Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 1044781-003 Date: September 28, 1970
Publication Title: Burroughs L/TC COBOL Reference Manual

Other Affected Publications: None

Supersedes: N/A

Description

This PCN incorporates changed and/or additional information for the following areas: File Control, PICTURE, Procedural Constructs, Appendixes A, B and D.

Replace	Add
4-3 and 4-5	6-39 through 6-65
5-5	B-17 through B-23
6-7 through 6-37	
7-5	
A-1	
B-1 through B-15	
D-1	

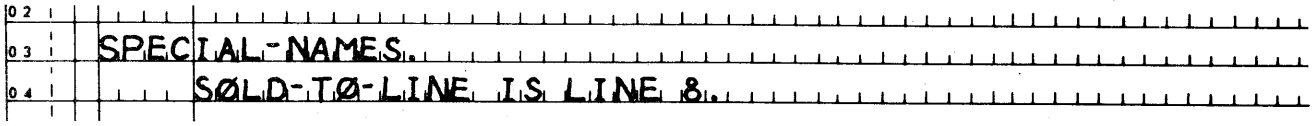
Retain this PCN as a record of change made to this basic publication.

Printed in U.S. America

The construct of this paragraph is:

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

Example:



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 identify each file, to specify input-output hardware assignment and to specify work areas and buffers for input and output data. The paragraph begins:

```
[FILE - CONTROL]
```

The appropriate option or options below may then be chosen. It is necessary that the OBJECT COMPUTER be equipped with appropriate firmware i.e., source programs which SELECT the CARD-READER must have their object programs run on computers equipped with 80-column card firmware sets, source programs which SELECT DATA – COMM IN or DATA-COMM OUT must utilize data communications firmware with their object programs. Firmware sets exist which allow both 80-column card and data communications operations. Each of the options below may be used only once per source program. These options are:

OPTION 1

```
[SELECT file-name ASSIGN TO CARD-READER {USE / NO} WORK - AREA]
```

This option ASSIGNS the chosen file-name to the card reader buffer in main memory. The use of the READ construct in the PROCEDURE DIVISION will cause an 80-column card to be read and the data stored in a character mode in the card reader buffer.

The program must specify either USE WORK-AREA or NO WORK-AREA. The USE WORK-AREA clause appears when the data is to be reformatted from character mode to main memory fields. These

fields are determined by the DATA RECORD definitions in the DATA DIVISION (see page 5-2). The FILL verb is used in the PROCEDURE DIVISION to transfer the data from the card reader buffer and reformat the data into individual main memory fields.

The NO WORK-AREA clause indicates that the data will be accessed directly from the card reader buffer. The FILL verb is not used for this case.

The nature of the card punch peripheral device excludes it from file control.

OPTION 2

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

This option ASSIGNS the chosen file-name to the data communication input buffer. The RESERVE ALTERNATE AREA clause will cause a working record area to be used along with the data communications input buffer. See READ page 60.

The ACCESS MODE IS SEQUENTIAL clause is used to specify that the DATA-COMM IN file will be accessed in a serial manner.

This clause can be used only if the fields of the buffer are all fixed length.

The ACCESS MODE IS SEQUENTIAL clause will prevent the compiler from generating field pointers and therefore save code.

The USE WORK-AREA clause or NO WORK-AREA clause must be specified.

The USE WORK-AREA clause signifies that the data received will be accessed from individual main memory fields. The FILL verb must be used in the PROCEDURE DIVISION to accomplish the transferring and reformatting of data from the buffer (or alternate area if specified) to the work-area. The main memory fields are determined by the DATA RECORD definitions within the DATA DIVISION.

The NO WORK-AREA clause indicates that the data will be accessed directly from the data communications input buffer.

OPTION 3

```
SELECT file-name ASSIGN TO DATA-COMM OUT [RESERVE ALTERNATE AREA]
{USE} WORK-AREA
{NO }
```

This option ASSIGNS the chosen file-name to the data communications output buffer. The RESERVE ALTERNATE AREA clause will cause a working record area to be used along with data communications output buffer. See WRITE page 6-60.

Either the USE WORK-AREA or NO WORK-AREA clause must be used. The USE WORK-AREA clause specifies that the output data will be accessed from individual main memory fields. The NO WORK-AREA clause signals the compiler that the data will be accessed directly into data communications output buffer (or alternate area if declared, accessing is in only a serial manner.)

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.

```
4   6 7 8   11 12      16      20      24      28      32      36      40      44      48      52      56      60
01      ENVIRØNMENT DIVISIØN.
02      CØNFIGURATIØN SECTIØN.
03      SØURCE-CØMPUTER. B-3500.
04      ØBJECT-CØMPUTER. L-2101.
05      SPECIAL-NAMES.
06          TØTAL-PØS IS PØSITIØN 99.
07          REMARKS-CØL IS CØLUMN 68.
08          TØTAL-LINE IS LINE 35.
09      INPUT-ØUTPUT SECTIØN.
10      FILE-CØNTRØL.
11          SELECT ØN-LINE-IN ASSIGN TØ DATA-CØMM IN
12              RESERVE ALTERNATE AREA USE WØRK-AREA.
13          SELECT ØN-LINE-ØUT ASSIGN TØ DATA-CØMM ØUT.
14          SELECT CARD-DATA-IN ASSIGN TØ CARD-READER
15              USE WØRK-AREA.
16      I-Ø-CØNTRØL.
17          SAME AREA FØR ØN-LINE-IN, ØN-LINE-ØUT.
18          SAME WØRK AREA FØR ØN-LINE-IN, ØN-LINE-ØUT,
19              CARD-DATA-IN.
```

Figure 4-1 Sample Coding for Environment Division

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:

```
19
20   02  DAILY-TOTALS  PC 9999 OCCURS 20 TIMES.
```

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:

{PC / PIC / PICTURE} 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:

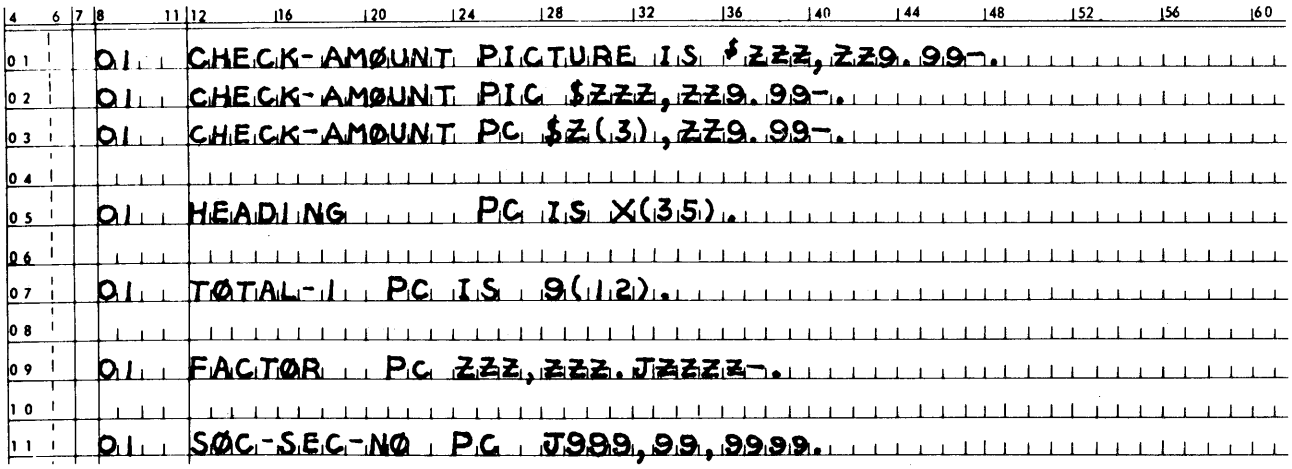
1. The letter J in the MSD position of a picture indicates suppress punctuation.
2. The letter J to the left of the decimal but not in the MSD position indicates reinitiate zero suppression and suppress for one digit (single digit zero suppress). The single digit zero suppress may be continued to the 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".
15. The letter S indicates allow reverse entry.

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 45
NAME	XXXX	NAME
CUSTOMER	X(8)	CUSTOMER
0000000004891723	$ZZZ,ZZZ.99	$48,917.23
0000000000012345	999999	012345

SPECIAL HARDWARE NAMES FOR BASIC CONSTRUCTS

ACCUMULATOR (ACCUM)	Refers to a special numeric word.
KEYBOARD	Refers to entry of data through the KEYBOARD, printing is not provided.
KEYBOARD-PRNTR	Refers to entry of data through the KEYBOARD, printing is provided.
KEYS	Refers to non-entry of data through the KEYBOARD.
PRNTR	Refers to printing of data.

PROCEDURAL CONSTRUCTS

Since the compiler provides constructs for use with all the standard firmware sets, and programs utilize only a single firmware set; the constructs are categorized by firmware type. These are:

- Part A Basic constructs
- Part B Paper Tape I/O constructs
- Part C 80-Column Card I/O constructs
- Part D Data Communications constructs
- Part E Check Digit constructs
- Part F Sterling capabilities
- Part G TC 700 constructs

It then becomes the programmers responsibility to utilize the "PARTS" of the procedural constructs which apply to the particular problem. For example, a program requiring 80-column card I/O would apply the constructs discussed in Parts A, C.

PART A: BASIC VERBS AND CONSTRUCTS

This part of the Procedural Constructs discusses the L/TC COBOL constructs which apply to any of the available firmware sets.

Accept

The use of this construct is to allow entry of alpha or numeric data through the appropriate keyboard.

The ACCEPT verb has six options:

OPTION 1

```
ACCEPT alpha-data-name [FROM {KEYBOARD      }]
                             {KEYBOARD-PRNTR}
```

Option 1 is used to ACCEPT alpha data into memory from the alpha keyboard. The number of characters ACCEPTED is equal to the PICTURE size. The ACCUMULATOR is not disturbed.

ACCEPT

Example:

```
11 12   16   20   24   28   32   36   40   44   48   52   56   60   64
    ACCEPT REMARKS FROM KEYBOARD-PRNTR.
```

OPTION 2

ACCEPT integer CHARACTERS [FROM KEYBOARD-PRNTR]

Option 2 is used to type integer number of characters from the alpha keyboard. The typed data is not stored in memory. The integer may range between 1 and 99, inclusive.

