENVIRONMENT DIVISION.

RED RIBBON
ROUND
SELECT

Red-Ribbon

The use of this construct is to provide for activating the red ribbon feature of the machine for the next display or print on the printer.

The construct is as follows:

```
RED-RIBBON
```

This construct has no effect on the ACCUMULATOR. This command reverses the ribbon for only the next DISPLAY instruction.

Example:

```
01    RED-RIBBON.
02    DISPLAY "CREDIT" UPON PRNTR.
```

Round

This construct is used to round to the Sterling unit PENCE data stored in memory or the ACCUMULATOR.

The construct is as follows:

```
ROUND { ACCUM     } TO PENCE
      { data-name }
```

Example:

```
03
04    ROUND TOTAL TO PENCE.
```

Select

The use of this construct is to provide the ability to:

1. Specify that a card be placed in the alternate stacker.
2. Specify a "SKIP" to a certain card column.
3. Specify "DUPLICATING" of certain card columns.

The construct is as follows:

```
                  { ALTERNATE STACKER                                          }
SELECT            {                                                            }   { special-name }
                  { { SKIP     } FUNCTION TO { THROUGH } integer               }   { integer      }
                  { { REPEAT   }             { THRU    }                       }
```

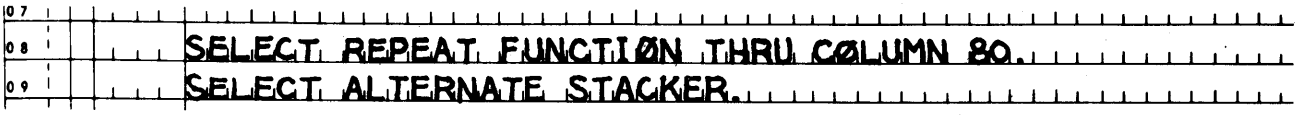
"SELECT ALTERNATE STACKER" will cause the card being punched to be stacked in the alternate stacker when it is stacked.

The "SKIP" option will cause a skip to card column "integer".

The "REPEAT" option will cause the card being punched to duplicate from the previous card to the "integer" card column.

This construct has no effect on the accumulator.

Example:



Stop Run

The use of this construct is to provide the ability to end execution of a job. The execution of this verb will cause the machine to return to Ready Mode.

The construct is as follows:

```
STOP RUN.
```

It is important to note that there must be at least one STOP RUN in every program.

Subtract

The use of this construct is to provide for the subtracting of one item from another.

The ACCUMULATOR cannot be SUBTRACTED from itself.

This construct has two options as follows:

OPTION 1

```
SUBTRACT { data-name-1 } FROM { data-name-2 } [GIVING data-name-3]
         { integer     }      { ACCUMULATOR }
         { ACCUMULATOR }

         [ROUNDED] [ON SIZE ERROR statements]
```

If no "GIVING" option is coded, the first operand will be subtracted from the second operand and the results stored into the second operand. If the "GIVING" option is used, the same subtraction will occur and the results will be stored into data-name-3.

Automatic decimal alignment is provided if both operands are data-names.

"ROUNDED" is only valid with a "GIVING" option and will produce correct results only when the

second operand has more decimal places than data-name-3.

When subtracting "data-name FROM ACCUMULATOR" or "ACCUMULATOR FROM data-name" with an "ON SIZE ERROR" clause, the ACCUMULATOR will contain undeterminable results if the error does occur.

See page 6-6 for a discussion of decimal alignment, ROUNDED and SIZE ERROR.

Example:

```
13    SUBTRACT DISCOUNT FROM AMOUNT
14        GIVING NET ROUNDED.
```

OPTION 2

SUBTRACT digit FROM ACCUMULATOR (integer) [ON SIZE ERROR statements]

Option 2 will provide for the subtraction of the digit (0-9) from the integer (1-15 from right to left) digit of the ACCUMULATOR.

Example:

```
15
16    SUBTRACT 3 FROM ACCUMULATOR (15)
17        ON SIZE ERROR ALARM.
```

Use

The use of this construct is to specify procedures or to define the program key table (PK-table). This would be used only in the DECLARATIVES.

The construct is as follows:

USE FOR { SUBROUTINE procedure-name | PK-TABLE table-name }

With the "SUBROUTINE" option, any combination of sentences and paragraphs may be used.

With the "PK-TABLE" option, the following constructs may be coded as PK table entries:

1. "ADVANCE" left or right with no "indexed by" clause.
2. "Display" options 2, 5, and 4 without the INDEXED BY option.
3. "GO TO"
4. "MOVE" options 4 through 7
5. "NO-OP"'
6. "OPEN CARRIAGE" with no "indexed by" clause.
7. "PERFORM"

8. "POSITION TO" with no "indexed by" clause.
9. "RED-RIBBON"

It is important to note that all PK tables must be declared before subroutines are declared, and END DECLARATIVES must follow the last entry in this part of the PROCEDURE DIVISION.

Example:

```
02
03  DECLARATIVES.
04      USE FØR PK-TABLE PK-CHØICE.
05          GØ TØ CØNTIN-PAGE.
06          GØ TØ SUBTØTAL.
07          MØVE 1 TØ SW2.
08      USE FØR SUBRØUTINE HEADING.
09          ADVANCE LEFT 1 LINE.
10          PØSITIØN TØ NAME-PØS.
11          ACCEPT 20 CHARACTERS FRØM KEYBØARD-PRNTR.
```

Recall that an EXIT from the subroutine will automatically be provided at the end. See page 6-15 for a further discussion of EXIT.

PART B: DATA COMMUNICATIONS

This part of the Procedural Constructs deals with the constructs required to activate the data communications equipment.

The specific verb formats together with a detailed discussion of each, appear on the following pages in alphabetical sequence.

Note:

The use of any of these verbs requires the presence of Data Communications firmware.

Accept

The purpose of the ACCEPT verb is to allow entry of alpha data directly into the DATA COMM output buffer.

The construct is:

ACCEPT alpha-file-data-name [FROM KEYBOARD]

This construct has no effect on the ACCUMULATOR.

IF
LOCATE (Part B)

If

The purpose of this variation of the "IF" verb is to allow interrogation of the flags associated with the data communications firmware and the adjunct exchange firmware.

The construct is as follows:

```
IF { XMT-RDY } [ { AND } { XMT-RDY } ] THEN { statements      } [ELSE statements]
   { RCV-RDY }   { OR  } { RCV-RDY }   { NEXT SENTENCE }
```

```
IF { XMT-RDY      }
   { RCV-RDY      }
   { DC-ERROR     }
   { POL-SEL-FLG  } THEN statements [ELSE statements]
   { BREAK-FLG    }
   { LINE-ACTVY-FLG }
```

```
IF { BUF-FULL  } THEN statements [ELSE statements]
   { BUF-EMPTY }
```

These constructs have no effect on the ACCUMULATOR, except when the POL-SEL-FLAG, BREAK-FLG, or LINE-ACTVY-FLG are tested.

Locate

The purpose of this verb is to provide the ability to properly handle the setting of the data communications buffer flags and pointers.

The format of this construct is as follows:

```
LOCATE file-name
```

When working directly with the data communications input buffer (NO WORK-AREA declared and no RESERVE ALTERNATE AREA declared), this construct must be used when accessing of the input buffer is complete in order to cause the data communications processor to "receive" the next record.

When working directly with the data communications output buffer (NO WORK-AREA declared), this construct must be used when accessing the output buffer to determine if the output buffer is available and to set the buffer pointers.

Move

The following variations of the "MOVE" verb are used for transfer of data to and from the data communications buffers and the adjunct exchange memory.

This construct has nine options:

OPTION 1

```
MOVE alpha-file-data-name [FROM BUFFER] TO alpha-data-name
```

Option 1 is used to move alpha data from the data communications buffer (or alternate buffer if RESERVE ALTERNATE AREA is declared) to memory.

OPTION 2

```
MOVE { alpha-data-name  } TO alpha-file-data-name [IN BUFFER]
     { alpha-literal    }
```

Option 2 is used to move alpha data from memory to the data communications buffer or alternate buffer if RESERVE ALTERNATE AREA is declared.

OPTION 3

```
MOVE { numeric-data-name } TO numeric-file-data-name [IN BUFFER]
     { literal           }
     { ACCUMULATOR       }
```

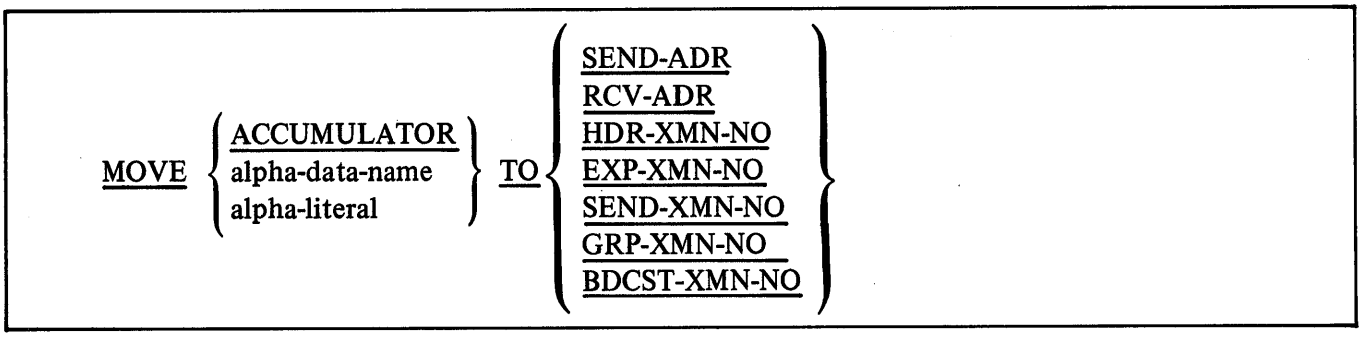
Option 3 is used to move numeric data directly to the data communications buffer or alternate buffer if RESERVE ALTERNATE AREA is declared.

OPTION 4

```
MOVE numeric-file-data-name [FROM BUFFER] TO { ACCUMULATOR      }
                                             { numeric-data-name }
```

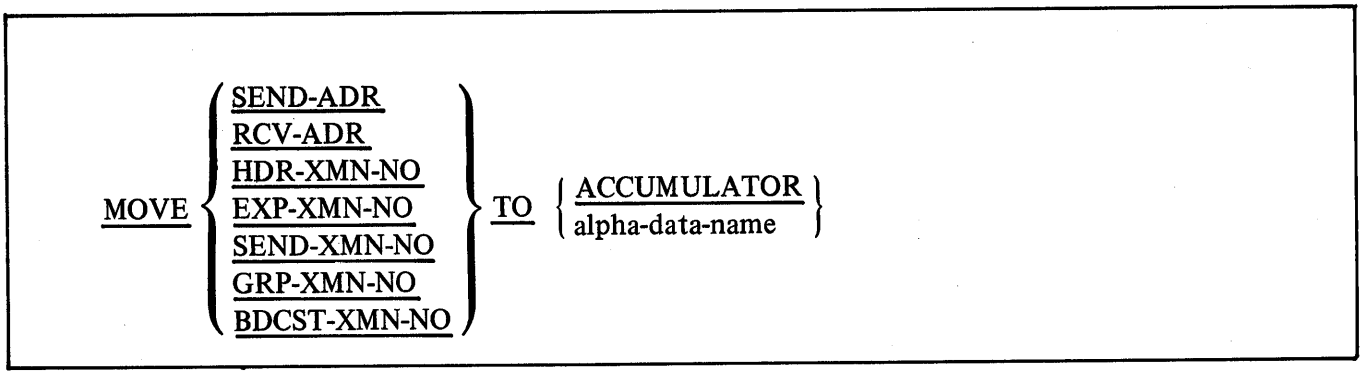
Option 4 is used to move numeric data directly from the data communications buffer (or alternate buffer if RESERVE ALTERNATE AREA is declared) to the ACCUMULATOR or to memory.

OPTION 5



Option 5 is used to load the various addresses and transmission numbers into adjunct exchange memory. See below for an interpretation of the abbreviations.

OPTION 6



Option 6 is used to retrieve the various address and transmission numbers from the adjunct exchange memory.

Options 5 and 6 are referring to the addresses and transmission numbers as follows:

1. SEND-ADR – Send Address Register
2. RCV-ADR – Receive Address Register
3. HDR-XMN-NO – Header Transmission Number
4. EXP-XMN-NO – Expected Transmission Number
5. SEND-XMN-NO – Send Transmission Number
6. GRP-XMN-NO – Group Transmission Number
7. BDCST-XMN-NO – Broadcast Transmission Number

OPTION 7

```
MOVE { 1 }
     { ONE }
     { 0 }
     { ZERO } TO DC-ERROR

MOVE { 1 }
     { ONE }
     { 0 }
     { ZERO } TO { RCV-RDY } [ { RCV-RDY } ]
                 { XMT-RDY }   { XMT-RDY }

MOVE { 1 }
     { ONE }
     { 0 }
     { ZERO } TO { POL-SEL-FLG }      [ { POL-SEL-FLG }      ... ]
                 { BREAK-FLG }          { BREAK-FLG }
                 { LINE-ACTVTY-FLG }    { LINE-ACTVTY-FLG }
```

Option 7 is used to set or reset the indicated flags.

OPTION 8

```
MOVE { TWO-WIRE-CNTL  } TO DATA-COMM
     { FOUR-WIRE-CNTL }
```

Option 8 is used to load the register in the adjunct exchange memory for two wire or four wire control.

OPTION 9

```
MOVE @ab@ TO DATA-COMM
```

Option 9 is used to load non-graphic USASCII characters (see USASCII Table in Appendix A) into the data communications buffer or alternate buffer if USE ALTERNATE AREA was declared. The "a" represents the column number and "b" represents the row number of the table. Since "b" may be only one digit, the row numbers 10-15 are designated with the letters A-F (10=A, 11=B, 12=C, 13=D, 14=E, and 15=F).

Read

The purpose of this construct is to read a file which has been assigned to data communications input.

The format of this construct is as follows:

```
READ file-name
```

STOP
WRITE (Part B)

This construct will determine if a record has been received into the data communications processor receive buffer.

If a record has been received and no ALTERNATE AREA was declared, control will be transferred to the next sentence. If an ALTERNATE AREA was declared, the data will be transferred from the receive buffer to the alternate buffer, the receive flag will be reset and control will be transferred to the next sentence.

If no record has been received, the system will hang in a wait loop until a record has been received.

Note:

A test of the RCV-RDY (receive) flag should normally be made before using the READ construct to prevent hanging in the wait loop.

Example:

```
13
14      IF RCV-RDY THEN READ DATA-COMM-MESSAGE
15          ELSE GO TO MESSAGE-NOT-RECEIVED.
```

Stop

This construct is used to turn the power off to the system.

This construct is as follows:

```
STOP MACHINE
```

Write

The use of this construct is to transmit a record through the data communications network.

This construct is as follows:

```
WRITE record-name
```

This construct will cause the transmit ready (send) flag to be set when NO WORK-AREA and no ALTERNATE AREA have been declared.

If RESERVE ALTERNATE AREA or USE WORK-AREA is declared, this construct will determine if the send buffer is clear and then transfer the data from the work area or alternate buffer to the send buffer and set the transmit ready (send) flag. If the send buffer is not clear, the system will hang in a wait loop until the send buffer is clear.

Note:

A test of the XMT-RDY (send) flag should normally be made before using the WRITE to prevent hanging in the wait loop.

Example:

```
17
18      IF XMT-RDY THEN GO TO BUFFER-NOT-CLEAR
19          ELSE WRITE DATA-COMM-RECORD.
```

SECTION 7

SERIES L/TC COBOL COMPILATION

B 3500 Environment

GENERAL

As stated in the introduction, a program written in Series L/TC COBOL, a source program, is accepted as input by the B 3500 version of the series "L/TC" COBOL Compiler. The compiler then verifies that all rules outlined in this manual are satisfied and translates the source program language into an object program language capable of operating on a Series L/TC Computer. Symbolic program output (for subsequent input to the Series L/TC Assembler) is available as well as the object program output. This section deals with the input/output, compilation, and assembly options available with the B 3500.

INPUT

The input for compilation-assembly is the magnetic library tape, the necessary control and option cards, and the source data which is either a card deck, magnetic tape or disk file that includes the source program.

The library tape contains the following programs:

COMPILER PROGRAMS:

L57305
Q57305 } These programs are the Compiler programs
M57305

ASSEMBLER PROGRAMS:

ASSEMB
QCONV
OBJCRD } These are the Assembler programs
LCNVRS
CRDCVR

XRFBTC This is the Assembler Cross-reference program

FILES LOADED ONTO DISK. The COND and OPTBL files are loaded onto a disk during the loading of the master tape and are used internally in the assembler portion of the compiler program. The COND (condition file) is used in the error detection routines and OPTBL is the operation code file.

TO EXECUTE THE ASSEMBLER ONLY. The specific procedure for executing only those programs dealing with the assembly process may be found in the discussion of Assembler III of Section 5 in the Series L/TC Assembler Manual.

OUTPUT

The output of the Series L/TC COBOL compiler may be any, or all, of the output options described under control options (below).

If only the compiler is used, a syntax check of the COBOL constructs will be executed and the output

will be a listing of the COBOL statements and any errors present. Figure 7-1 is an example of a compiler listing.

If the compiler-assembler is used, and if the "CODE" option is elected, a print-out, which lists the symbolic code, object code developed and any error comments is produced after the source listing. The symbolic code for each source statement appears under that statement on the source listing (see Figure 7-2). This is in addition to the paper tape or card media produced as output – Figure 7-3 is an example of an Assembler listing. The word and syllable of the instruction is listed along with sequence number, object code, expanded print-out of the source card, and decimal equivalent for each label used within a source statement.

```
02319  EACH-TOTAL.
02320      POSITION TO TALLY-POS.
02401      DISPLAY DESCRIPTION-PRINT (ITEM-MODIFIER) UPON PRNTR
02402      ADVANCE RIGHT 1 LINE.
02403      POSITION TO TALLY-POS.
02404      DISPLAY GRAND-TOTALS-PRINT (ITEM-MODIFIER) UPON PRNTR
024045     DISPLAY "-" GRAND-TOTALS-PRINT (ITEM-MODIFIER) NEGATIVE.
02405      IF ITEM-MODIFIER IS EQUAL TO 6
02406                                        STOP RUN.
02407      ADVANCE RIGHT 2 LINES.
02408      ADD 1 TO ITEM-MODIFIER.
02409      GO TO EACH-TOTAL.
031900 END-OF-JOB.
```

Figure 7-1 Source Listing

```
 1504      MULTIPLY ITEM-MODIFIER BY 2.
                                   01880        TRA     DN0021 000
                                   01890        LSR     00
                                   01900        MUL     LIT    001
                                   01910        TRM     DN0021 000
01505      POSITION TO SO INDEXED BY ITEM-MODIFIER
                                   01920        TRA     DN0021 000
                                   01930        TAIR    1
                                   01940        MOD     1
                                   01950        POS     050
01506      ACCEPT ITEM-QUANTITY-ENTER FROM KEYBOARD
                                   01960        NKR     02
                                   01970        TRM     DN0022 000 00
015065     ADD ITEM-QUANTITY TO LINE-QUANTITY.
                                   01980        TRA     DN0023 000
                                   01990        ADM     DN0011 000
```

Figure 7-2 Source Listing with Code

WORD	SYL	OBJECT CODE	SEQ. NO.	SYM. LOC.	OP CODE	FD. LN.	A-PARAMETER LABEL	INC	B PAR	C PAR	LABEL DEC	LABEL EQU
65	0	EB1D	161 0		POS		030					
	1	AC06	162 0		TK		006					
	2	EB24	163 0		POS		037					
	3	AC05	164 0		TK		005					
66	0	EB28	165 0		POS		044					
	1	AC07	166 0		TK		007					
	2	EB35	167 0		POS		054					
	3	4192	168 0		SK		K		1	1		
67	0	7844	169 0		BRU			+006			68	2
	1	AC13	170 0		TK		019					
	2	EB49	171 0		POS		074					
	3	A440	172 0		NKR		04		00			
68	0	3010	173 0		TRM		DN0011				16	
	1	704C	174 0		BRU		LB0014				76	0
	2	A620	175 0	LB0015	NK		02		00			
	3	3019	176 0		TRM		DN0021				25	

Figure 7-3 Assembler Listing from Compiler-Assembler

MCP CONTROL CARDS

The following MCP (Master Control Program) Control Cards are used in the compiler process:

```
1
2  indicates an invalid code in card column one.
3
```

EXECUTE CARD

The execute card initiates program execution. It must be punched in the following formats:

```
1
2  EXECUTE  57305
3
```

This is used to initiate the compiler-assembler (unless a SYNTAX option card is used, in which case only compilation will occur).

FILE EQUATE CARD

If the NEWD option is used to create an updated disk file of the source program, or if the DISK option is used to specify that the source input is from disk, the control cards –

```
1
2  FILE SORSE = file-label
3
1
2  FILE SORSD = file-label
3
```

must be placed between the EXECUTE card and the DATA (or DATAB) SORCE cards. The file-name is assigned by the operator and must be the same on each card. The FILE SORSE card is used to equate the file-name with the input disk file and the FILE SORSD card equates the file-name with the output disk file.

DATA CARD

The Data Card specifies the type source media. It must be punched in the following format:

```
1                        1
2  DATA SORCE    or      2  DATAB SORCE
3                        3
```

This tells the system that the source media is cards, punched in either EBCDIC or BCL code, respectively. BCL does not include some special characters available with EBCDIC. If EBCDIC code is used, it will be accepted by the compiler only if the 12-0 card code is used for the plus sign (+).

DOLLAR SIGN CARD

The dollar sign card specifies the options which control the input and output during the compilation and assembly process. It must be punched in the following format:

```
$ (List option(s) here)
```

A list of the available options are below.

DATA DECK

The source or symbolic deck to be compiled and/or assembled.

END CARD

The end card must follow any card deck. It is punched in the following format:

```
1
2  END
3
```

It tells the system that the input from the Card Reader is complete.

OPTION CARDS

The following Dollar ($) Options are available to use with the L/TC COBOL Compiler. The Dollar sign ($) must be coded in card column 7. The options are coded free form starting in card column 9. If more than one non-continued card is used, the last one used will set the various parameters. The others will be disregarded.

The options are:

COMPILER OPTIONS

1. LIST – This will cause a listing of the COBOL statements. If no $ options are used, LIST is automatic.
2. CODE – "CODE" will cause the symbolic code to be listed for each COBOL construct.
3. SYNTAX – This will cause a compilation for syntax purposes. No code will be generated nor will the assembler be activated.
4. TAPE – This will specify that the input is an "LSOLT" (Source Language Tape) magnetic tape with "patch" cards in the card reader.
5. DISK – This will specify that the input is from disk with "patch" cards in the card reader.
6. NEWT – This will cause an updated LSOLT tape to be created.
7. NEWD – This will cause an updated disk file to be created.
8. NEWC – This will cause the compiler to give a BCL source card deck as output.
9. RESEQ – This will cause the symbolic program to be re-sequenced starting at 100 and increased by an increment of 100.
10. BLNK – This will cause all cards with card columns 7 through 72 blank to be purged when a "NEWT" is requested.
11. Identification: Any characters punched in columns 73-80 of the card will be inserted into all source statements in columns 73-80.

ASSEMBLER OPTIONS

12. SYM-PT – This will cause the assembler to create a symbolic paper tape.
13. SYM-CN – This will cause the assembler to create a symbolic card deck punched in "EBCDIC" card codes.
14. SYM-CD – This will cause the assembler to create a symbolic card deck punched in "BCL" card codes.

15. MEMORY nnn – This will cause the assembler to limit the generated program to the nnn size and print error messages if the nnn limit is exceeded.
16. OBJCD – This will cause the assembler to punch the object program into punched cards instead of paper tape.

As stated earlier, the assembler will always be activated by the compiler after compilation unless the "SYNTAX" option is used or unless an error occurs during compilation.

After the assembly process, punching of an object paper tape is always assumed unless the "OBJCD" option is used.

$ cards may be stacked, however, only the last one will set up the various parameters. Should the situation arise where all the desired options cannot be punched into a single card, a continuation card with a "-" in column 7 may be used.

EQUIPMENT REQUIRED

The following system hardware is required for the L/TC Compiler-Assembler Program:

B 3500 – 60 KB Core

1 Module Disk

1 Tape Unit (7 or 9 channel)

Card Reader

Paper Tape Punch (Optional for object or symbolic tape out).