The ACCUMULATOR is not disturbed.

Example:

```
10
11      ACCEPT A CHARACTERS FROM KEYBOARD-PRNTR.
12
```

OPTION 3

ACCEPT numeric-data-name [FROM {KEYBOARD / KEYBOARD-PRNTR}]

Option 3 is used to ACCEPT numeric data through the numeric keyboard. The decimal key factor is provided by the FORMAT or PICTURE clause associated with the numeric data-name. The Reverse entry (negative number) key, C key, and M key are activated by the appropriate FORMAT symbols. The Reverse entry key can be activated by the PICTURE symbol S.

The indexed data will be contained in memory and the ACCUMULATOR.

Example:

```
06
07      ACCEPT STR-NUM FROM KEYBOARD-PRNTR.
```

OPTION 4

ACCEPT INTO ACCUMULATOR data-name [FROM {KEYBOARD / KEYBOARD-PRNTR}]

Option 4 functions the same as option 3 except the indexed data is not stored in memory and remains in the ACCUMULATOR.

Example:

```
08    ACCEPT INTØ ACCUMULATØR INVØICE-NUM.
```

OPTION 5

ACCEPT INTO ACCUMULATOR FROM KEYBOARD

Option 5 is used to ACCEPT a maximum of 15 numeric digits into the ACCUMULATOR from the numeric keyboard.

Example:

```
17
18    ACCEPT INTØ ACCUMULATØR FRØM KEYBØARD.
```

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.

OPTION 6

ACCEPT FROM KEYS

Option 6 will halt machine operation until a PK or OCK selection is made. No data (alpha or numeric) may be entered. This construct will generally appear after the ENABLE statement.

Example:

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

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

Add

The ADD verb will algebraically add two numeric data items. Refer to page 6-6 for a discussion of arithmetic.

There are 7 options.

ADD

OPTION 1

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

If option 1 is used, the two operands will be added and the sum stored in the second operand. Automatic decimal alignment does occur. The ACCUMULATOR contains operand 1 after execution. See Page 6-6 for an explanation of ON SIZE ERROR.

Example:

```
ADD INV-TOT TO GRAND-TOT ON SIZE ERROR
    GO TO SHIFT-MODE-1.
```

OPTION 2

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

Option 2 will add the first operand to the second operand, storing the sum in the third operand. The sum is decimally aligned according to the PIC/FMT of numeric-data-name-3. ROUNDED generates correct results only when numeric-data-name-3 has fewer decimal places than the other operands.

The ACCUMULATOR and numeric-data-name-3 contain identical values after option 2 is executed.

See page 6-6 for an explanation of ON SIZE ERROR.

Example:

```
ADD 1.4 INCREM-NUM GIVING NEW-INCREM.
```

OPTION 3

```
ADD ACCUMULATOR TO numeric-data-name [ON SIZE ERROR statements]
```

Option 3 will add the ACCUMULATOR to the specified numeric-data-name, storing the sum in the numeric data-name. The ACCUMULATOR remains unchanged. Automatic decimal alignment does not occur.

See page 6-6 for an explanation of ON SIZE ERROR.

OPTION 4

```
ADD ACCUMULATOR numeric-data-name-1 GIVING numeric-data-name-2 [ROUNDED]
    [ON SIZE ERROR statements]
```

Option 4 will add the ACCUMULATOR and specified numeric-data-name, storing the result in numeric-data-name-2. The sum is decimally aligned to numeric-data-name-2.

ROUNDED generates correct results only when numeric-data-name-1 has more decimal places than numeric-data-name-2.

The ACCUMULATOR and the third operand contain identical results after execution of Option 4.

See page 6-6 for an explanation of ON SIZE ERROR.

Example:

```
ADD ACCUMULATOR TOTAL GIVING GRAND-TOTAL.
```

OPTION 5

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

Option 5 provides for adding operand 1 to the ACCUMULATOR. The sum is stored in the ACCUMULATOR. Automatic decimal alignment does not occur.

See page 6-6 for an explanation of ON SIZE ERROR.

OPTION 6

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

Option 6 will add operand 1 to the ACCUMULATOR and store the sum in the third operand. The sum is decimally aligned to numeric-data-name-2.

ROUNDED generates correct results only when numeric-data-name -2 contains more decimal positions than the first operand.

The ACCUMULATOR and third operand contain identical values after execution of Option 6.

See page 6-6 for an explanation of ON SIZE ERROR.

ADVANCE

OPTION 7

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

If Option 7 is used the "digit" will be added to the "integer" position of the ACCUMULATOR. The digit may be one of 1-9, while the integer may be one of 1-15.

Example:

```
06
07     ADD 1 TØ ACCUMULATØR(4).
08
```

If in the above example the ACCUMULATOR contained 000000000054321 prior to execution, the ACCUMULATOR will contain 000000000055321 after execution.

Advance

The use of this verb is to space the forms in the forms handler.

In the constructs if LEFT, RIGHT or BOTH is not specified, LEFT is assumed.

There are 2 options:

OPTION 1

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

This option will ADVANCE the selected platen integer number of lines. The integer may not be zero.

If the INDEXED BY clause is used, the number of lines advanced will be equal to the sum of the integer and second operand. There is no effect upon the ACCUMULATOR unless the INDEXED BY clause is used. With this clause the ACCUMULATOR will contain the value of the second operand.

Example:

```
09
10     ADVANCE 1 LINE.
```

OPTION 2

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

Option 2 will ADVANCE the platen TO the selected line. Special names are discussed on page 4-2.

If the INDEXED BY clause is utilized, the platen will advance TO the sum of the integer and the second operand. There is no effect upon the ACCUMULATOR unless the INDEXED BY clause is used. With this clause the ACCUMULATOR will contain the value of the second operand.

Example:

```
ADVANCE RIGHT TO LINE-POS INDEXED BY ACCUMULATOR.
```

Alarm

The ALARM verb will ring the alarm once.

The construct is:

```
ALARM
```

The ACCUMULATOR is undisturbed.

Close Forms Handler

The use of this construct is to provide for closing the forms handler.

The construct is:

```
CLOSE TRANSPORT
```

The forms handler is automatically closed under the following circumstances:

1. The execution of a DISPLAY construct.
2. The ACCEPTING of alpha data through the keyboard.

Display

The use of this verb is to provide for printing of data from memory.

There are 5 options.

OPTION 1

```
DISPLAY { alpha-data-name } [UPON PRNTR]
        { "alpha-literal"  }
```

DISPLAY

Option 1 will print the contents of the specified alpha-data-name beginning at the current print ball position. A maximum of 99 characters can be DISPLAYED by the DISPLAY "alpha-literal" clause. The ACCUMULATOR is undisturbed.

Example:

```
015 |
016 |     DISPLAY "SUBTØTAL" UPØN PRNTR.
```

OPTION 2

```
DISPLAY { "character" } [PREVIOUS-RIBBON]
        { QUOTE      }
```

Option 2 will DISPLAY upon the printer the indicated character at the current print ball position. If the PREVIOUS-RIBBON clause is used the character (or quote) will be printed using the same color ribbon as the last DISPLAY.

The figurative constant QUOTE will DISPLAY a quotation mark.

The ACCUMULATOR is not affected.

Example:

```
001 |     DISPLAY "/".
004 |     DISPLAY QUØTE.
```

OPTION 3

```
DISPLAY { "character" } { numeric-data-name } { NEGATIVE }
        { QUOTE      } { ACCUMULATOR      } { POSITIVE }
```

Option 3 will DISPLAY the chosen character only if the numeric-data-name or ACCUMULATOR is NEGATIVE or POSITIVE (as selected).

The character will be DISPLAYED at the current print ball position.

Example:

```
001 |     DISPLAY "-" ACCUMULATØR NEGATIVE.
002 |
```

OPTION 4

```
DISPLAY numeric-data-name [UPON PRNTR]
```

Option 4 will print the contents of the numeric data-name as dictated by the assigned PICTURE. Printing is right justified in the field which begins at the current print ball position and extends to the right the number of places equal to the number of characters in the PICTURE clause.

The ACCUMULATOR will contain the printed numeric-data-name.

Example:

```
17    DISPLAY TØTAL.
```

OPTION 5

DISPLAY ACCUMULATOR [(integer)] numeric-data-name [UPON PRNTR]

Option 5 allows for printing all or part of the ACCUMULATOR according to the PICTURE of the numeric-data-name.

The (integer) option will DISPLAY the integer rightmost digits of the ACCUMULATOR. If the (integer) is omitted, the ACCUMULATOR will be DISPLAYED as dictated by the PICTURE of the numeric-data-name.

The ACCUMULATOR is unchanged.

Example:

```
09
10    DISPLAY ACCUMULATØR INV-NUM.
```

In the above example if the ACCUMULATOR contained 000000098765321 and INV-NUM was given a PICTURE of 999, the printed result would be: 321.

Divide

The DIVIDE verb provides the ability to divide.

The construct is:

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

If both operand 1 and operand 2 have the same sign, the quotient is positive. Unlike signs will generate a negative quotient.

ENABLE

Without the GIVING option this construct divides numeric-data-name-1 into numeric-data-name-2 storing the quotient in numeric-data-name-2.

With the GIVING option the quotient is stored in numeric-data-name-3.

The ACCUMULATOR will also contain the quotient after execution of the DIVIDE construct.

See page 6-6 for a discussion of ON SIZE ERROR.

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

Example:

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

The remainder is stored in a special memory location.

See page 6- 26.

Enable

The use of the ENABLE verb is to enable selected PK keys at the next keyboard operation. When this construct is executed, the selected PK lights will be lit.

The construct is as follows:

ENABLE [table-name] PK1 ... PK24

The table-name option allows for activating multiple PK-TABLES. If only one PK-TABLE is defined in the DECLARATIVES, then a table-name need not be coded. If more then one TABLE is utilized, the table-name need only be included when the referenced table changes. The number of PK keys vary from machine style to machine style.

This construct is generally used in conjunction with the ACCEPT FROM KEYS construct, as the ENABLE construct does not automatically provide for a keyboard halt.

Attempting to enable a PK greater than the table size, results in a syntax error.

The ACCUMULATOR is undisturbed.

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.

This construct must be the last statement in every program.

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

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.
```

Go To

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

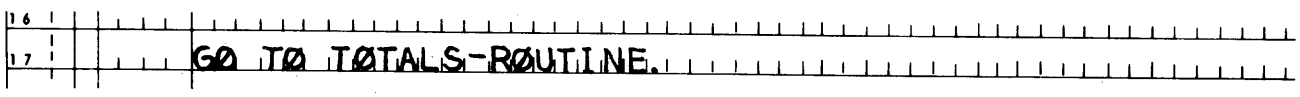
The construct is:

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

The paragraph-name can be any paragraph-name within the PROCEDURE DIVISION, but not a procedure (subroutine) name. Caution must be used when GOing TO a paragraph-name within a subroutine because if execution continues to the end of the subroutine the automatic subroutine EXIT would be executed and incorrect program execution could result.

This construct has no effect on the ACCUMULATOR.

Example:



If

The function of this verb is to control the sequence of program execution depending upon the TRUE or FALSE value of certain statements.

IF

These statements are subdivided into seven major categories:

1. relative tests
2. ACCUMULATOR tests
3. ZERO tests
4. forms limit test
5. ACCUMULATOR flag tests
6. SWITCH tests
7. OCK flag tests

See Execution of CONDITIONAL SENTENCES page 6-2.

RELATIVE TESTS

These tests compare data-names and/or literals with regard to their numerical value.

OPTION 1

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

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

This option compares two numeric items and transfers control according to the TRUE or FALSE condition of the option.

The ACCUMULATOR content is unknown.

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.

OPTION 2

```
IF { alpha-data-name     } IS { GREATER THAN   ( > )     } { alpha-data-name     }
   { non-numeric-literal }   { LESS THAN      ( < )     } { non-numeric literal }
                             { EQUAL TO       ( = )     }
                             { NOT EQUAL TO   ( NOT= )  }

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

See page 6-5 for a discussion of non-numeric literals. Program execution continues according to the TRUE/FALSE condition of Option 2.

The alpha data-names must be of equal word length.

The ACCUMULATOR content is unknown.

ACCUMULATOR TESTS

These tests examine the contents of the ACCUMULATOR.

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 1-15. Integer-2 may be any integer from 1 to 9.

The ACCUMULATOR is undisturbed.

Example:

```
18
19     IF ACCUMULATOR (3) LESS THAN 1 THEN
20       NEXT SENTENCE ELSE GO TO START.
```

OPTION 2

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

Option 2 will test to see if a SIZE ERROR condition exists and transfer control accordingly. A discussion of SIZE ERROR may be found on page 6-6.

IF

ZERO TESTS

This category of tests examines a data-name or ACCUMULATOR in regard to a ZERO value.

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

If the data-name or ACCUMULATOR is ZERO then the statement is TRUE.

Example:

```
IF ACCUMULATOR IS ZERO GO TO NUCARD.
```

FORMS LIMIT TEST

This test determines if the value of the forms limit register has been reached.

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

The above construct will have a TRUE value only on the line advance after the respective count register and limit register are equal. At all other line advances the construct will have a FALSE value.

This construct is used to control the forms in the forms handler.

The ACCUMULATOR is undisturbed.

See page 6-27, 6-28.

Example:

```
IF END-OF-PAGE
    GO TO NEW-FORM
ELSE
    GO TO INDEX-ACCOUNT-NUMBER.
```

ACCUMULATOR FLAG TESTS

This category of the IF verb tests the condition of the ACCUMULATOR FLAGS.

The four ACCUMULATOR FLAGS are:

1. NFLAG – negative condition flag
2. SFLAG – special flag

3. CFLAG – per hundred flag
4. MFLAG – per thousand flag

The ACCUMULATOR FLAGS are turned ON when:

1. Whenever the Reverse Entry (NFLAG), C (CFLAG) or M (MFLAG) key is depressed the associated ACCUMULATOR FLAG is turned ON. The others are not affected. The desired keys must be enabled with the appropriate FORMAT or PICTURE. See ACCEPT Option 3.

   The SFLAG cannot be turned on through keyboard input.
2. The use of the MOVE verb Option 1.
3. A code read through external media.

```
IF { numeric-data-name } { NFLAG }                [ { AND } { NFLAG }      ]      THEN { statements      }
   { ACCUMULATOR      } { SFLAG }                [ { OR  } { SFLAG } ... ]           { NEXT SENTENCE }
                        { CFLAG }                                { CFLAG }
                        { MFLAG }                                { MFLAG }

   [ ELSE statements ]
```

If the selected ACCUMULATOR FLAG is ON, the statement will have a TRUE value. With the logical connector AND all the specified flags must be ON in order for the statement to be TRUE. The logical connector OR will allow the statement to be TRUE whenever only one of the selected flags is ON. OR and AND cannot be mixed in the same statement.

Example:

```
11
12      IF ACCUMULATØR CFLAG ØR MFLAG NEXT SENTENCE
13        ELSE GØ TØ SETUP.
```

SWITCH TESTS

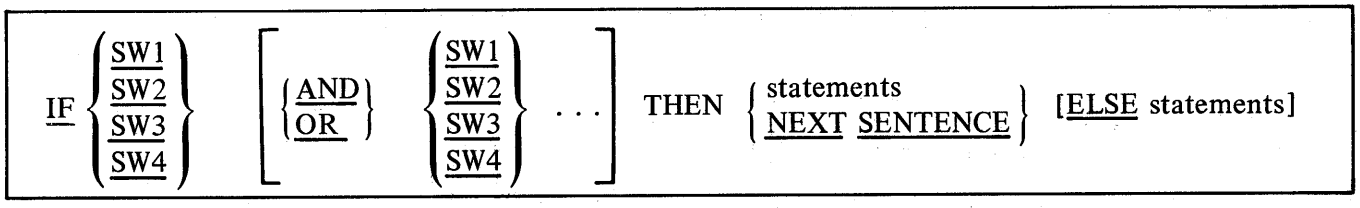
This category of the IF verb tests to see if any of the eight internal program switches are ON.

These switches are automatically OFF at the start of every program.

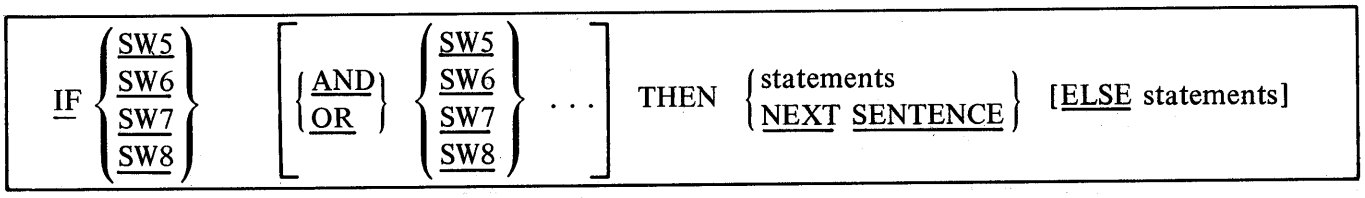
The program switches are turned ON in two ways:

1. Move Option 8 page 6-26.
2. Reading of certain codes through a peripheral device. See appendix A, table A-3.

Group A Switches



Group B Switches



If the selected SWITCH is ON, the statement will have a TRUE value. With the logical operator AND all the selected SWITCHES must be ON in order for the statement to be true. The logical operator OR requires only that one of the selected switches be ON in order for the statement to be true. Switches from GROUP A (SW1, SW2, SW3, SW4) may not be mixed with switches from GROUP B (SW5, SW6, SW7, SW8).

The ACCUMULATOR is undisturbed.

OPERATION CONTROL KEYS (OCK) FLAG TESTS

This category of the IF verb enables the programmer to determine if an OCK FLAG is ON.

Each OCK (1,2,3,4) has an associated OCK FLAG (1,2,3,4).

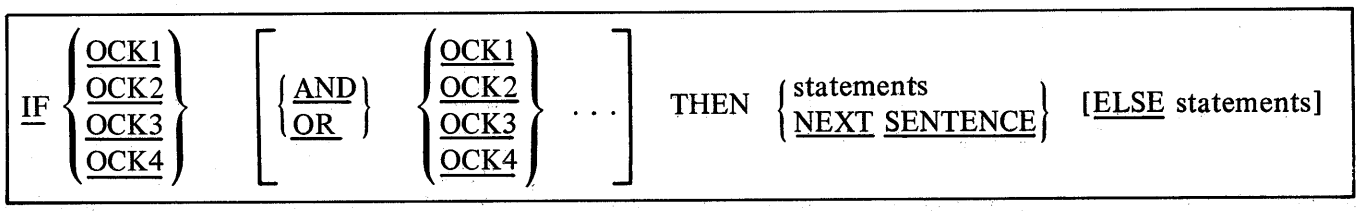
The OCK FLAGS may be ON because:

1. An OCK was depressed to terminate a keyboard entry. In this case the OCK FLAG associated with the OCK will be turned ON, all other OCK FLAGS are turned OFF.

   If the keyboard entry was terminated by a PK (program key) all OCK FLAGS are turned OFF.

2. The MOVE verb Option 9.
3. A code read from external media. See Appendix A, table A-2.

The construct is as follows:



If the specified OCK FLAG is ON, the statement is TRUE. The logical operator AND requires that all the specified OCK FLAGS be ON in order for the statement to be TRUE. If anyof the specified OCK FLAGS are ON when OR is used, the statement will be TRUE.

AND and OR cannot be mixed in the same statements.

The ACCUMULATOR is undisturbed.

Example:

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

Move

The use of this verb is to transfer data from one area of memory to another area of memory, to load data into memory locations, to set forms control Limit and Count Registers, to turn switches and flags ON and OFF and to isolate parts of words through shifting the ACCUMULATOR.

This construct has twelve options.