Paper Tape Reader (Optional for symbolic paper tape input)

Card Punch (Optional for symbolic card object card or source card output)

Line Printer

OPERATING INSTRUCTIONS

1. Magnetic Tape Units
   Mount the master tape containing the Series L/TC COBOL Compiler programs.
2. Card Punch (If symbolic card output is required)
   Load the hopper on the card punch with sufficient cards and depress the Start button.
3. To Load the Tape
   a. VIA CARD READER – Load the single card:
      ? LOAD tape-name, program-name, program-name, etc., in the card reader, depress the RESET button and then the Start button. This will load the specified programs of the master tape.
4. To Execute the Compiler-assembler
   The following cards should be placed in the card reader hopper:
   a.
   ```
   1
   2 EXECUTE L57305
   3
   ```
   b.
   ```
   1                    1
   2 DATA SORCE or 2 DATAB SORCE
   3                    3
   ```
   c.
   ```
   $ (option card(s)
   ($ is coded in column 7)
   ```

d. The program source deck

e. 1
2 End
3

Depress the Reset button and then the Start button.

5. To Execute the Assembler Only
The specific procedures for executing only those programs dealing with the assembly process may be found in the discussion of Assembler III of Section 5 in the Series L/TC Assembler Manual.

ERROR DETECTION

If an error occurs during compilation, the error will be listed and compilation will continue but Assembly will not take place. A list of Compiler errors is in Appendix E.

UNITED STATES OF AMERICA
STANDARD CODE FOR INFORMATION INTERCHANGE
(USASCII)

b_7 →					0	0	0	0	1	1	1	1
b_6 →					0	0	1	1	0	0	1	1
b_5 →					0	1	0	1	0	1	0	1
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	1	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Table A-1

USASCII COLUMN 1 FIELD IDENTIFIER CODES					
	PAPER TAPE VALUE	FLAG PATTERN SET BY CODE* OCK FLAG NUMBER			
CODE	a, b	3	2	1	4
DLE	9,0	0	0	0	0
DC1	1,1	0	0	0	1
DC2	1,2	0	0	1	0
DC3	9,3	0	0	1	1
DC4	1,4	0	1	0	0
NAK	9,5	0	1	0	1
SYN	9,6	0	1	1	0
ETB	1,7	0	1	1	1
CAN	1,8	1	0	0	0
EM	9,9	1	0	0	1
SUB	9,A	1	0	1	0
ESC	1,B	1	0	1	1
FS	9,C	1	1	0	0
GS	1,D	1	1	0	1
RS	1,E	1	1	1	0
US	9,F	1	1	1	1

*0 = flag is reset 1 = flag is set

Table A-2

USASCII COLUMN 0 FIELD IDENTIFIER CODES**					
	PAPER TAPE VALUE	FLAG PATTERN SET BY CODE* Y FLAG NUMBER			
CODE		3	2	1	4
NUL	0,0	0	0	0	0
SOH	8,1	0	0	0	1
STX	8,2	0	0	1	0
ETX	0,3	0	0	1	1
EOT	8,4	0	1	0	0
ENQ	0,5	0	1	0	1
ACK	0,6	0	1	1	0
BEL	8,7	0	1	1	1
BS	8,8	1	0	0	0
HT	0,9	1	0	0	1
IF	0,A	1	0	1	0
VT	8,B	1	0	1	1
FF	0,C	1	1	0	0
CR	8,D	1	1	0	1
SO	8,E	1	1	1	0
SI	0,F	1	1	1	1

*0 = flag is reset 1 = flag is set
**Setting depends on firmware set

Table A-3

ACCUMULATOR FLAG CODES: The following chart shows the paper tape codes that set the Accumulator Flags during Read Numeric instructions (when code is contained in table of code assignments).

TAPE CODES				ACCUMULATOR FLAGS*			
				M	C	S	-
A,0		C,0	5,0	0	0	0	0
2,1		4,1	D,1	0	0	0	1
2,2		4,2	D,2	0	0	1	0
A,3		C,3	5,3	0	0	1	1
2,4		4,4	D,4	0	1	0	0
A,5		C,5	5,5	0	1	0	1
A,6		C,6	5,6	0	1	1	0
2,7		4,7	D,7	0	1	1	1
2,8		4,8	D,8	1	0	0	0
A,9		C,9	5,9	1	0	0	1
A,A	3,A	C,A	5,A	1	0	1	0
2,B	B,B	4,B	D,B	1	0	1	1
A,C	3,C	C,C	5,C	1	1	0	0
2,D	B,D	4,D	D,D	1	1	0	1
2,E	B,E	4,E	D,E 7,E	1	1	1	0
A,F	3,F	C,F	5,F	1	1	1	1

* 0 = flag is reset;
1 = flag is set

Table A-4

SERIES L/TC CHARACTER SET

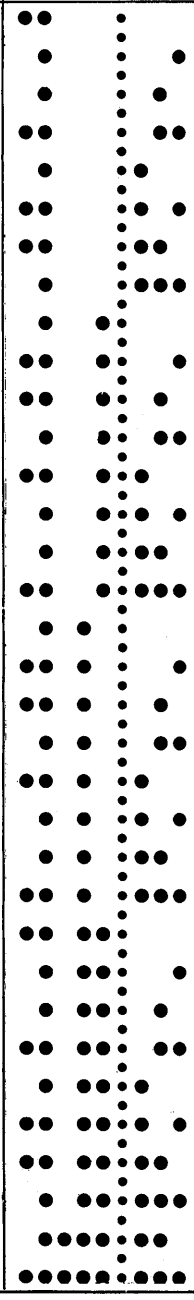
The USASCII and Commercial character sets for the Series L/TC Systems are listed below in their collating sequence in ascending order. Each character set consists of 64 graphic characters, the Space code, and the End of Alpha code. The USASCII character set consists of the USASCII characters in columns 2, 3, 4, and 5 of the USASCII table, plus End of Alpha (NUL) and Overline. Those Commercial characters that differ from the USASCII characters are shown in parentheses.

The internal or machine language code for each character is given; this code consists of two hexadecimal digits which correspond to the column and row number of the character in the USASCII table (A=row 10, B=11, C=12, D=13, E=14, F=15). In addition, the decimal value of each character is given as required when using Index Registers for modification.

Character	Internal Code	Indexing Value	Character	Internal Code	Indexing Value	Character	Internal Code	Indexing Value	Character	Internal Code	Indexing Value
End of Alpha (NUL)	0 0	0									
Space	2 0	32	0	3 0	48	@	4 0	64	P	5 0	80
!	2 1	33	1	3 1	49	A	4 1	65	Q	5 1	81
”	2 2	34	2	3 2	50	B	4 2	66	R	5 2	82
#	2 3	35	3	3 3	51	C	4 3	67	S	5 3	83
$	2 4	36	4	3 4	52	D	4 4	68	T	5 4	84
%	2 5	37	5	3 5	53	E	4 5	69	U	5 5	85
&	2 6	38	6	3 6	54	F	4 6	70	V	5 6	86
’	2 7	39	7	3 7	55	G	4 7	71	W	5 7	87
(	2 8	40	8	3 8	56	H	4 8	72	X	5 8	88
)	2 9	41	9	3 9	57	I	4 9	73	Y	5 9	89
*	2 A	42	:	3 A	58	J	4 A	74	Z	5 A	90
+	2 B	43	;	3 B	59	K	4 B	75	[(¾)	5 B	91
,	2 C	44	<(½)	3 C	60	L	4 C	76	\ (¢)	5 C	92
-	2 D	45	=	3 D	61	M	4 D	77	](CR)	5 D	93
.	2 E	46	>(¼)	3 E	62	N	4 E	78	∧ (°)	5 E	94
/	2 F	47	?	3 F	63	O	4 F	79	—	5 F	95
									~(◇)	7 E	126
									DEL	7 F	127

Table A-5

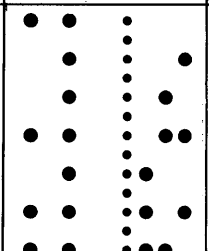
EBCIDIC AND BCL CODE

		EBCIDIC			BCL		
L/TC GRAPHIC	PAPER TAPE CODE	GRAPHIC CHAR	CARD CODE	KEY	GRAPHIC CHAR	CARD CODE	KEY
@		@	8-4	*	@	8-4	*
A		A	12-1	*	A	12-1	*
B		B	12-2	*	B	12-2	*
C		C	12-3	*	C	12-3	*
D		D	12-4	*	D	12-4	*
E		E	12-5	*	E	12-5	*
F		F	12-6	*	F	12-6	*
G		G	12-7	*	G	12-7	*
H		H	12-8	*	H	12-8	*
I		I	12-9	*	I	12-9	*
J		J	11-1	*	J	11-1	*
K		K	11-2	*	K	11-2	*
L		L	11-3	*	L	11-3	*
M		M	11-4	*	M	11-4	*
N		N	11-5	*	N	11-5	*
O		O	11-6	*	O	11-6	*
P		P	11-7	*	P	11-7	*
Q		Q	11-8	*	Q	11-8	*
R		R	11-9	*	R	11-9	*
S		S	0-2	*	S	0-2	*
T		T	0-3	*	T	0-3	*
U		U	0-4	*	U	0-4	*
V		V	0-5	*	V	0-5	*
W		W	0-6	*	W	0-6	*
X		X	0-7	*	X	0-7	*
Y		Y	0-8	*	Y	0-8	*
Z		Z	0-9	*	Z	0-9	*
[or ¾		[	12-8-2	M		12-8-4	M
\ or ¢		\	11-8-7	#	\	11-8-7	M
] or cr		→	11-8-2	M		0-8-6	M
^ or °		+	12-8-6	#			
-		-	0-8-5	#	-	0-8-2	M
~ or ◇		|	12-8-7	#	←	12-8-7	M
		DELETE	12-9-7	M			

* Keys on 026, 029, 149 and 150 Punch Correct Code.
M Multipunch on 026, 029, 149 and 150.
Keys on 029, and 150 punch correct code; multipunch on 026 and 149.
I Keys on 029 and 150 punch invalid code; multipunch on 026, 029, 149 and 150.

Table A-6

EBCIDIC AND BCL CODE (cont'd)

		EBCIDIC			BCL		
L/TC GRAPHIC	PAPER TAPE CODE	GRAPHIC CHAR	CARD CODE	KEY	GRAPHIC CHAR	CARD CODE	KEY
SP		SP	BLANK	*	SP	BLANK	*
!		!	11-0	I	X	11-0	M
”		”	8-7	#	”	0-8-7	M
#		#	8-3	*	#	8-3	*
$		$	11-8-3	*	$	11-8-3	*
%		%	0-8-4	*	%	0-8-4	*
&		&	12	*	&	12	*
'		'	8-5	#	⩾	8-7	M
(		(	12-8-5	#	(	12-8-5	#
)		)	11-8-5	#	)	11-8-5	#
*		*	11-8-4	*	*	11-8-4	*
+		+	12-0	I	+	12-0	I
,		,	0-8-3	*	,	0-8-3	*
–		–	11	*	–	11	*
.		.	12-8-3	*	.	12-8-3	*
/		/	0-1	*	/	0-1	*
0		0	0	*	0	0	*
1		1	1	*	1	1	*
2		2	2	*	2	2	*
3		3	3	*	3	3	*
4		4	4	*	4	4	*
5		5	5	*	5	5	*
6		6	6	*	6	6	*
7		7	7	*	7	7	*
8		8	8	*	8	8	*
9		9	9	*	9	9	*
:		:	8-2	#	:	8-5	M
;		;	11-8-6	#	;	11-8-6	#
< or ½		<	12-8-4	#	<	12-8-6	M
=		=	8-6	#	=	0-8-5	M
> or ¼		>	0-8-6	#	>	8-6	M
?		?	0-8-7	#	?	ALL OTHER	M

* Keys on 026, 029, 149 and 150 Punch Correct Code.

M Multipunch on 026, 029, 149 and 150.

Keys on 029, and 150 punch correct code; multipunch on 026 and 149.

I Keys on 029 and 150 punch invalid code; multipunch on 026, 029, 149 and 150.

Table A-6 (cont'd)

COBOL SYNTAX

IDENTIFICATION DIVISION

IDENTIFICATION DIVISION.

[PROGRAM-ID. Any entry from 1 to 30 characters.]

[AUTHOR. Any entry including appropriate copyright statement.]

[INSTALLATION. Any entry.]

[DATE-WRITTEN. Any entry.]

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

[SECURITY. Any entry.]