OPTION 1

MOVE { alpha-data-name / non-numeric literal } TO alpha-data-name

Option 1 will transfer operand 1 to operand 2. This first operand must be less than or equal to the second operand in regard to word size.

The ACCUMULATOR content is unknown after execution of option 1.

Example:

```
MØVE REMARKS-1 TØ REMARKS-2.
```

OPTION 2

MOVE { numeric-data-name / integer / ZERO / ACCUMULATOR } TO numeric-data-name

Option 2 will MOVE numeric-data from operand 1 to a numeric-data-name. Operand 1 is not changed.

The NFLAG, SFLAG, CFLAG and MFLAG conditions associated with operand 1 will also be MOVED to operand 2.

With the integer and ZERO clauses the FLAG conditions are OFF.

The ACCUMULATOR contains the prior contents of operand 1 (including FLAG conditions if the CFLAG was on in Operand 1, it is now on in the ACCUM).

Example:

```
03
04
05     MOVE 1 TO COUNTER.
```

OPTION 3

```
MOVE { numeric-data-name } TO ACCUMULATOR
     { integer          }
     { ZERO             }
```

Option 3 will MOVE operand 1 to the ACCUMULATOR. Operand 1 is not changed.

FLAG conditions are MOVED.

With the integer and ZERO clauses all ACCUMULATOR FLAGS are OFF.

Example:

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

This will cause the ACCUMULATOR to be cleared.

OPTION 4

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

Option 4 will insert the digit (0-9) into the integer (1-15 right to left) position of the ACCUMULATOR.

The FLAG conditions are not changed.

Example:

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

OPTION 5

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

Option 5 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     MØVE ACCUMULATØR (5) TØ PRØD-CØDE.
```

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     MØVE ACCUMULATØR (6) TØ ACCUMULATØR (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:

```
01
02     MØVE ACCUMULATØR (8 2) TØ ITEM-CØDE.
03
```

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:

```
04
05     MØVE ACCUMULATØR (10 5) TØ ACCUMULATØR(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.

MOVE

OPTION 6

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

Option 6 will transfer the DIVISION REMAINDER to operand 2.

Example:

```
06
07     MOVE REMAINDER TO ACCUMULATOR.
```

OPTION 7

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

Option 7 is used to turn ON and OFF the ACCUMULATOR FLAGS. The MOVE ZERO clause will turn the specified FLAG OFF, while MOVE ONE will turn the specified FLAG ON.

Example:

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

OPTION 8

Group A

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

Group B

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

Option 8 will turn ON or OFF the internal program switches specified.

The MOVE ZERO clause will turn the specified FLAG OFF, while MOVE ONE will turn the specified FLAG ON.

Switches from Group A (SW1, SW2, SW3, SW4) cannot be mixed from Group B (SW5, SW6, SW7, SW8).

The ACCUMULATOR is not changed.

Example:

```
10    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 9

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

Option 9 is used to turn ON or OFF the specified OCK FLAG.

The MOVE ZERO clause will turn OFF the specified OCK FLAG(s).

The MOVE ONE clause will turn ON the specified OCK FLAG(s).

The ACCUMULATOR is undisturbed.

Example:

```
12    MOVE 1 TO OCK4.
```

OPTION 10

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

Option 10 will insert the value of the first operand into the selected FORMS LIMIT REGISTER. The value cannot exceed 255.

If the INDEXED BY option is used, the value inserted will be the sum of the first operand and the selected INDEXED BY option.

There is no effect upon the ACCUMULATOR unless the INDEXED BY option is utilized, in which case the ACCUMULATOR will contain the value of the INDEXED BY operand.

MOVE

OPTION 11

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

Option 11 is used to load a value in the respective FORMS COUNT-REG. The value cannot exceed 255.

If RIGHT or LEFT is not specified, LEFT is assumed.

When the INDEXED BY option is used, the value loaded will be equal to operand 1 plus the value of the INDEXED BY operand.

There is no effect upon the ACCUMULATOR unless the INDEXED BY option is used, in which case the ACCUMULATOR will contain the value of the INDEXED BY operand.

Example:

```
08
09      MOVE 10 TO RIGHT COUNT-REG.
10
```

OPTION 12

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

Option 12 is used to clear all the 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 memory this construct should not be used unless the data items within the group occupy more than 6 words.

The ACCUMULATOR is undisturbed.

Example:

```
11
12      MOVE ZEROES TO ALL-TOTALS.
13
```

Multiply

The MULTIPLY verb functions to MULTIPLY two numeric-data-items.

The construct is as follows:

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

Operand 1 is MULTIPLIED BY operand 2. The product is stored in operand 2 unless the GIVING option is specified; in which case the product is stored in numeric-data-name-3.

Automatic decimal alignment is provided if both operand 1 and operand 2 are numeric-data-names. MULTIPLY is the only arithmetic verb which allows ROUNDING without the GIVING option.

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

Example:

```
01
02      MULTIPLY 3 BY TOTAL GIVING FINAL-RESULT ROUNDED.
```

No-Op

The use of this verb is to allow insertion of a NO OPERATION machine code.

The construct is:

```
NO-OP
```

Program execution continues sequentially uninterrupted. 10 milliseconds are expended.

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
OPEN

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:

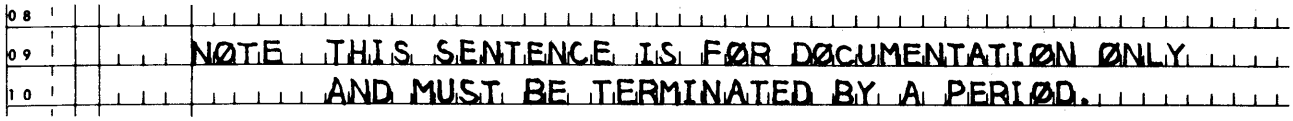
OPTION 1

```
NOTE sentence.
```

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:

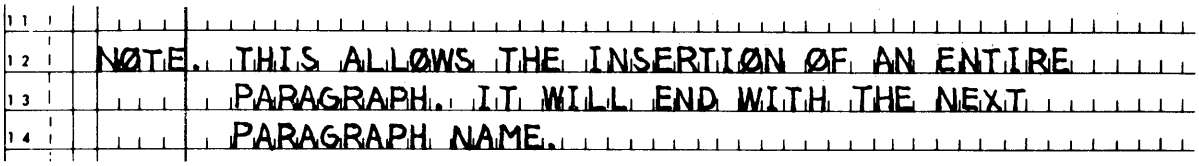


OPTION 2

```
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:



Open

This construct provides for the opening of the forms handler.

The construct is as follows:

```
OPEN TRANSPORT  TO [{integer       }]   [INDEXED BY {ACCUMULATOR      }]
                   {special-name}]      {numeric-data-name}]
```

The forms handler will be opened.

If the integer/special-name option is used, the forms handler will stay open until a print or programed close is made, at which time the left 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.

There is no effect upon the ACCUMULATOR unless the INDEXED BY option is used. With that option the ACCUMULATOR is unknown.

Example:

```
           OPEN FORMS HANDLER 19.
```

Perform

The use of this verb is to depart from the normal sequence of program execution in order to execute a procedure which was declared as a subroutine in the DECLARATIVES. When the subroutine execution is complete, program control is automatically returned to the statement after the PERFORM statement.

See EXIT page 6-17.

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. Nested subroutines are EXITed in reverse sequence to that in which they were PERFORMed. It is the programmers responsibility not to nest more than four subroutines.

The ACCUMULATOR is undisturbed.

Example:

```
17
18         PERFORM FINAL-TOTALS-ROUTINE.
19
```

Position

The POSITION verb allows the programmer to position the print ball.

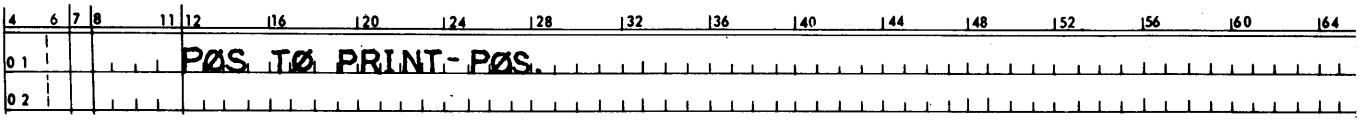
```
{POSITION} TO {integer      } [ INDEXED BY {ACCUMULATOR      } ]
{POS     }    {special-name}                {numeric-data-name}
```

The print positions are numbered 1 through 150 left to right (some machine styles allow 1-255).

RED-RIBBON
STOP RUN
SUBTRACT

This construct has no effect on the ACCUMULATOR unless the INDEXED BY option is used, in which case the ACCUMULATOR will contain the value of the INDEXED BY operand.

Example:



Red-Ribbon

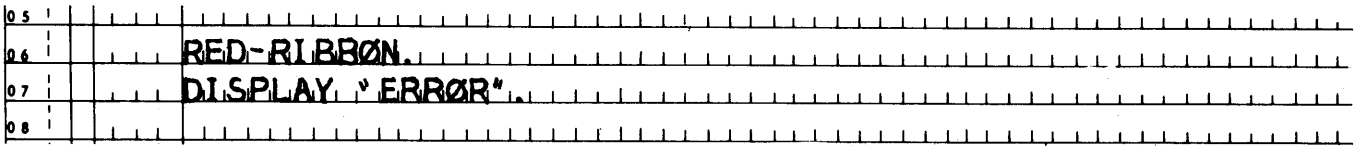
The use of this construct is to provide for activating the red-ribbon feature of the machine for the next DISPLAY or ACCEPT statement.

The construct is as follows:

```
RED-RIBBON
```

The construct is in effect only for the next DISPLAY or ACCEPT statement.

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, otherwise warning message will appear.

Subtract

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

OPTION 1

```
SUBTRACT {numeric-data-name-1} FROM {numeric-data-name-2}
         {integer              }      {ACCUMULATOR     }

    [GIVING numeric-data-name-3 [ROUNDED]] [ON SIZE ERROR statements]
```

Option 1 will SUBTRACT operand 1 from operand 2 storing the difference in operand 2.

If the GIVING option is used, the difference is stored in numeric-data-name-3.