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

ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
```

```
[CONFIGURATION SECTION.]
```

```
    [SOURCE-COMPUTER. B-3500.]
```

```
    [OBJECT-COMPUTER. { TC-500 }
                      { TC-700 }
                      { L-2000 }]
```

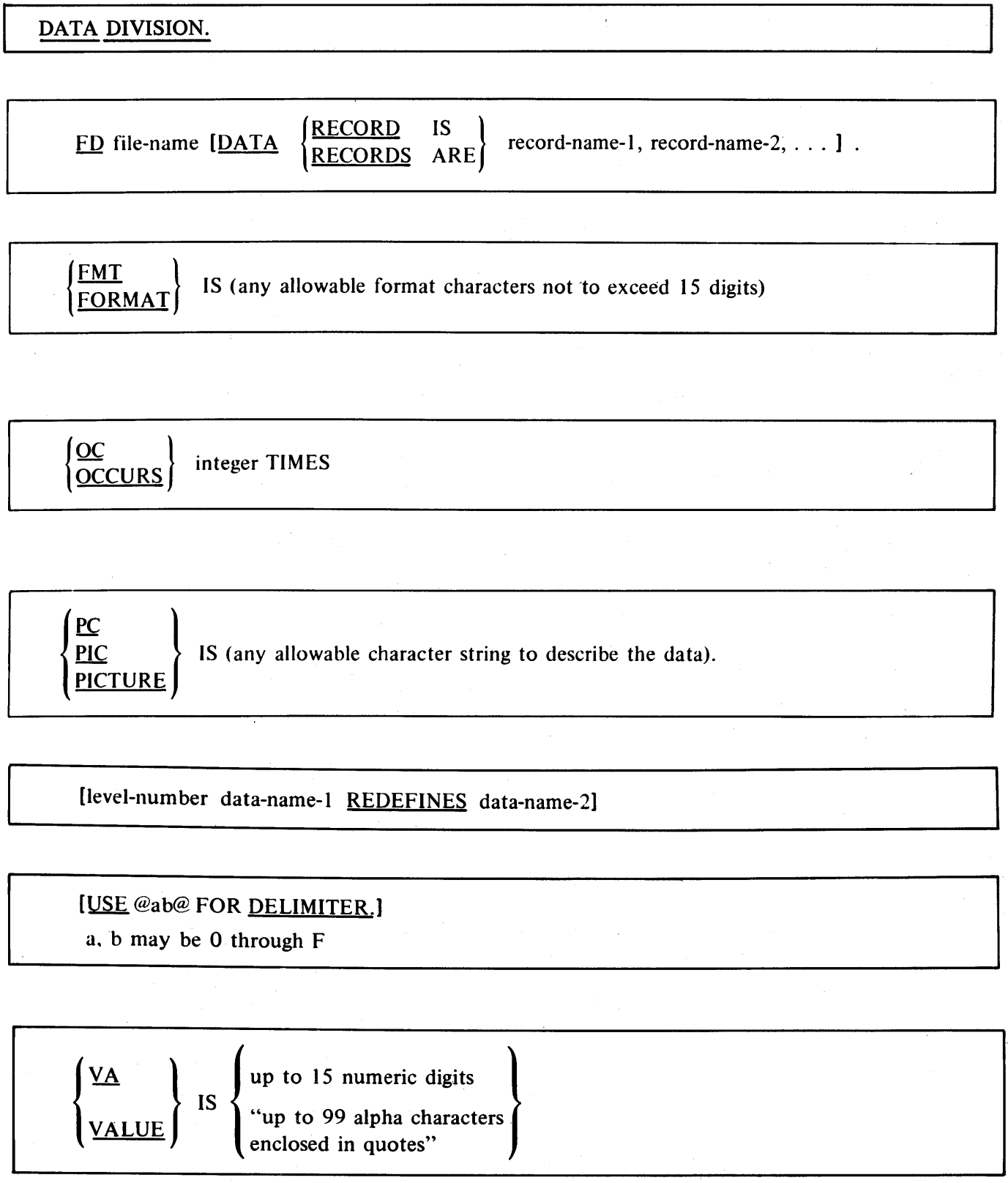
```
    [SPECIAL-NAMES.
        [data-name IS { POSITION } integer.]
                      { POS      }
        [data-name IS LINE integer.]
        [data-name IS { COLUMN } integer.]
                      { COL    }]
```

```
[INPUT-OUTPUT SECTION.]
```

```
[FILE-CONTROL.
    SELECT file-name ASSIGN TO { CARD-READER } { IN  }
                               { DATA-COMM  } { OUT }
    [RESERVE ALTERNATE AREA] [ACCESS MODE IS SEQUENTIAL] { USE } WORK-AREA.]
                                                         { NO  }
```

```
    [I-O-CONTROL.
        [SAME AREA FOR file-name-1, file-name-2] .
        [SAME WORK-AREA FOR file-name-1, file-name-2, . . .] .]
```

DATA DIVISION



PROCEDURE DIVISION

PART I. PROCESSING; PAPER TAPE I/O: 80-COLUMN CARD I/O.

```
PROCEDURE DIVISION.
```

Accept

OPTION 1

```
ACCEPT { data-name                          } [ FROM { KEYBOARD          } ]
       { INTO ACCUMULATOR data-name         }        { KEYBOARD-PRNTR    }
                                                     { KEYBOARD-PCH      }
                                                     { RDR               }
                                                     { RDR-PRNTR         }
                                                     { RDR-PCH           }
                                                     { RDR-PRNTR-PCH     }
                                                     { KYBRD-PRNTR-PCH   }
```

OPTION 2

```
ACCEPT integer CHARACTERS [ FROM { KEYBOARD-PRNTR    } ]
                                 { RDR-PRNTR         }
                                 { RDR-PRNTR-PCH     }
                                 { KYBRD-PRNTR-PCH   }
```

OPTION 3

```
ACCEPT FROM KEYS
```

OPTION 4

```
ACCEPT INTO ACCUMULATOR FROM { KEYBOARD }
                             { RDR      }
```

Add

OPTION 1

```
ADD { data-name-1  } TO { data-name-2  } [ON SIZE ERROR statements]
    { integer      }    { ACCUMULATOR  }
    { ACCUMULATOR  }
```

OPTION 2

```
ADD { data-name-1 } { data-name-2 }          GIVING data-name-3
    { integer     } { ACCUMULATOR }
    { ACCUMULATOR }
    [ROUNDED] [ON SIZE ERROR STATEMENTS]
```

OPTION 3

```
ADD digit TO ACCUMULATOR (integer) [ON SIZE ERROR statements]
```

Advance

```
ADVANCE [ { LEFT  } ] { TO special-name } [INDEXED BY { ACCUMULATOR } ]
        [ { RIGHT } ] { TO integer LINE }             { data-name   } ]
        [ { BOTH  } ] { integer LINES   }
```

Alarm

```
ALARM
```

Close

```
CLOSE CARRIAGE
```

Convert

OPTION 1

```
CONVERT { ACCUM     } TO { SHILLING      }
        { data-name }    { FARTHING      }
                         { SPEC-FARTHING }
                         { POUNDS        }
                         { PENCE         }
                         { SPEC-PENCE    }
```

OPTION 2

```
CONVERT { ACCUM data-name } TO data-name-2 CHECK-DIGIT [ (integer) ]
        { data-name-1     }
```

Display

OPTION 1

```
DISPLAY { data-name          } [FROM BUFFER] [ UPON { PRNTR     } ]
        { "literal"          }               {      { PCH       } }
        { ACCUMULATOR data-name }              {      { PRNTR-PCH } }
```

OPTION 2

```
DISPLAY { "character" } [ { PREVIOUS-RIBBON                                  } ]
        { QUOTE       }   { { data-name  } { NEGATIVE } }                    }
                          { { ACCUMULATOR } { POSITIVE } }                   }
```

OPTION 3

```
DISPLAY ACCUMULATOR (integer) data-name [ UPON { PRNTR     } ]
                                               { PCH       }
                                               { PRNTR-PCH }
```

OPTION 4

```
DISPLAY integer SPROCKET-HOLES [INDEXED BY PCH-REG]
```

OPTION 5

```
DISPLAY @ab@ UPON PCH
```

OPTION 6

```
DISPLAY { ACCUM     } { FARTHING   }
        { data-name } { HALF-PENNY }
```

Divide

```
DIVIDE { data-name-1 } INTO { data-name-2   } [GIVING data-name-3]
       { integer     }      { ACCUMULATOR  }
    [ROUNDED] [ON SIZE ERROR statements]
```

Enable

```
ENABLE [table-name] PK1 PK2 . . . . PK8 PK9 . . . . PK16
```

End-of-Job

```
END-OF-JOB.
```

Exit

```
EXIT
```

Fill

```
FILL record-name.
```

Go To

```
GO TO paragraph-name.
```

If

A. RELATIVE TESTS

OPTION 1

```
IF { data-name-1 } IS { GREATER THAN  } { data-name-2 }
   { literal-1   }    {    >         }  { literal-2   }
                      { LESS THAN     }
                      {    <         }
                      { EQUAL TO      }
                      {    =         }
                      { NOT EQUAL TO  }
                      { NOT =         }
     THEN { statements-1    } [ELSE statements-2].
          { NEXT SENTENCE   }
```

B. ACCUMULATOR TESTS

OPTION 1

```
IF ACCUMULATOR (integer-1) LESS THAN integer-2 THEN { statements | NEXT SENTENCE } [ELSE statements].
```

OPTION 2

```
IF { data-name | ACCUMULATOR } IS ZERO THEN { statements | NEXT SENTENCE } [ELSE statements] .
```

OPTION 3

```
IF SIZE ERROR THEN statements [ELSE statements] .
```

C. ERROR CONDITION TESTS

OPTION 1

```
IF { RDR-ERR | RDR-COND } [ { AND | OR } { RDR-ERR | RDR-COND } ]
    THEN { statements | NEXT SENTENCE } [ELSE statements] .
```

OPTION 2

```
IF { PCH-ERR | PUNCH-OFF | NO-MEDIA | LOW-TAPE } [ { AND | OR } { PCH-ERR | PUNCH-OFF | NO-MEDIA | LOW-TAPE } ... ]
    THEN { statements | NEXT SENTENCE } [ELSE statements] .
```

OPTION 3

```
IF END-OF-PAGE THEN { statements | NEXT SENTENCE } [ELSE statements] .
```

D. ACCUMULATOR FLAG TESTS

```
IF {data-name    } { NFLAG } [ {AND} { NFLAG } ... ]
   {ACCUMULATOR  } { SFLAG }   {OR } { SFLAG }
                   { CFLAG }         { CFLAG }
                   { MFLAG }         { MFLAG }

   THEN {statements    } [ELSE statements] .
        {NEXT SENTENCE }
```

E. SWITCH TESTS

```
IF { SW1 } [ {AND} { SW1 } ... ] THEN {statements    } [ELSE statements] .
   { SW2 }   {OR } { SW2 }            {NEXT SENTENCE }
   { SW3 }         { SW3 }
   { SW4 }         { SW4 }
```

```
IF { SW5 } [ {AND} { SW5 } ... ] THEN {statements    } [ELSE statements] .
   { SW6 }   {OR } { SW6 }            {NEXT SENTENCE }
   { SW7 }         { SW7 }
   { SW8 }         { SW8 }
```

F. OCK TESTS

```
IF { OCK1 } [ {AND} { OCK1 } ... ] THEN {statements    } [ELSE statements] .
   { OCK2 }   {OR } { OCK2 }            {NEXT SENTENCE }
   { OCK3 }         { OCK3 }
   { OCK4 }         { OCK4 }
```

G. CHECK DIGIT TEST

```
IF {ACCUM data-name} CHECK-DIGIT [ (integer) ] [FROM data-name] [TRUNCATED]
   {data-name      }

   THEN statements [ELSE statements] .
```

H. STERLING TESTS

```
IF ACCUMULATOR IS {NON-DECIMAL } THEN statements [ELSE statements] .
                  {NON-STERLING}
```

I. TC-700 FLAG TESTS

```
IF { TELLER-1
     TELLER-2
     SUPERVISOR
     PB-FIRST-LINE
     PB-LAST-LINE
     PB-FOLD
     PB-PRESENT } THEN statements [ELSE statements] .
```

Move

OPTION 1

```
MOVE { data-name
       integer
       ZERO
       ACCUMULATOR
       "literal" } TO { data-name
                        ACCUMULATOR }
```

OPTION 2

```
MOVE digit TO ACCUMULATOR (integer)
```

OPTION 3

```
MOVE { special-name
       integer } TO [ { LEFT
                        RIGHT } ] { LIMIT-REG
                                    COUNT-REG } [ [INDEXED BY { ACCUMULATOR
                                                                data-name } ]
```

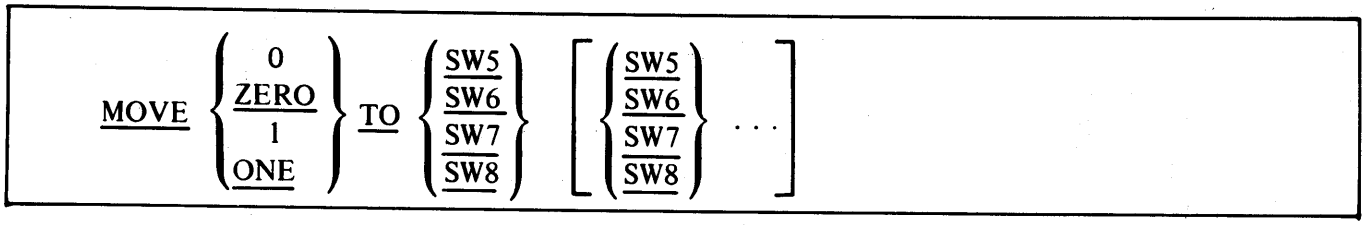
OPTION 4

```
MOVE { 0
       ZERO
       1
       ONE } TO { NFLAG
                  SFLAG
                  CFLAG
                  MFLAG } [ { NFLAG
                              SFLAG
                              CFLAG
                              MFLAG } ... ]
```

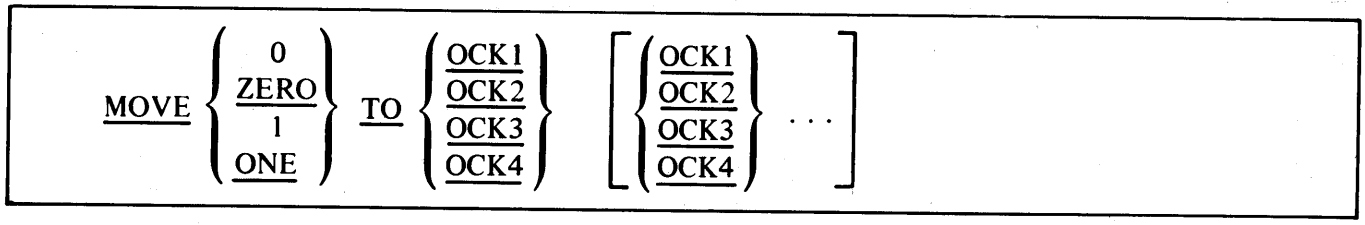
OPTION 5

```
MOVE { 0
       ZERO
       1
       ONE } TO { SW1
                  SW2
                  SW3
                  SW4 } [ { SW1
                            SW2
                            SW3
                            SW4 } ... ]
```

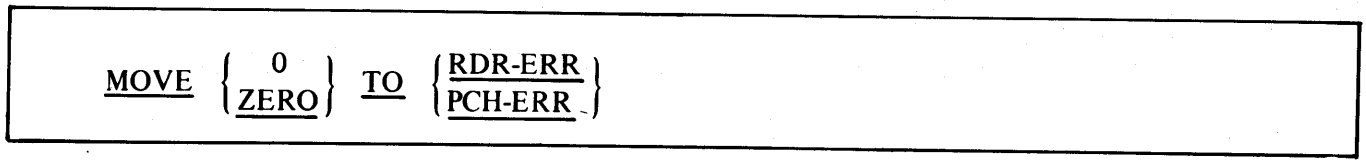
OPTION 6



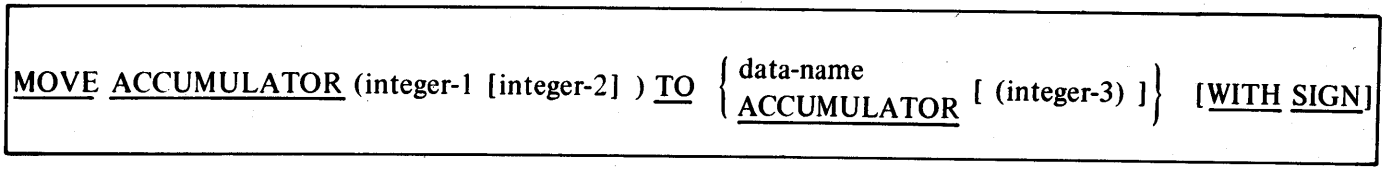
OPTION 7



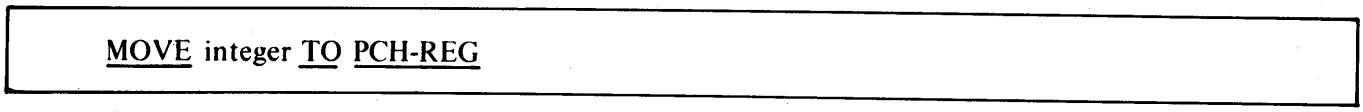
OPTION 8



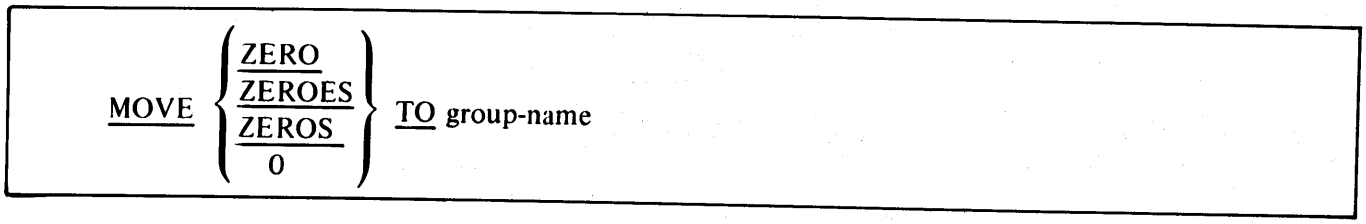
OPTION 9



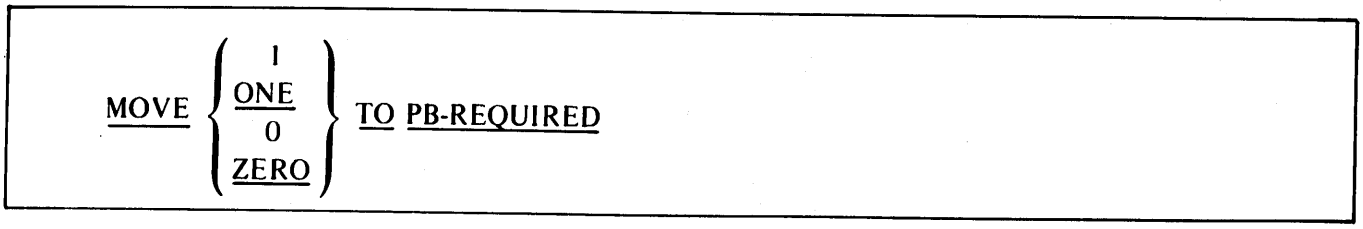
OPTION 10



OPTION 11



OPTION 12



OPTION 13

```
MOVE REMAINDER TO { ACCUMULATOR }
                  { data-name   }
```

Multiply

```
MULTIPLY { data-name-1 } BY { data-name-2  } [GIVING data-name-3] [ROUNDED]
         { literal     }    { ACCUMULATOR }
    [ON SIZE ERROR statements]
```

No-Op

```
NO-OP.
```

Note

OPTION 1

```
NOTE sentence.
```

OPTION 2

```
NOTE. paragraph
```

Open

```
OPEN { MEDIA-CLAMP [ { integer      } ]   [ INDEXED BY { ACCUMULATOR } ] }
     { CARRIAGE    [ { special-name } ]                 { data-name   } ]
```

Perform

```
PERFORM procedure-name.
```

Position

```
{ POSITION } TO { integer      } [ INDEXED BY { ACCUMULATOR } ]
{ POS      }    { special-name }               { data-name   }
```

Read

```
READ file-name
```

Red-Ribbon

```
RED-RIBBON
```

Round

```
ROUND { ACCUM      } TO PENCE
      { data-name  }
```

Select

```
                                                  { special-name }
SELECT { ALTERNATE STACKER                        { integer      }
       {                                          {              }
       { { SKIP   } FUNCTION TO { THROUGH }       {              }
       { { REPEAT } FUNCTION    { THRU    }       {              }
```

Stop Run

```
STOP RUN.
```

Subtract

OPTION 1

```
                { data-name-1 }                 { data-name-2 }
SUBTRACT        { integer     } FROM            { ACCUMULATOR } [GIVING data-name-3]
                { ACCUMULATOR }

[ROUNDED] [ON SIZE ERROR statements]
```

OPTION 2

```
SUBTRACT digit FROM ACCUMULATOR (integer) [ON SIZE ERROR statements]
```

Use

```
USE FOR { SUBROUTINE procedure-name }
        { PK-TABLE  table-name      }
```

PART II. DATA COMMUNICATIONS

Accept

```
ACCEPT alpha-file-data-name [FROM KEYBOARD]
```

If

OPTION 1

```
IF { XMT-RDY } [ { AND } { XMT-RDY } ] THEN { statements      } [ELSE statements]
   { RCV-RDY }   { OR  } { RCV-RDY }        { NEXT SENTENCE }
```

OPTION 2

```
IF { XMT-RDY      } THEN statements [ELSE statements]
   { RCV-RDY      }
   { DC-ERROR     }
   { POL-SEL-FLG  }
   { BREAK-FLG    }
   { LINE-ACTVY-FLG }
```

OPTION 3

```
IF { BUF-FULL  } THEN statements [ELSE statements]
   { BUF-EMPTY }
```

Locate

```
LOCATE file-name
```

Move

OPTION 1

```
MOVE alpha-file-data-name [FROM BUFFER] TO alpha-data-name
```

OPTION 2

```
MOVE { alpha-data-name } TO alpha-file-data-name [IN BUFFER]
     { alpha-literal   }
```

OPTION 3

```
MOVE { numeric-data-name } TO numeric-file-data-name [IN BUFFER]
     { literal           }
     { ACCUMULATOR       }
```

OPTION 4

```
MOVE numeric-file-data-name [FROM BUFFER] TO { ACCUMULATOR       }
                                             { numeric-data-name }
```

OPTION 5

```
                                 { SEND-ADR      }
                                 { RCV-ADR       }
        { ACCUMULATOR     }      { HDR-XMN-NO    }
MOVE    { alpha-data-name }  TO  { EXP-XMN-NO    }
        { alpha-literal   }      { SEND-XMN-NO   }
                                 { GRP-XMN-NO    }
                                 { BDCST-XMN-NO  }
```

OPTION 6

```
        { SEND-ADR      }
        { RCV-ADR       }
        { HDR-XMN-NO    }
MOVE    { EXP-XMN-NO    }  TO  { ACCUMULATOR     }
        { SEND-XMN-NO   }      { alpha-data-name }
        { GRP-XMN-NO    }
        { BDCST-XMN-NO  }
```

OPTION 7

```
        {  1   }
MOVE    { ONE  }  TO DC-ERROR
        {  0   }
        { ZERO }

        {  1   }
MOVE    { ONE  }  TO  { RCV-RDY }    [ { RCV-RDY } ]
        {  0   }      { XMT-RDY }    [ { XMT-RDY } ]
        { ZERO }

        {  1   }      { POL-SEL-FLG       }    [ { POL-SEL-FLG       }     ]
MOVE    { ONE  }  TO  { BREAK-FLG         }    [ { BREAK-FLG         } ... ]
        {  0   }      { LINE-ACTVTY-FLG   }    [ { LINE-ACTVTY-FLG   }     ]
        { ZERO }
```

OPTION 8