Automatic decimal alignment is provided if both operands are numeric-data-names. ROUNDED is only valid with the GIVING option and will produce correct results only when the third operand has fewer decimal places than other two operands.

OPTION 2

SUBTRACT ACCUMULATOR FROM numeric-data-name-1 [GIVING numeric-data-name-2 [ROUNDED]] [ON SIZE ERROR statements]

Option 2 will SUBTRACT the ACCUMULATOR from the specified numeric-data-name. Without the GIVING option the remainder is stored in the second operand. With the GIVING option the remainder is stored in the third operand.

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

Example:

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

Use

The use of the verb is to specify procedures or to define the program key table (PK-table). The USE verb can only be contained in the DECLARATIVES.

OPTION 1

USE FOR PK-TABLE table-name

It is important to note that if they are used, all PK tables must be declared before subroutines are declared.

The following constructs may be coded as PK table entries:

1. ADVANCE — without INDEXED BY clause.
2. DISPLAY — options 2 and 3.
3. GO TO
4. MOVE — options 7 through 9
5. NO-OP
6. OPEN HANDLER — without INDEXED BY clause.
7. PERFORM
8. POSITION TO — without INDEXED BY clause.
9. RED-RIBBON

USE

PK table entries must appear in the following order:

Entry for PK 1
Entry for PK 2
Entry for PK 3
Entry for PK 4
Entry for PK 5
Entry for PK 6
Entry for PK 7
Entry for PK 8
Entry for PK 9
Entry for PK 10
Entry for PK 11
Entry for PK 12
Entry for PK 13
Entry for PK 14
Entry for PK 15
Entry for PK 16
Entry for PK 17
Entry for PK 18
Entry for PK 19
Entry for PK 20
Entry for PK 21
Entry for PK 22
Entry for PK 23
Entry for PK 24

If an entry is made for a PK, all PK's with numbers less than the chosen PK must be represented in the table.

Example:

Provide table entries for PK 1, PK 3, PK 5.

```
10
11      USE FØR PK-TABLE PRØGRAM-CHØICE.
12      GØ TØ CLEAR-MEMØRY.
13      NØ-ØP.
14      GØ TØ ERRØR-RØUTINE.
15      NØ-ØP.
16      GØ TØ PRINT-FINAL-TØTALS.
```

OPTION 2

USE FOR SUBROUTINE procedure-name

The option allows any combination of sentences and paragraphs to use as a subroutine. Program control is transferred to the procedure name by the PERFORM verb. Control is transferred from the subroutine to the mainline program by one of the two following methods:

The program uses the EXIT verb.

The automatic EXIT is utilized.

PART B: PAPER TAPE I/O

This part of the PROCEDURAL CONSTRUCTS presents and explains in detail the constructs which can only be utilized with firmware sets providing paper tape input and output capabilities.

References to paper tape and edge-punched cards are equivalent.

All data-names associated with paper tape input and output are defined in the WORKING-STORAGE SECTION of the DATA DIVISION.

SPECIAL HARDWARE NAMES FOR PAPER TAPE

Some paper tape constructs contain special hardware names. These names and these meanings are:

KEYBOARD-PCH	Refers to entering of data through the keyboard, punching same, but not printing
PCH	Refers to punching of data.
PRNTR-PCH	Refers to printing and punching of data.
RDR	Refers to reading of paper tape.
RDR-PCH	Refers to reading and punching of paper tape.
RDR-PRNTR	Refers to reading of paper tape and printing.
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 the same.

READING OF PAPER TAPE

When a statement calls for reading of paper tape, reading occurs by character until the read instruction is terminated. Termination of reading takes place once one of the two conditions below is satisfied:

1) the number of characters associated with the PICTURE or FORMAT of the data name have been read;
2) a field identifier code.

Certain field identifier codes will turn certain flags on and off (see Appendix A).

PUNCHING OF PAPER TAPE

When punching numeric data, punching occurs for the number of digits associated with the PICTURE of the data name.

When punching alpha data, punching occurs for the number of characters contained in the field at the time of the punch statement.

Field identifier codes are not punched automatically.

APPLICABILITY OF PART A

Any construct explained in PART A can be used with PART B.

Accept

This verb provides the ability to read paper tape and as by-products type and/or punch into paper tape.

This construct has four options:

OPTION 1

ACCEPT alpha-data-name FROM { KEYBOARD-PCH | RDR | RDR-PRNTR | RDR-PCH | RDR-PRNTR-PCH | KYBRD-PRNTR-PCH }

Option 1 provides the functions as described under SPECIAL HARDWARE NAMES FOR PAPER TAPE.

Data will be stored in memory.

There is no effect upon the ACCUMULATOR.

Example:

```
NO.
4  6 7 8   11 12   16   20   24   28   32   36   40   44   48   52   56   60   64
01 |   |   |  ACCEPT DESCRIPTION FROM RDR.
02 |   |   |
```

OPTION 2

ACCEPT integer CHARACTERS FROM { RDR-PRNTR | RDR-PRNTR-PCH | KYBRD-PRNTR-PCH }

Option two will provide the specified function for an integer number of alpha characters. Data is not stored in memory.

Example:

```
03 |   |   |
04 |   |   |  ACCEPT 25 CHARACTERS FROM RDR-PRNTR.
```

DISPLAY

(PT)

OPTION 3

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

Option 3 will provide the specified function.

The ACCEPT numeric-data-name option will store the entered numeric-data in memory.

The ACCEPT ACCUMULATOR numeric-data-name option will not store the entered data in memory.

In either case the ACCUMULATOR contains the entered data after Option 3 is executed.

Example:

```
07
08     ACCEPT QUANTITY FROM KEYBOARD-PCH.
```

OPTION 4

```
ACCEPT INTO ACCUMULATOR FROM RDR
```

Option 4 will read a maximum of 15 numeric characters into the ACCUMULATOR.

The entered data is not stored in memory.

Example:

```
05
06     ACCEPT ACCUMULATOR FROM RDR.
```

Display

The use of this verb is to provide for printing and punching of data from memory.

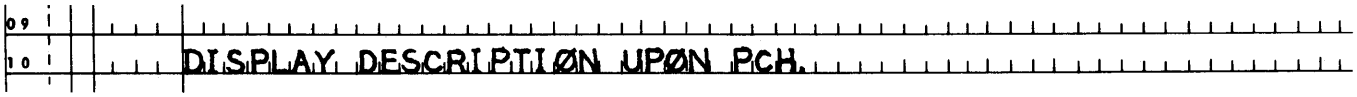
There are five options:

OPTION 1

```
DISPLAY { alpha-data-name      }  UPON { PRNTR-PCH }
        { "non-numeric literal" }      { PCH       }
```

Option 1 will punch from memory the contents of the specified alpha-data-name (or alpha-literal). There is not a field identifier code automatically punched. The ACCUMULATOR is not disturbed.

Example:



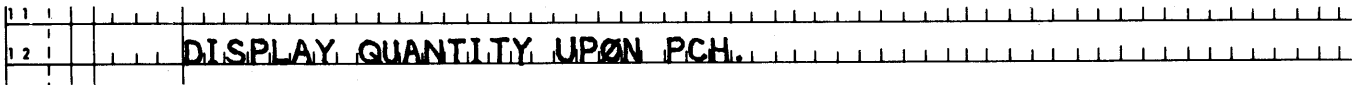
OPTION 2

DISPLAY numeric-data-name UPON { PRNTR-PCH / PCH }

Option 2 provides for punching of numeric-data. A field identifier is not punched.

The ACCUMULATOR will contain the contents of the stated numeric-data-name.

Example:



OPTION 3

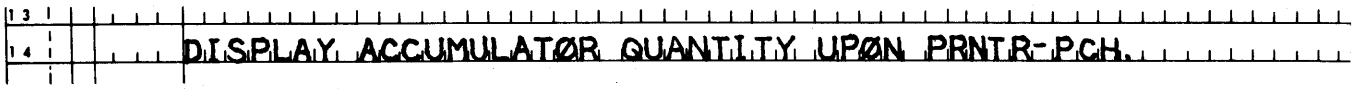
DISPLAY ACCUMULATOR [(integer)] numeric-data-name UPON { PRNTR-PCH / PCH }

Option 3 will provide the specified function for the integer rightmost positions of the ACCUMULATOR. The nature of the output is determined by the PICTURE assigned the numeric-data-name.

If the integer option is omitted, the entire ACCUMULATOR will be used.

A field identifier is not automatically punched.

Example:

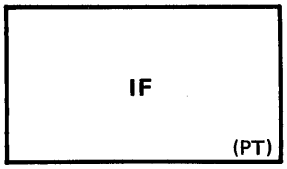


OPTION 4

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

Option 4 is most commonly used to control the edge-punched cards in the paper tape punch.

Without the INDEXED BY clause, integer number of 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.

OPTION 5

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

Option 5 provides for punching into paper tape the bit pattern specified by a and b. Any one of 128 possible characters may be punched.

Example:

Appendix A contains the tables which reference the bit patterns for the characters.

If

These constructs provide the ability to check the condition of the peripheral devices.

OPTION 1

```
IF { RDR-ERR    } THEN { statements      } [ELSE statements]
   { RDR-COND   }      { NEXT SENTENCE  }
```

OPTION 2

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

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.

Move

This verb provides the ability to reset the paper tape reader error and paper tape punch error flags, and to load values into the PCH-REG.

There are two options:

OPTION 1

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

Option 1 resets the selected flag.

OPTION 2

```
MOVE integer TO PCH-REG
```

Option 2 is used to load the punch count register. The integer may be any value less than 256.

Example:

```
17
18     MOVE 87 TO PCH-REG.
```

Open

This construct provides the ability to open the paper tape media clamp.

```
OPEN MEDIA-CLAMP
```

Example:

```
19
20     OPEN MEDIA-CLAMP.
```

PART C: 80-COLUMN CARD I/O

Part C presents and explains in detail the constructs which can only be utilized with firmware sets providing 80-column card input and output capabilities.

Card input data must be ASSIGNed to the CARD-READER in the FILE-CONTROL paragraph (see page 4-3). The fields of the card input file are defined in the FILE SECTION of the DATA DIVISION. Because of the nature of the output, the CARD PUNCH is excluded from FILE CONTROL.