```
MOVE { TWO-WIRE-CNTL  } TO DATA-COMM
     { FOUR-WIRE-CNTL }
```

OPTION 9

```
MOVE @ab@ TO DATA-COMM
```

Read

```
READ file-name
```

Stop Machine

```
STOP MACHINE
```

Write

```
WRITE record-name
```

GLOSSARY

ACTUAL DECIMAL POINT	A decimal point used for "DISPLAY" purposes. When a numeric value is listed on a printed report, the decimal point will often appear as an actual printed character.
ALPHABETIC	With respect to data, consisting of one or more of the letters of the alphabet and/or one or more spaces, as used in this manual, the term does not include other non-numeric characters.
ALPHANUMERIC	Pertaining to a character set that contains both letters and numerals, and usually other characters.
ASSEMBLE	To prepare a machine language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes.
ASSEMBLER	A program that assembles.
ASSUMED DECIMAL POINT	The point within a numeric item at which the decimal point is assumed to be located. The assumed decimal point of an item is considered by the computer to be at the right (right-justified) unless otherwise specified by a FORMAT or PICTURE entry. It is used by the computer to align the value properly for calculation.
BUFFER	A storage device used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another.
CARD HOPPER	A device that holds cards and makes them available to a card feed mechanism.
CHARACTER	One of the set of elementary symbols which are arranged in groups to express information.
COBOL WORD	A word is created from a combination of not more than 30 characters, selected from the following: A through Z, 0 through 9, and the hyphen (-).
COMPILE	To prepare a machine language program from a computer program written in Series L/TC COBOL by making use of the overall logic structure of the program, or generating one or more machine instructions for each symbolic statement or both.
COMPILER	A program that compiles.
COMPILE TIME	The time at which a source program is translated into an object program through the use of a compiler.
CONDITION	1.) One of a set of specified values that a data item can assume. 2.) A simple conditional expression.
CONDITIONAL EXPRESSION	An expression which, taken as a whole, may be either true or false, in accordance with the rules given in Section 6 of this manual.
DATA DESCRIPTION	The entry or entries in the DATA DIVISION used to describe the characteristics of a data item. See Section 5.
DATA ITEM	A unit of information which may be identified by a name or by a combination of names and subscripts.

DATA-NAME	A name assigned to a data item by the programmer. See the rules for Data-Name formation in Section 2 of this manual.
DIGIT	One of the numerals 0 through 9.
DISPLAY	A visual representation of data.
FIGURATIVE CONSTANT	One of several constants which have been pre-defined in the Series L/TC compiler and which can be used in a program without description in the DATA DIVISION. A list of figurative constants will be found in Section 2 of this manual.
FORMAT	A predetermined arrangement of the types of characters of which a data item is composed.
HARDWARE	Physical equipment e.g., mechanical, magnetic, electrical, or electronic devices.
IMPERATIVE STATEMENT	A statement consisting of a verb and its operand(s) which expresses a complete unit of procedure.
INTEGER	In Series L/TC COBOL a non-negative whole number e.g., 13 is an integer while 13.5 is not.
JUSTIFICATION	The alignment of characters with respect to the left and right boundaries of data items.
KEY WORD	A word which is essential to the meaning and structure of a Series L/TC COBOL statement. In this manual key words are indicated in verb constructs and other entries by underscoring.
LEVEL	In the COBOL system, the status of one data item relative to another, showing whether one is to be treated as part of the other or whether they are unrelated, as specified in the rules governing level-numbers in Section 5 of this manual.
LEVEL INDICATOR	In the COBOL system, a symbol or level-number used in a DATA DIVISION entry to indicate level. For example, FD is a level indicator. (See the discussion of levels in Section 5 of this manual.)
LEVEL NUMBER	A numeric level indicator.
LITERAL	A character, or group of characters, used in a program to represent the value "literally" expressed. Thus, the literal "7" represents the value 7, whereas SEVEN is a name that could be used to represent the value 7. (See the rules governing literals in Section 2 of this manual.)
LOOP	A sequence of instructions that is executed repeatedly until a terminal condition prevails.
MACHINE LANGUAGE	An operation code that a machine can recognize and execute.
MEMORY	Main storage.
NON-NUMERIC	Not having a numeric value.
NUMERIC	Having a numeric value.
OBJECT PROGRAM	A program in machine language resulting from the translation of a source program by an assembler.

OFF-LINE	Pertaining to equipment or devices not under direct control of the central processing unit.
ON-LINE	Pertaining to equipment or devices under the direct control of the central processing unit.
OPERAND	That which is operated upon, the object of a verb or an operator.
OPERATOR	That which indicates the action to be performed on operands.
PARAGRAPH	A set of one or more sentences which direct the computer to perform some operation.
RECORD	A collection of related items of data grouped to be handled by an input/output device.
RELATIONAL EXPRESSION	An expression that describes a relationship between two terms, e.g., A. is less than B.
ROUNDED	To shorten a number, increasing the least significant remaining digit by 1 when the most significant digit of the part removed is greater than or equal to 5.
SECTION	One of the portions of a program defined as a section in the rules governing the format of a Series L/TC COBOL program e.g., CONFIGURATION SECTION and WORKING-STORAGE SECTION.
SENTENCE	A series of one or more statements, the last of which is terminated by a period.
SOURCE PROGRAM	A program written in the source language (i.e., in the Series L/TC COBOL language).
STATEMENT	A group of words which expresses a command. A statement consists of one or more verbs and their associated operands.
SUBSCRIPT	An integer used to identify a particular item in a list or table in accordance with the rules specified in Section 5 of this manual.
SWITCH	A mechanical, electromechanical or electronic device, built into a unit of equipment, which can be interrogated in order to select a course of action.
SYNTAX	(1) The structure of expressions in a language. (2) The rules governing the structure of a language.
TRUNCATION	The process of dropping one or more digits of a number, either at the left or the right, without altering any of the remaining digits. For example, the number 384.67 would become 384.6 when truncated one place at the right while it would become 384.7 when rounded correspondingly. (See Rounded.)
VERB	One of a selected list of words that specify one or more operations to be performed by the Series L/TC computer.
WORD	(1) A basic unit of language serving the same general purposes as words in other languages. (2) A subdivision of storage having a fixed size.
WORK AREA	A portion of storage in which a data item may be processed or temporarily stored. The term often refers to a place in storage used to retain intermediate results of calculation, especially those results which will not appear directly as output from the program.

SERIES L/TC RESERVED WORD LIST

ACCEPT
ACCEPTING
ACCESS
ACCUM
ACCUMULATOR
ACUM
ADD
ADVANCE
ALARM
ALIGN
ALPHA
ALTERNATE
AND
ARE
AREA
AREAS
ASSIGN
AT
AUTHOR
AUTO-READER

BDCST-XMN-NO
BOTH
BREAK-FLAG
BUFFER
BUF-EMPTY
BUF-FILL
BY

CARD-PCH
CARD-RDR
CARD-RDR-1
CARD-RDR-2
CARD-READER
CARD-READER-1
CARD-READER-2
CARRIAGE
CFLAG
CHARACTER
CHARACTERS
CHECK-DIGIT
CLOSE
CD-TABLE
COL
COLUMN
COMMA
CONFIGURATION
CONSOLE
CONTAINS
CONVERT
COUNT-REG
CTL-REG

DATA
DATA-COM
DATA-COMM
DATE-COMPILED
DATE-WRITTEN
D-C
DC-ERROR
DECLARATIVES
DELIMITER
DISPLAY
DIVIDE
DIVISION

EJECT
ELSE
ENABLE
END
END-OF-JOB
END-OF-LDGR
END-OF-PAGE
ENVIRONMENT
EQUAL
ERROR
EXIT
EXP-XMN-NO

FARTHING
FD
FILE
FILE-CONTROL
FILL
FILLER
FMT
FOR
FORMAT
FOUR-WIRE-CNTL
FROM
FUNCTION

GIVING
GO
GREATER
GRP-XMN-NO

HALF-PENNY
HALT
HDR-XMN-NO
HIGH

IDENTIFICATION
IF
IN
INDEXED
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
I-O-CONTROL
IS

KEYBOARD
KEYBOARD-PCH
KEYBOARD-PRNTR
K-REG
KYBRD-PRNTR-PCH

LDGR-ERR
LEDGER
LEFT
LENGTH
LESS
LIMIT-REG
LINE
LINE-ACTVY-FLG
LINES
LOCATE
LOW
LOW-TAPE

MACHINE
MASK
MEDIA-CLAMP
MFLAG
MODE
MOVE
MULTIPLY

NEGATIVE
NEXT

NFLAG
NO
NO-MEDIA
NON-ALIGN
NON-DECIMAL
NON-READ
NON-STERLING
NO-OP
NORMAL
NOT
NOTE
NUMERIC

OBJECT-COMPUTER
OC
OCCURS
OCK1
OCK2
OCK3
OCK4
OF
ON
ONE
OPEN
OR
OTHERWISE
OUT
OUTPUT

PB-FIRST-LINE
PB-FOLD
PB-LAST-LINE
PB-PRESENT
PB-REQUIRED
PC
PCH
PCH-ERR
PCH-REG
PENCE
PERFORM
PIC
PICTURE
PK-TABLE
PK1
PK2
PK3
PK4
PK5
PK6
PK7
PK8
PK9
PK10
PK11
PK12
PK13
PK14
PK15
PK16
PLACE
PLACES
POL-SEL-FLG
POS
POSITION
POSITIVE
POUNDS
P-REG
PREVIOUS-RIBBON
PRINTER
PRNTR
PRNTR-PCH
PROCEDURE
PROGRAM
PROGRAM-ID
PUNCH
PUNCH-OFF

QUOTE

RCV-ADR
RCV-RDY
RDR
RDR-COND
RDR-ERR
RDR-PCH
RDR-PRNTR
RDR-PRNTR-PCH
READ
READER
RECORD
RECORDING
RECORDS
REDEFINES
RED-RIBBON
REMARKS
REPEAT
RESERVE
RESETTING
RETRACT
RIGHT
ROUND
ROUNDED
RUN

SAME
SECTION
SECURITY
SELECT
SEND-ADR
SEND-XMN-NO
SENTENCE
SEQUENTIAL
SETTING
SFLAG
SHILLING
SIGN
SIZE
SKIP
SOURCE-COMPUTER
SPEC-FARTHING
SPECIAL-NAMES
SPEC-PENCE
SPROCKET-HOLES
STACKER
STANDARD
STOP
STRIPE
SUBROUTINE
SUBTRACT
SUPERVISOR
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8

TELLER-1
TELLER-2
THAN
THEN
TIMES
TO
TRUNCATED
TWO-WIRE-CNTL

UPON
USE

VA
VALUE

WITH
WORD
WORDS
WORK-AREA
WORKING-STORAGE
WRITE

XMT-RDY

ZERO
ZEROES
ZEROS

SERIES L/TC COBOL COMPILER ERROR MESSAGES

002 MISSING QUOTE ON CONTINUATION CARD
003 UNIDENTIFIED WORD
004 DUPLICATE NAME
005 DUPLICATE LABEL
006 INVALID VALUE CLAUSE
007 SUBJECT OF SAME AREA NOT PREVIOUSLY SELECTED
008 DUPLICATE DATA NAME
009 MISSING DIVISION HEADER
010 ELEMENT GREATER THAN 99 CHARACTERS
011 INVALID USE OF A RESERVED WORD
013 LITERAL POOL GREATER THAN 1000 CHARACTERS
014 FATAL SEQUENCE ERROR
015 VALUE LENGTH NOT EQUAL TO PICTURE LENGTH
017 MISSING PERIOD
018 FD ENTRY NOT SELECTED
019 VALUE CLAUSE OUT OF RANGE
020 INVALID LEVEL NUMBER
021 VALUE CLAUSE MUST BE PRECEDED BY A PICTURE OR A FORMAT CLAUSE
022 REDEFINED ITEM NOT PREVIOUSLY DEFINED
300 INVALID PICTURE CLAUSE CONSTRUCTION
301 ALPHA PICTURE CLAUSE FOLLOWING FORMAT
302 PICTURE CLAUSE GREATER THAN 15 CHARACTERS
303 FORMAT DECLARED ON ALPHA ITEM
304 INVALID FORMAT CLAUSE CONSTRUCTION
305 FORMAT CLAUSE GREATER THAN 15 CHARACTERS
306 INVALID OCCURS CLAUSE CONSTRUCTION
307 PICTURE TABLE SIZE EXCEEDED
310 DUPLICATE CLAUSE FOR DATA NAME
501 MISSING FILE NAME
502 MISSING RESERVED WORD WORK-AREA
503 MISSING OR INVALID HARDWARE NAME
504 INVALID CONSTRUCT
505 REDEFINES LENGTH EXCEEDED
506 MISSING SECTION HEADER
507 MISPLACED FD
508 MISSING DATA NAME

509 MISSING FD
510 01 LEVEL MUST BE GROUP ITEM
511 INVALID CLAUSE ON CD-TABLE ENTRY
512 MISSING 01 LEVEL
513 INVALID OR MISPLACED LEVEL 77
514 INVALID OCCURS OR REDEFINES ON LEVEL 77
515 INVALID DESCRIPTIVE CLAUSE ON GROUP ITEM
516 LEVEL 77 CANNOT BE GROUP ITEM
519 INVALID VALUE CLAUSE WITH OCCURS CLAUSE
520 VALUE CLAUSE WITH REDEFINES CLAUSE
522 REDEFINED ITEM CANNOT HAVE OCCURS CLAUSE
523 INVALID REDEFINES CLAUSE
601 INVALID VERB
602 MISSING OR MISSPELLED RESERVED WORD DECLARATIVES
603 INVALID USE STATEMENT
604 INVALID FIRST ARGUMENT
605 INVALID ENABLE STATEMENT UNDECLARED TABLE NAME
606 READ ADDRESS MUST BE FILE NAME
607 WRITE ADDRESS MUST BE RECORD NAME
608 INVALID OPEN/CLOSE STATEMENT
609 PROGRAM HAS NO STOP RUN
610 MISSING RESERVED WORD TO
611 INVALID STOP STATEMENT
612 MISSING RESERVED WORD SENTENCE
613 INVALID PROCEDURE NAME FOR PERFORM STATEMENT
614 MISSING RESERVED WORD FROM
615 INVALID HARDWARE NAME
616 INVALID PARAGRAPH NAME FOR GO TO STATEMENT
617 INVALID SECOND ARGUMENT
618 MISSING OR INVALID COLUMN ON SELECT
619 INVALID POSITION STATEMENT
620 INVALID FUNCTION ON SELECT
621 INVALID DISPLAY STATEMENT
622 MISSING RESERVED WORD EQUAL
623 INVALID REGISTER NAME
624 MISSING OR INVALID ACCUMULATOR SUBSCRIPT
625 INVALID THIRD ARGUMENT
626 INVALID ALPHA COMPARE – LENGTHS NOT EQUAL

627 GROUP ITEM SIZE GREATER THAN 99
628 INVALID STATEMENT TOO MANY DATA NAMES
629 LITERAL SUBSCRIPT GREATER THAN 99
630 VARIABLE MUST HAVE SUBSCRIPT
631 VARIABLE CANNOT HAVE SUBSCRIPT
632 INVALID SUBSCRIPT
634 MISSING PARENTHESIS
635 INVALID STATEMENT
636 INVALID OR MIXED MODE LIST
637 MIXED BOOLEAN OPERATORS
638 LIST LENGTH EXCEEDED
639 INVALID OPERATOR
640 MISSING PICTURE CLAUSE ON DISPLAY ITEM
642 MISPLACED EXIT PARAGRAPH MUST BE SEPARATE
643 MISPLACED EXIT USE ONLY WITH DECLARATIVES
644 MISSING PERIOD OR ELSE FOLLOWING GO TO
645 MISPLACED DECLARATIVES MUST BE FIRST PROCEDURE
646 INVALID STATEMENT IF NOT TERMINATED
647 MISSING RESERVED WORD END DECLARATIVES
648 MISSING RESERVED WORD DECLARATIVES
649 MISSING RESERVED WORD IF
650 INVALID INTEGER
651 INVALID LITERAL MUST BE SINGLE CHARACTER
652 INVALID PK STATEMENT
653 MISPLACED USE STATEMENT
654 INVALID ARGUMENT MUST BE LITERAL
655 PK-TABLE GREATER THAN 16
656 MISSING RESERVED WORD LESS
657 MISSING RESERVED WORD LEFT/RIGHT
658 ALPHA MOVE FIRST ARGUMENT GREATER THAN SECOND
659 INVALID LITERAL MUST BE ALPHA NUMERIC
660 MISSING RESERVED WORD BY
661 MISSING RESERVED WORD INTO
662 INVALID
663 PROGRAM MUST BEGIN WITH PARAGRAPH NAME
701 CARD FORMAT TABLE SIZE EXCEEDED
702 INVALID READ OR WRITE STATEMENT
703 FIXED ITEM FOLLOWING VARIABLE ITEM – DATA COMM

704 RECORD LENGTHS NOT EQUAL

705 SAME AREA FIRST FD MUST BE LARGEST

706 INVALID SAME AREA CLAUSE – DATA COMM ONLY

707 PROGRAM MUST HAVE LOCATE STATEMENT

708 HARDWARE PREVIOUSLY ASSIGNED

709 RECORD SIZE GREATER THAN HARDWARE SIZE

710 RECORD NAME MUST BE GROUP ITEM

711 SAME AREA MUST HAVE ALTERNATE AREA DECLARED

712 INVALID SAME WORK-AREA CLAUSE

901 SEQUENCE WARNING

902 MISSING WORK-AREA CLAUSE – NO WORK-AREA ASSUMED

903 INVALID OPTION CONSTRUCTION – $CARD

SAMPLE BILLING PROGRAM

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. LCOBOL-BASIC-BILLING-PROGRAM.
000300 AUTHOR. JDOE.
000400 INSTALLATION. L-COBOL REFERENCE MANUAL.
000500 DATE-WRITTEN. 8/29/69.
000600 DATE-COMPILED.  09/09/69
000700 REMARKS. THIS PROGRAM IS A BASIC BILLING PROGRAM USING A
000800          CONTINUOUS PIN FEED INVOICE(FORM NUMBER 1037124)
000900          WRITTEN TO USE THE SERIES  L  COBOL PROGRAMING
001000          LANGUAGE.
001100*****************************************************************
001200 ENVIRONMENT DIVISION.
001300 CONFIGURATION SECTION.
001400 SOURCE-COMPUTER. B-3500.
001500 OBJECT-COMPUTER. L-2000.
001600 SPECIAL-NAMES.
001700     SOLD-TO-LINE          IS LINE      7.
001800     SHIP-TO-LINE          IS LINE     13.
001900     RIBBON-LINE           IS LINE     19.
002000     FIRST-LINE            IS LINE     22.
002100     TYPING-POS            IS POSITION 24.
002200     TERMS-POS             IS POSITION 17.
002300     ORDER-NO-POS          IS POSITION 30.
002400     CUSTOMER-NO-POS       IS POSITION 42.
002500     SOLD-BY-POS           IS POSITION 53.
002600     SHIP-VIA-POS          IS POSITION 67.
002700     DATE-POS              IS POSITION 78.
002800     INV-NO-POS            IS POSITION 91.
002900     CODE-POS              IS POSITION 17.
003000     QTY-POS               IS POSITION 22.
003100     DESC-POS              IS POSITION 34.
003200     PRICE-POS             IS POSITION 56.
003300     UNIT-POS              IS POSITION 67.
```