SPECIAL HARDWARE NAMES FOR 80-COLUMN CARD

Some 80-column card constructs contain special hardware names. These names and their meanings are:

KEYBOARD-PCH	Refers to entering data through the keyboard, punching same, but not printing.
PCH	Refers to punching of data.
PRNTR	Refers to printing of data.
PRNTR-PCH	Refers to printing and punching of data.
KYBRD-PRNTR-PCH	Refers to entering of data through the keyboard, printing, and punching same.

80-COLUMN CARD I/O CONSTRUCTS

Since the allowable constructs are dependent upon whether USE WORK-AREA or NO WORK-AREA was declared in the FILE CONTROL paragraph, the constructs are explained in three sections:

Constructs which apply to both USE WORK-AREA and NO WORK-AREA.

Constructs which apply to NO WORK-AREA.

Constructs which apply to USE WORK-AREA.

CONSTRUCTS APPLICABLE TO BOTH USE WORK-AREA AND NO WORK-AREA

In addition to the constructs described below, any construct presented in PART A can be used.

Accept

This verb provides for entering of data through the KEYBOARD and punching the same. Printing occurs only as specified.

There are four options:

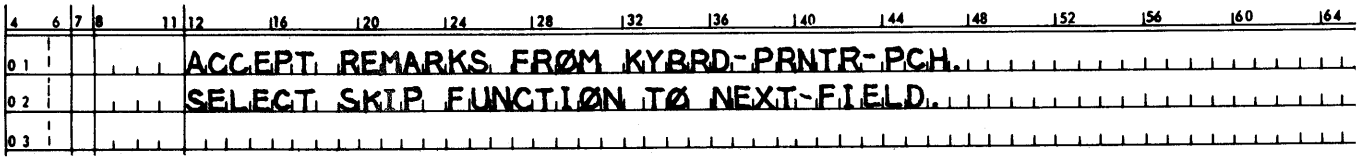
OPTION 1

```
ACCEPT alpha-data-name FROM { KEYBOARD-PCH    }
                            { KYBRD-PRNTR-PCH }
```

Option 1 will allow typing and punching of data as defined by the PICTURE of the alpha-data-name. Printing occurs only when KYBRD-PRNTR-PCH is used. The entered data will be stored in memory.

If the punch is off-line, the option is executed without punching.

Example:



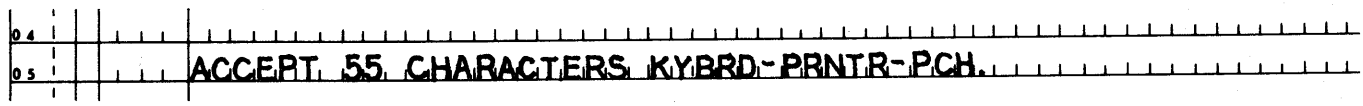
OPTION 2

ACCEPT integer CHARACTERS FROM KYBRD-PRNTR-PCH

Option 2 provides for typing, punching and printing an integer number of characters from the alpha keyboard. The integer may range from 1-80. Any data entered is not stored in memory.

If the punch is off-line, the option will be executed without punching.

Example:



OPTION 3

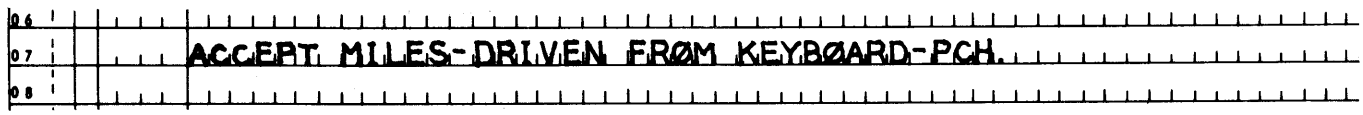
ACCEPT numeric-data-name FROM { KEYBOARD-PCH / KYBRD-PRNTR-PCH }

Option 3 provides for entering of numeric-data through the numeric keyboard and punching the entered data. Printing occurs only when KYBRD-PRNTR-PCH is specified.

The entered data is stored in memory and the ACCUMULATOR.

If the punch is off-line, option 3 will be executed without punching.

Example:



DISPLAY

(CRD)

OPTION 4

ACCEPT INTO ACCUMULATOR numeric-data-name { KEYBOARD-PCH / KYBRD-PRNTR-PCH }

Option 4 functions as Option 3 except the entered data remains in the ACCUMULATOR and is not stored in memory.

Display

This construct provides for punching of data from memory.

There are 4 options:

OPTION 1

DISPLAY { alpha-data-name / "literal" } UPON { PRNTR-PCH / PCH }

Option 1 provides for punching of alpha-data from memory. Printing occurs only when PRNTR-PCH is specified.

If the punch is off-line, PRNTR-PCH will function only as PRNTR. PCH will function as a NO-OP under the same circumstances.

Example:

```
09
10     DISPLAY "A" UPON PCH.
```

OPTION 2

DISPLAY numeric-data-name UPON { PRNTR-PCH / PCH }

Option 2 provides for punching of numeric-data. Printing occurs only when PRNTR-PCH is specified. Punching occurs as determined by the PICTURE associated with the numeric-data-name. The ACCUMULATOR will contain the numeric-data-name after execution of option 2.

If the punch is off-line, PRNTR-PCH will function only as PRNTR, PCH will function as a NO-OP under the same circumstances.

Example:

```
11
12     DISPLAY MILES-PER-HOUR UPON PRNTR-PCH.
```

OPTION 3

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

Option 3 provides for punching all or part of the ACCUMULATOR. Printing occurs only when PRNTR-PCH is specified. Punching occurs as determined by the PICTURE associated with the numeric-data-name.

If the integer option is used, punching will begin at the integer digit position of the ACCUMULATOR. The punching format is determined by the PICTURE associated with the numeric-data-name beginning with the integer digit position.

If the integer is omitted, punching will be determined by the PICTURE of the associated numeric-data-name. If the punch is off-line, PRNTR-PCH will function only as PRNTR. PCH will function as a NO-OP under the same circumstances.

The ACCUMULATOR remains unchanged.

Example:

```
13
14    DISPLAY ACCUMULATOR(10) QUANTITY-ON-ORDER UPON PCH.
```

OPTION 4

DISPLAY @ab@ UPON PCH

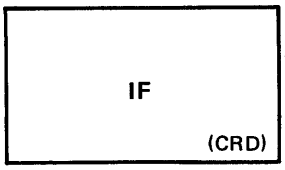
This option permits the outputting of any desired single card code (without it being resident in memory) or any special punch pattern in a card column (except only one punch can be created in rows 1 to 7 in a card column although any punch combination in the other rows can be obtained). The a controls punching in card rows 12, 11, 0 and 9; the b controls punching in card rows 1 through 8.

Printing does not occur. If the punch is off option 4 is executed as NO-OP.

Example:

	To punch Rows					
	12	11	0	9	8	1-7
use a value of	8	4	2	1		
use b value of					8	1-7

To punch an A (a=12, b=1) the statement should read:



DISPLAY 8, 1 UPON PCH

Which means punch row 12 and row 1.

If

This construct provides the ability to check the condition of the peripheral devices.

There are three options:

OPTION 1

IF RDR-ERR THEN { statements / NEXT SENTENCE } [ELSE statements]

Option 1 will be TRUE if an invalid code is transferred to the ACCUMULATOR by the MOVE numeric-file-data-name TO ACCUMULATOR construct. The statement will become FALSE after the next statement.

OPTION 2

IF PCH-ERR THEN { statements / NEXT SENTENCE } [ELSE statements]

Option 2 will be TRUE if a card punch malfunction or misoperation occurs. The statement becomes FALSE only when the card punch RESET key is depressed.

Example:

```
15
16    IF RDR-ERR THEN GØ TØ CØRRECTIØN-RØUTINE.
```

OPTION 3

IF PCH-OFF THEN { statements / NEXT SENTENCE } [ELSE statements]

Option 4 will be TRUE if the card punch is off-line, or if a card punching instruction is attempted and the ON/OFF switch is not on. Program execution does not halt. Because execution does not halt, it is recommended that during program initialization, the SELECT SKIP FUNCTION TO 1 be used followed by option 3. It is then possible to alert the operator that the punch is off-line.

Example:

```
17
18      SELECT SKIP FUNCTION TO 1.
19      IF PCH-OFF THEN ALARM, ACCEPT FROM KEYS.
```

Read

This construct provides the ability to read 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-READER in the SELECT clause of the ENVIRONMENT DIVISION.

The data in the BUFFER is in a character mode, and therefore cannot be accessed directly.

The ACCUMULATOR is destroyed.

All the 80 columns of the card are read and placed into memory including blank columns.

Example:

```
20      READ CARD-DATA.
```

Select

The SELECT verb provides control of the card punch.

There are three options:

OPTION 1

SELECT ALTERNATE STACKER

Option 1 indicates that the card in the punch station is to be placed in the ALTERNATE STACKER.

OPTION 2

SELECT SKIP FUNCTION TO { special-name / integer }

Option 2 will cause a SKIP to the indicated card column. A SKIP to 1 causes a new card to be registered at column 1 (releases a card).

This option is used to ensure program and card punch synchronization.

Example:

```
01   SELECT SKIP FUNCTION TO NEXT-FIELD.
```

OPTION 3

SELECT REPEAT FUNCTION { THROUGH / THRU } { special-name / integer }

Option 3 will cause duplication through the specified card column. A REPEAT THROUGH 80 will cause the card to be duplicated through column 80 and a new card to be registered in column 1.

Attempting to DUPLICATE THROUGH a card column already passed will result in an error condition. That is, program and card synchronization will be upset.

Example:

```
03   SELECT REPEAT FUNCTION THRU 80.
```

NO WORK-AREA DECLARED IN FILE CONTROL

The following constructs apply when the programmer has declared NO WORK-AREA.

Display

This construct provides the ability to print and/or punch alpha-file-data-names from the BUFFER.

DISPLAY alpha-file-data-name [FROM BUFFER] UPON { PRNTR / PRNTR-PCH / PCH }

Notice that numeric-file-data-names cannot be DISPLAYed directly from the BUFFER. See programing considerations page 6-50.