APPENDIX F (cont'd)

```
003400      GROSS-POS            IS POSITION 71.
003500      DISCOUNT-POS         IS POSITION 82.
003600      NET-POS -POS         IS POSITION 91.
003700*****************************************************************
003800 DATA DIVISION.
003900 WORKING-STORAGE SECTION.
004000 77   DATE PICTURE IS X(11).
004100 77   INVOICE-NO PICTURE IS ZZZ,ZZ9.
004200 77   INVOICE-TOTAL PICTURE IS ZZ,ZZZ.99-.
004300 77   TOTAL-NET PICTURE IS ZZ,ZZZ.99-.
004400 77   TOTAL-DISCOUNT PICTURE IS ZZ,ZZZ.99-.
004500 77   TOTAL-TAX PICTURE IS ZZ,ZZZ.99-.
004600 77   TOTAL-ADD-ON PICTURE IS ZZ,ZZZ.99-.
004700 77   TOTAL-A/R PICTURE IS ZZ,ZZZ.99-.
004800 77   TOTAL-INV-DISCOUNT PICTURE IS ZZ,ZZZ.99-.
004900 01   WORKING-LOCATION-1.
005000      02  CODE PICTURE IS ZZ9.
005100      02  PRICE-IN REDEFINES CODE FORMAT IS 999999CM.
005200      02  PRICE REDEFINES CODE PICTURE IS Z,ZZZ.99.
005300      02  PRICE-C REDEFINES CODE FORMAT IS 99V9999.
005400      02  PRICE-M REDEFINES CODE FORMAT IS 9V99999.
005500      02  DISCOUNT-PERCENT REDEFINES CODE FORMAT S99V99 PIC ZZ.JZ-.
005600      02  DISCOUNT-FACTOR REDEFINES CODE FORMAT IS V9999.
005700      02 TAX-PERCENT REDEFINES CODE PICTURE IS ZZ.JZ.
005800      02 TAX-FACTOR REDEFINES CODE FORMAT IS V9999.
005900 01   WORKING-LOCATION-2.
006000      02 QUANTITY FORMAT IS S9999V999 PICTURE IS Z,ZZZ,JZZ-.
006100      02 GROSS REDEFINES QUANTITY PICTURE IS ZZ,ZZZ.99-.
006200      02 NET REDEFINES QUANTITY PICTURE IS ZZ,ZZZ.99-.
006300      02 ADD-ON REDEFINES QUANTITY FORMAT IS S9999999.
006400      02 ADD-ON-AMT REDEFINES QUANTITY PICTURE IS ZZ,ZZZ.99-.
006500 01   WORKING-LOCATION-3.
006600      02 DISCOUNT PICTURE IS ZZ,ZZZ.99-.
006700      02 TAX REDEFINES DISCOUNT PICTURE IS ZZ,ZZZ.99-.
006800*****************************************************************
```

```
006900 PROCEDURE DIVISION.
007000 DECLARATIVES.
007100     USE FOR PK-TABLE PROGRAM-KEYS.
007200     GO TO PK1-INVOICE-ROUTINE.
007300     GO TO PK2-INVOICE-SUBTOTAL.
007400     GO TO PK3-CLEAR-TOTALS-LOAD.
007500     GO TO START-INVOICE-LINE.
007600     GO TO PK5-LOAD-DATE-INV-NO.
007700     GO TO PK6-DISCOUNT-ROUTINE.
007800     GO TO PK7-TOTALING-ROUTINE.
007900     GO TO PK8-TAX-ROUTINE.
008000     GO TO PK9-CONTINUATION-PAGE.
008100     GO TO PK10-INVOICE-TOTAL.
008200 USE FOR SUBROUTINE TYPE-HEADING.
008300     POSITION TO TYPING-POS.
008400     ACCEPT 24 CHARACTERS FROM KEYBOARD-PRNTR.
008500     ADVANCE 1 LINE.
008600       IF OCK4 GO TO RIBBON-ROUTINE.
008700       IF OCK3 ADVANCE TO SHIP-TO-LINE
008800               POSITION TO TYPING-POS
008900               DISPLAY "SAME" UPON PRNTR
009000               GO TO RIBBON-ROUTINE.
009100       IF OCK2 GO TO SHIP-TO-ROUTINE.
009200 USE FOR SUBROUTINE TYPE-SHIP-TO.
009300     POSITION TO TYPING-POS.
009400     ACCEPT 24 CHARACTERS FROM KEYBOARD-PRNTR.
009500     ADVANCE 1 LINE.
009600     IF OCK2 OR OCK3 OR OCK4 GO TO RIBBON-ROUTINE.
009700 END DECLARATIVES.
009800*****************************************************************
009900 START-OF-PROGRAM.
010000     MOVE 51 TO LEFT LIMIT-REG.
010100     MOVE 18 TO RIGHT LIMIT-REG.
010200     MOVE 1 TO LEFT COUNT-REG.
010300 SELECT-ROUTINE.
```

```
010400      ENABLE PK1, PK3, PK5, PK7.
010500      ACCEPT FROM KEYBOARD.
010600      ALARM, GO TO SELECT-ROUTINE.
010700*****************************************************************
010800*           PK 1  INVOICE ROUTINE.                              *
010900*           PK 3  CLEAR TOTALS, LOAD DATE, INVOICE NO.          *
011000*           PK 5  LOAD DATE, INVOICE NO.                        *
011100*           PK 7  PRINT TOTALS.                                 *
011200*****************************************************************
011300 PK1-INVOICE-ROUTINE.
011400      ADVANCE TO SOLD-TO-LINE.
011500      ENABLE PK7.
011600      PERFORM TYPE-HEADING.
011700      PERFORM TYPE-HEADING.
011800      PERFORM TYPE-HEADING.
011900      PERFORM TYPE-HEADING.
012000 SHIP-TO-ROUTINE.
012100      ADVANCE TO SHIP-TO-LINE.
012200      PERFORM TYPE-SHIP-TO.
012300      PERFORM TYPE-SHIP-TO.
012400      PERFORM TYPE-SHIP-TO.
012500      PERFORM TYPE-SHIP-TO.
012600 RIBBON-ROUTINE.
012700      ADVANCE TO RIBBON-LINE.
012800      POSITION TO TERMS-POS.
012900        ACCEPT 11 CHARACTERS FROM KEYBOARD-PRNTR.
013000      POSITION TO ORDER-NO-POS.
013100        ACCEPT 10 CHARACTERS FROM KEYBOARD-PRNTR.
013200      POSITION TO CUSTOMER-NO-POS.
013300        ACCEPT 9 CHARACTERS FROM KEYBOARD-PRNTR.
013400      POSITION TO SOLD-BY-POS.
013500        ACCEPT 12 CHARACTERS FROM KEYBOARD-PRNTR.
013600      POSITION TO SHIP-VIA-POS.
013700        ACCEPT 9 CHARACTERS FROM KEYBOARD-PRNTR.
013800      POSITION TO DATE-POS.
```

```
013900       DISPLAY DATE UPON PRNTR.
014000     POSITION TO INV-NO-POS.
014100       DISPLAY INVOICE-NO UPON PRNTR.
014200       ADD 1 TO INVOICE-NO.
014300     ADVANCE TO FIRST-LINE.
014400     MOVE 1 TO RIGHT COUNT-REG.
014500     MOVE ZERO TO SW1.
014600*****************************************************************
014700 START-INVOICE-LINE.
014800     ENABLE PK2.    : PK2 SELECTS THE INVOICE TOTALING ROUTINE.
014900     POSITION TO CODE-POS. ACCEPT CODE FROM KEYBOARD.
015000     IF ACCUM ZERO, ALARM, GO TO START-INVOICE-LINE.
015100     DISPLAY CODE UPON PRNTR.
015200     POSITION TO QTY-POS. ACCEPT QUANTITY FROM KEYBOARD-PRNTR.
015300 TYPE-DESCRIPTION.
015400     POSITION TO DESC-POS.
015500        ACCEPT 21 CHARACTERS FROM KEYBOARD-PRNTR.
015600     IF OCK1, THEN GO TO INDEX-PRICE,
015700        ELSE ADVANCE BOTH 1 LINE.
015800           IF END-OF-PAGE, ALARM, MOVE ONE TO SW1.
015900        GO TO TYPE-DESCRIPTION.
016000 INDEX-PRICE.
016100     POSITION TO PRICE-POS.
016200        ACCEPT PRICE-IN FROM KEYBOARD.
016300        IF ACCUM CFLAG AND MFLAG, ALARM, GO TO INDEX-PRICE.
016400     DISPLAY ACCUM PRICE UPON PRNTR.
016500     POSITION TO UNIT-POS.
016600     IF ACCUM CFLAG THEN DISPLAY "C",
016700        MULTIPLY PRICE-C BY QUANTITY GIVING GROSS ROUNDED,
016800        GO TO PRINT-GROSS.
016900     IF ACCUM MFLAG THEN DISPLAY "M",
017000        MULTIPLY PRICE-M BY QUANTITY GIVING GROSS ROUNDED,
017100        GO TO PRINT-GROSS.
017200     MULTIPLY PRICE BY QUANTITY GIVING GROSS ROUNDED.
017300 PRINT-GROSS.
```

```
017400      POSITION TO GROSS-POS.
017500      DISPLAY GROSS UPON PRNTR.
017600      IF OCK1 THEN POSITION TO DISCOUNT-POS,
017700         ACCEPT DISCOUNT-PERCENT FROM KEYBOARD-PRNTR,
017800         MULTIPLY DISCOUNT-FACTOR BY GROSS GIVING DISCOUNT ROUNDED,
017900         ADD ACCUM TO TOTAL-DISCOUNT,
018000         SUBTRACT ACCUM FROM GROSS GIVING NET.
018100      POSITION TO NET-POS,
018200         DISPLAY NET UPON PRNTR.
018300         ADD ACCUM TO INVOICE-TOTAL.
018400         ADD ACCUM TO TOTAL-NET.
018500      ADVANCE BOTH 1 LINE.
018600         IF END-OF-PAGE THEN MOVE ONE TO SW1.
018700         IF SW1 THEN GO TO END-OF-PAGE-ROUTINE.
018800      GO TO START-INVOICE-LINE.
018900*****************************************************************
019000*        INVOICE SUBTOTAL, ADD-ON, DISCOUNT, TAX, TOTAL ROUTINE   *
019100*****************************************************************
019200*****************************************************************
019300 PK2-INVOICE-SUBTOTAL.
019400      ADVANCE LEFT 1 LINE.
019500      POSITION TO GROSS-POS.
019600      DISPLAY "SUBTOTAL" UPON PRNTR.
019700      POSITION TO NET-POS.
019800      DISPLAY INVOICE-TOTAL UPON PRNTR.
019900 SELECT-TAX-DISC-TOTAL.
020000      ADVANCE LEFT 1 LINE.
020100      POSITION TO DESC-POS. ENABLE PK6, PK8, PK10.
020200*****************************************************************
020300*             PK6  DISCOUNT-ROUTINE.                            *
020400*             PK8  TAX-ROUTINE.                                 *
020500*             PK10 INVOICE-TOTAL-ROUTINE.                       *
020600*****************************************************************
020700      ACCEPT 21 CHARACTERS FROM KEYBOARD-PRNTR.
020800      POSITION TO NET-POS.
```

```
020900      ACCEPT ADD-ON FROM KEYBOARD.
021000      DISPLAY ADD-ON-AMT UPON PRNTR. DISPLAY "-" ACCUM NEGATIVE.
021100      ADD ACCUMULATOR TO TOTAL-ADD-ON.
021200      ADD ACCUMULATOR TO INVOICE-TOTAL.
021300      GO TO SELECT-TAX-DISC-TOTAL.
021400*****************************************************************
021500 PK6-DISCOUNT-ROUTINE.
021600      DISPLAY "DISCOUNT" UPON PRNTR.
021700      POSITION TO DISCOUNT-POS.
021800      ACCEPT DISCOUNT-PERCENT FROM KEYBOARD-PRNTR. DISPLAY "%".
021900      MULTIPLY DISCOUNT-FACTOR BY INVOICE-TOTAL GIVING DISCOUNT.
022000      SUBTRACT ACCUMULATOR FROM INVOICE-TOTAL.
022100      ADD ACCUMULATOR TO TOTAL-INV-DISCOUNT.
022200      POSITION TO NET-POS.
022300      DISPLAY ACCUMULATOR DISCOUNT UPON PRNTR.
022400      GO TO SELECT-TAX-DISC-TOTAL.
022500*****************************************************************
022600 PK8-TAX-ROUTINE.
022700      DISPLAY "TAX" UPON PRNTR.
022800      POSITION TO DISCOUNT-POS.
022900      ACCEPT TAX-PERCENT FROM KEYBOARD-PRNTR. DISPLAY "%".
023000      MULTIPLY TAX-FACTOR BY INVOICE-TOTAL GIVING TAX.
023100      ADD ACCUMULATOR TO TOTAL-TAX.
023200      ADD ACCUMULATOR TO INVOICE-TOTAL.
023300      POSITION TO NET-POS, DISPLAY ACCUM TAX UPON PRNTR.
023400      GO TO SELECT-TAX-DISC-TOTAL.
023500*****************************************************************
023600 PK10-INVOICE-TOTAL.
023700      ADVANCE LEFT 1 LINE.
023800      POSITION TO GROSS-POS.
023900      DISPLAY "TOTAL" UPON PRNTR.
024000      POSITION TO NET-POS.
024100      DISPLAY INVOICE-TOTAL UPON PRNTR. ADD ACCUM TO TOTAL-A/R.
024200      MOVE ZEROS TO INVOICE-TOTAL.
024300      GO TO PK1-INVOICE-ROUTINE.
```

```
024400*****************************************************************
024500 PK7-TOTALING-ROUTINE.
024600     ADVANCE TO FIRST-LINE.
024700     POSITION TO 34.
024800     DISPLAY "TOTAL" UPON PRNTR.
024900     POSITION TO 40.
025000     DISPLAY "NET" UPON PRNTR.
025100     POSITION TO 71.
025200     DISPLAY TOTAL-NET UPON PRNTR.
025300     ADVANCE LEFT 2 LINES.
025400     POSITION TO 40.
025500     DISPLAY "DISCOUNT" UPON PRNTR.
025600     POSITION TO 71.
025700     DISPLAY TOTAL-DISCOUNT UPON PRNTR.
025800     ADVANCE LEFT 2 LINES.
025900     POSITION TO 40.
026000     DISPLAY "TAX" UPON PRNTR.
026100     POSITION TO 71.
026200     DISPLAY TOTAL-TAX UPON PRNTR.
026300     ADVANCE LEFT 2 LINES.
026400     POSITION TO 40.
026500     DISPLAY "ADD-ON" UPON PRNTR.
026600     POSITION TO 71.
026700     DISPLAY TOTAL-ADD-ON UPON PRNTR.
026800     ADVANCE LEFT 2 LINES.
026900     POSITION TO 40.
027000     DISPLAY "A/R" UPON PRNTR.
027100     POSITION TO 71.
027200     DISPLAY TOTAL-A/R UPON PRNTR.
027300     ADVANCE LEFT 2 LINES.
027400     POSITION TO 40.
027500     DISPLAY "INV-DISCOUNT" UPON PRNTR.
027600     POSITION TO 71.
027700     DISPLAY TOTAL-INV-DISCOUNT UPON PRNTR.
027800     ADVANCE TO SOLD-TO-LINE LINE.
```

```
027900      STOP RUN.
028000*****************************************************************
028100 PK3-CLEAR-TOTALS-LOAD.
028200      MOVE ZEROS TO INVOICE-TOTAL.
028300      MOVE ZEROS TO TOTAL-NET.
028400      MOVE ZEROS TO TOTAL-DISCOUNT.
028500      MOVE ZEROS TO TOTAL-TAX.
028600      MOVE ZEROS TO TOTAL-ADD-ON.
028700      MOVE ZEROS TO TOTAL-A/R.
028800      MOVE ZEROS TO TOTAL-INV-DISCOUNT.
028900*****************************************************************
029000 PK5-LOAD-DATE-INV-NO.
029100      ADVANCE TO FIRST-LINE.
029200      POSITION TO 40.
029300      DISPLAY "DATE    ".
029400      ACCEPT DATE FROM  KEYBOARD-PRNTR.
029500      ADVANCE LEFT 2 LINES.
029600      POSITION TO 40.
029700      DISPLAY "INV NO ".
029800      ACCEPT INVOICE-NO FROM KEYBOARD-PRNTR.
029900      GO TO PK1-INVOICE-ROUTINE.
030000*****************************************************************
030100 END-OF-PAGE-ROUTINE.
030200      ALARM, ENABLE PK2, PK4, PK9, ACCEPT FROM KEYBOARD.
030300      ALARM, GO TO END-OF-PAGE-ROUTINE.
030400*****************************************************************
030500*        PK2  IS NORMAL INVOICE TOTAL ROUTINE.                  *
030600*        PK4  TO CONTINUE NEXT LINE.                            *
030700*        PK9  SUBTOT INVOICE, GO TO CONTINUATION PAGE.          *
030800*****************************************************************
030900 PK9-CONTINUATION-PAGE.
031000      ADVANCE LEFT 1 LINE.
031100      POSITION TO GROSS-POS.
031200      DISPLAY "SUBTOTAL".
031300      POSITION TO NET-POS.
```

```
031400      DISPLAY INVOICE-TOTAL UPON PRNTR.
031500      ADVANCE TO SOLD-TO-LINE.
031600      POSITION TO 80.
031700      DISPLAY "CONTINUATION-PAGE".
031800      GO TO PK1-INVOICE-ROUTINE.
031900 END-OF-JOB.

   TOTAL NUMBER OF WARNINGS IS 0001
   COMPILE DATE 02/09/69  15:24  USING (OCT 69) SERIES-L COMPILER   PROGRAM ID IS LCOBOL
   ELAPSED TIME IS 0223 SECONDS.
   ELAPSED TIME IS TOTAL CLOCK TIME. NOT TIME CHARGEABLE TO COMPILATION.
   0319 SYMBOLIC RECORDS COMPILED AT 085 RECORDS PER MINUTE.
```

ALPHABETICAL INDEX

ALPHABETICAL INDEX (cont'd)

1044781

12-69

Printed in U.S. America