Example:

```
05   DISPLAY ACCOUNT-STATUS FROM BUFFER UPON PRNTR.
```

Move

This verb provides the ability to MOVE alpha-file-data-names to alpha-data-names; numeric-file-data-names to numeric-data-names or the ACCUMULATOR.

There are two options:

OPTION 1

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

This construct will move operand 1 from the BUFFER to operand 2. Operand 1 must be less than or equal to operand 2 in regard to size.

OPTION 2

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

This construct will MOVE operand 1 to operand 2. When operand 2 is a numeric-data-name, operand 1 is decimally aligned to operand 2.

Example:

```
06
07     MOVE GROSS-WEIGHT TO ACCUMULATOR.
```

Fill

The format of this construct is as follows:

```
FILL record-name
```

This construct can only appear after the READ statement has been used. FILL will cause the data in the 80-column card input buffer to be moved from the buffer and reformatted into the fields of the designated record. The data will be in word mode. The alpha-file-data-names and numeric-file-data-names can then be used as alpha data-names and numeric-data-names of PART A.

Example:

```
08
09      MØVE CARD-CØDE FRØM BUFFER TØ ACCUMULATØR.
10      IF CARD-CØDE IS EQUAL TØ 4
11          FILL PARTS-ØN-ØRDER-CARD
12          GØ TØ PRØCESS-1.
13      IF CARD-CØDE IS EQUAL TØ 6
14          FILL PARTS-IN-STØCK-CARD
15          GØ TØ PRØCESS-2
16      ELSE GØ TØ ERRØR-RØUTINE.
```

This example illustrates a technique used to FILL the desired record depending upon a card code.

If the CARD-CODE equals 4, the PARTS-ON-ORDER-CARD will be FILLED. If the CARD-CODE equals 6, the PARTS-IN-STOCK-CARD will be FILLED otherwise an error routine will be entered.

The programmer then provides the necessary constructs for the desired data manipulation.

PROGRAMING CONSIDERATIONS

1) Since numeric-file-data-names cannot be DISPLAYED directly from the BUFFER, a technique similar to the one below should be employed.

   In the WORKING-STORAGE SECTION

```
4 6 7 8  11 12  16  20  24  28  32  36  40  44  48  52  56  60  64
01      01  NUMERIC-FILE-DATA-PRINTING.
02          02 WØRK-1 PC 9(15).
03          02 MASK-1 REDEFINES WØRK-1 PC 999,999.99.
04          02 MASK-2 REDEFINES WØRK-1 PC ZZZ,ZZZ,ZZZ.99.
05          02 MASK-3 REDEFINES WØRK-1 PC $ZZZ,ZZ9.99-.
```

   In the PROCEDURE DIVISION

```
08
09      MØVE GRØSS-TØNNAGE TØ ACCUM.
10      DISPLAY ACCUMULATØR MASK-2.
11      MØVE DØLLAR-VALUE TØ ACCUM.
12      PØS TØ 50.
13      DISPLAY ACCUMULATØR MASK-1.
```

   WORK-1 can be used as a working-area. The above technique accomplishes the necessary task in a minimum of memory.

2) Negative data is indicated in the following manner: If an 11 overpunch is present in any column of a numeric field, the field will be negative (ACCUMULATOR NFLAG ON). The hyphen character (-) will have the same effect but will transfer the respective card column as zero.

3) A 12 or 0 overpunch will give the IF RDR-ERR statement a TRUE value.

4) Punching as well as printing is a function of the PICTURE. The characters and punching results are described on page 5-5 under the PICTURE clause discussion.

PART D: DATA COMMUNICATIONS CAPABILITY

Part D presents and explains in detail the constructs which can only be utilized with firmware sets providing data communications capabilities.

The data received by the TC must be ASSIGNED to the DATA-COMM INput buffer. The fields of the received message are defined in the FILE SECTION of the DATA DIVISION. The data to be sent by the TC must be ASSIGNED to the DATA-COMM OUTput buffer. The message fields are defined in the FILE SECTION of the DATA DIVISION. (See page 4-3)

MODES OF OPERATION

The diagrams below summarize the permissible modes of operation and the basic procedures associated with each mode.

DATA-COMM OUT

	NO WORK-AREA	USE WORK-AREA
NO ALTERNATE AREA		
	LOCATE file-name pack BUFFER with only these data comm constructs: ACCEPT Move option 1, option 2. Must pack BUFFER sequentially (i.e., pack fields into BUFFER in the same order as defined in record) WRITE record-name	Use any DATA-COMM OUT file-data-name as a working storage data-name when packing BUFFER (See part A Pack WORK-AREA in any order. WRITE record-name
RESERVE ALTERNATE AREA	Same as above	Same as above

ACCESS MODE IS SEQUENTIAL SPECIFIED

	NO WORK-AREA	USE WORK-AREA
NO ALTERNATE AREA		
	READ file-name Unpack BUFFER with only these data comm constructs: DISPLAY MOVE option 3, option 4. Must MOVE and DISPLAY all characters in sequence only once LOCATE file-name	READ file-name FILL record-name can now use file-data-names as working storage data-names. LOCATE file-name
RESERVE ALTERNATE AREA	Same as above	Same as above

ACCESS MODE NOT SPECIFIED

	NO WORK-AREA	USE WORK-AREA
NO ALTERNATE AREA		
	READ file-name Unpack BUFFER with only these data comm constructs: DISPLAY MOVE option 3, option 4. LOCATE file-name	READ file-name Can use "FROM BUFFER" constructs until FILL record-name can now use file-data-names as working-storage data-names
RESERVE ALTERNATE AREA	Same as above	Same as above

RESERVE ALTERNATE AREA

The RESERVE ALTERNATE AREA clause will set aside an area in user memory equal in word size (32) to the DATA-COMM SEND or RECEIVE BUFFER. The compiler will then automatically transfer the contents of the DATA-COMM RECEIVE BUFFER to the ALTERNATE AREA and the contents of the ALTERNATE AREA to the DATA-COMM SEND BUFFER at the appropriate time. The use of an ALTERNATE AREA increases throughput of the system as once data is transferred to the DATA-COMM SEND BUFFER the program can continue execution while the data is being transmitted. The DATA-COMM RECEIVE BUFFER will be free to receive another message as soon as the data is transferred to the ALTERNATE AREA, so a message received can be processed while another message is being transmitted to the terminal computer.

The Assembler memory size option must be utilized to ensure proper placement of the Reserved Alternate area.

FIXED AND VARIABLE SIZE FIELDS

Fixed fields must always precede variable fields in the records contained within the file ASSIGNED to DATA-COMM IN. Variable length fields are identified by the USE FOR DELIMITER clause.

APPLICABILITY OF PART A

Any construct found in PART A applies.

Accept

The Accept construct will allow entry of data directly into the DATA-COMM SEND BUFFER.

The construct is:

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

This construct has no effect on the ACCUMULATOR.

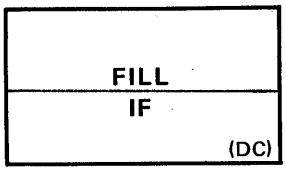
Example:

```
03     ACCEPT MESSAGE-CODE FROM KEYBOARD.
04
```

Display

The function of this construct is to allow printing of alpha-file-data-names directly from the DATA-COMM RECEIVE BUFFER.

DISPLAY alpha-file-data-name [FROM BUFFER] UPON PRNTR



Numeric-file-data-names cannot be DISPLAYED directly from the BUFFER. They first must be MOVED to the ACCUMULATOR and the DISPLAY ACCUMULATOR working-storage-data-name construct must be used. See page 6-15.

FROM BUFFER is optional unless USE WORK-AREA is declared and the program requires printing alpha prior to the use of FILL.

Fill

The FILL construct will cause the data in the DATA-COMM RECEIVE BUFFER to be transferred from the buffer and reformatted into main memory fields.

The construct is:

FILL record-name.

FILL should only be used after the READ construct.

See page 6-52.

The file-data can then be utilized as working-storage-data. The buffer can be interrogated in the same manner the card buffer is interrogated.

See pages 6-49, and 6-50.

If

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

There are nine options:

OPTION 1

IF XMT-RDY THEN statements [ELSE statements]

Option 1 will be TRUE whenever the DATA COMM processor has set the remote terminal in a transmit ready state. When using NO ALTERNATE AREA, the above option must be FALSE before accessing of the DATA-COMM SEND BUFFER should begin.

See LOCATE page 6-56.

Example:

```
05
06     IF XMT-RDY MOVE 0 TO XMT-RDY.
```

OPTION 2

IF RCV-RDY THEN statements [ELSE statements]

Option 2 will be TRUE whenever the remote terminal is in a receive ready state.

Example:

```
07
08     IF RCV-RDY THEN MØVE O TØ RCV-RDY.
```

OPTION 3

IF { XMT-RDY / RCV-RDY } [{ AND / OR } { XMT-RDY / RCV-RDY } ...] THEN statements [ELSE statements]

Option 3 provides the ability to test the XMT-RDY and/OR RCU-RDY flag in the same statement.

OPTION 4

IF DC-ERROR THEN statements [ELSE statements]

Option 4 will be TRUE whenever the Header Transmission Number is not equal to the Expected Transmission Number. The system will respond with an ACK.

Example:

```
09
10     IF DC-ERRØR THEN
11         PØS TØ ERRØR-PØSITIØN
12         DISPLAY 'ERRØR IN TRANSMISSIØN".
```

LOCATE

(DC)

OPTION 5

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

This option allows the program to interrogate the condition of the keyboard buffer and to determine whether or not data has been indexed upon the keyboard (alpha or numeric).

Locate

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

The construct is:

LOCATE file-name

When working directly with the DATA-COMM RECEIVE BUFFER with NO WORK-AREA, 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-COMM SEND BUFFER with NO-WORK-AREA, this construct must be used prior to accessing the send buffer to determine if the send buffer is available and to set the buffer pointers.

Example:

```
13
14     LØCATE ØN-LINE-IN.
```

Move

The following MOVE constructs are used to transfer the data to and from the data communications buffers and adjunct exchange memory.

There are 11 options:

OPTION 1

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

Option 1 is used to transfer alpha data from working-storage memory to the alpha-file-data-name. The ACCUMULATOR is undisturbed.

OPTION 2

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

Option 2 will transfer the data in operand 1 to the specified numeric-file-data-name.

OPTION 3

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

Option 3 will be used to transfer alpha data from the receive area to the alpha-data-name. The ACCUMULATOR is undisturbed.

OPTION 4

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

Option 4 will be used to transfer numeric data from the receive area to the ACCUMULATOR or a numeric-data-name.

MOVE

(DC)

OPTION 5

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

Option 5 is used to load the various addresses and memory. See below for an interpretation of the abbreviations.

OPTION 6

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

Option 6 is used to load the various addresses and transmission numbers into 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 — Broadcast Transmission Number

OPTION 7

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

Option 7 is used to turn off or on the DC-ERROR flag.

OPTION 8

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

Option 8 is used to turn on or off the RCV-RDY and XMT-RDY flags.

OPTION 9

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

Option 9 is used to load two wire or four wire control. It is essential to load the proper mode of operation depending upon the communications link between the computers. Two wire is specified when switch line or two wire lease line is used. Four wire is specified when four wire lease line or any type of direct connect is used.

OPTION 10

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

Option 10 is 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).

Example:

```
15
16     MØVE @14@ TØ DATA-CØMM.
```

READ
STOP
WRITE (DC)

Read

The purpose of this construct is to READ a file which has been assigned to DATA-COMM IN.

The construct is:

READ file-name.

This construct will determine if a record has been received into the DATA-COMM RECEIVE BUFFER. If a record has not been received, the computer will hang in a wait loop, otherwise execution occurs thusly:

1. If a record has been received and no ALTERNATE AREA was declared, control will be transferred to the next statement.

2. If a record has been received and an ALTERNATE AREA was declared, the data will be transferred from the RECEIVE BUFFER to the ALTERNATE AREA, the RCU-RDY flag will be turned off and control transferred to the next sentence.

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 will turn power off to the system.

STOP MACHINE

Write

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

The construct is:

WRITE record-name

When NO WORK-AREA and NO ALTERNATE AREA has been declared, the construct will turn the XMT RDY flag ON.

When RESERVE ALTERNATE AREA or USE WORK-AREA has been declared, this construct will determine if the DATA-COMM SEND BUFFER is clear and then transfer the data from the declared area to the DATA-COMM SEND BUFFER and turn ON the XMT-RDY 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 GØ TØ BUFFER-NØT-CLEAR
19          ELSE WRITE DATA-CØMM-RECØRD.
```

CONVERT
IF
(CD)

PART E: CHECK DIGIT CAPABILITIES

Part E presents and explains in detail the constructs which only can be utilized with firmware sets providing check digit capabilities. These constructs require a check digit table (CD-TABLE) to be resident in memory. The CD-TABLE is declared in the WORKING-STORAGE SECTION of the DATA DIVISION (see page 5-11).

Convert

The purpose of this construct is to provide for check digit computation.

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

The CHECK-DIGIT will be calculated using the number of characters in the PICTURE clause of the data-name in the first operand. The second data-name indicates the table to be utilized.

The integer option will state the integer to be used as a remainder factor. No entry will result with an assumed zero remainder.

Example:

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

If

The use of this verb is to provide for check digit verification.

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

   [TRUNCATED] THEN statements [ELSE statements]
```

This construct will verify the check digit using the table located at data-name-2 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.

PART F: STERLING CAPABILITIES

Part F presents and explains the constructs which can only be used with firmware sets providing STERLING capabilities.

Convert

The use of this construct is to provide for the conversion of data to the Sterling monetary system.

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

Display

The use of this construct is to provide for the printing of data in the specified unit of the Sterling monetary system.

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

Example:

```
DISPLAY ACCUM HALF-PENNY.
```

If

The use of this construct is to provide for the testing of keyboard entered data to determine if the Sterling keys were used incorrectly.

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

Round

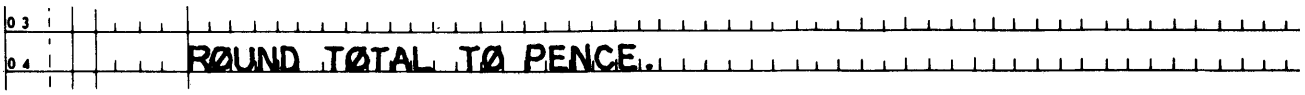
This construct is used to round to the Sterling unit PENCE data stored in memory or the ACCUMULATOR.

ROUND

(ST)

ROUND { ACCUM / data-name } TO PENCE

Example:



PART G: TC 700

Part G presents the constructs which are particular to the TC 700 Series equipment.

If

TC 700 Flag Tests

These tests are used to check the status of certain TC 700 flags.

IF { TELLER-1 / TELLER-2 / SUPERVISOR / PB-FIRST-LINE / PB-LAST-LINE / PB-FOLD / PB-NOT-PRESENT } THEN statements [ELSE statements]

If the flag associated with operand 1 is ON (set), the statement is TRUE.

Teller-1, Teller-2 and SUPERVISOR refer to the Teller locks. When a Teller key is inserted in its lock and turned, the Teller key flag for that lock will be turned ON (set).

When the Passbook is inserted to the fixed rear limit, the PB-FIRST-LINE flag will be ON (set). It will be OFF (reset) at all other times. When the Passbook is so situated in the alignment area that the current print line will fall within the Passbook fold area, the PB-FOLD flag will be ON (set). It will be OFF when the above condition does not exist.

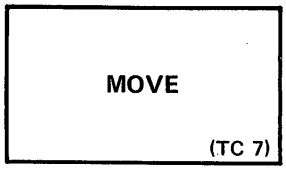
If the Passbook is so aligned that the current print line is below the last printing line of the passbook, the PB-LAST-LINE flag will be ON (set). It will be OFF whenever the passbook is aligned to any of the actual printing lines of the passbook.

If a passbook is not present, the PB-NOT-PRESENT statement will be TRUE.

Example:

```
01  IF TELLER-1 THEN GØ TØ INDEX ELSE GØ TØ ERRØR-RØUTINE
02
```

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



Move

The construct is used to turn on or off programmatically the Passbook Required indicator. The PB-LAST-LINE will be interrogated to determine whether the PB-REQUIRED indicator should be turned on or off.

MOVE ONE will turn the indicator on.

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.
17. SAVE XXXXXX – This will cause the Assembler to retain the object program upon the disk. Punching of paper tape or 80-column card does not occur. XXXXXX represents a 6-alpha character disk file-name.
18. EXTMEM – This will indicate that this program will utilize a 40-track Series L style.

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

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	SWITCH PATTERN SET BY CODE* SWITCH NUMBER				
CODE		7	6	5	8	
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

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 | POS} integer.]
        [data-name IS LINE integer.]
        [data-name IS {COLUMN | COL} integer.]]

[INPUT-OUTPUT SECTION.]

[FILE - CONTROL]

[SELECT file-name ASSIGN TO CARD-READER {USE | NO} WORK - AREA]

SELECT file-name ASSIGN TO DATA-COMM IN [RESERVE ALTERNATE AREA]
[ACCESS MODE IS SEQUENTIAL] {USE | NO} WORK - AREA

SELECT file-name ASSIGN TO DATA-COMM OUT [RESERVE ALTERNATE AREA]
{USE | NO} WORK-AREA
```

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

DATA DIVISION

```
DATA DIVISION.
```

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

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

```
{OC / OCCURS} integer TIMES
```

```
{PC / PIC / PICTURE} IS (any allowable character string to describe the data).
```

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

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

```
{VA    } IS {up to 15 numeric digits          }
{VALUE }    {"up to 99 alpha characters       }
            {enclosed in quotes"              }
```

PROCEDURE DIVISION

PART A: BASIC VERBS AND CONSTRUCTS

Accept

OPTION 1

```
ACCEPT alpha-data-name [FROM {KEYBOARD      }]
                             {KEYBOARD-PRNTR}
```

OPTION 2

```
ACCEPT integer CHARACTERS [FROM KEYBOARD-PRNTR]
```

OPTION 3

```
ACCEPT numeric-data-name [FROM {KEYBOARD      }]
                               {KEYBOARD-PRNTR}
```

OPTION 4

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

OPTION 5

```
ACCEPT INTO ACCUMULATOR FROM KEYBOARD
```

OPTION 6

```
ACCEPT FROM KEYS
```

Add

OPTION 1

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

OPTION 2

```
ADD {numeric-data-name-1 | integer} numeric-data-name-2 GIVING numeric-data-name-3 [ROUNDED]
    [ON SIZE ERROR statements]
```

OPTION 3

```
ADD ACCUMULATOR TO numeric-data-name [ON SIZE ERROR statements]
```

OPTION 4

```
ADD ACCUMULATOR numeric-data-name-1 GIVING numeric-data-name-2 [ROUNDED]
    [ON SIZE ERROR statements]
```

OPTION 5

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

OPTION 6

```
ADD {numeric-data-name-1 | integer} ACCUMULATOR GIVING numeric-data-name-2 [ROUNDED]
    [ON SIZE ERROR statements]
```

OPTION 7

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

Advance

OPTION 1

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

OPTION 2

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

Alarm

```
ALARM
```

Close Handler

```
CLOSE TRANSPORT
```

Display

OPTION 1

```
DISPLAY {alpha-data-name} [UPON PRNTR]
        {"alpha-literal"}
```

OPTION 2

```
DISPLAY {"character"} [PREVIOUS-RIBBON]
        {QUOTE     }
```

OPTION 3

```
DISPLAY { "character" } { numeric-data-name } { NEGATIVE }
        { QUOTE       } { ACCUMULATOR     } { POSITIVE }
```

OPTION 4

```
DISPLAY numeric-data-name [UPON PRNTR]
```

OPTION 5

```
DISPLAY ACCUMULATOR [(integer)] numeric-data-name [UPON PRNTR]
```

Divide

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

Enable

```
ENABLE [table-name] PK1 ... PK24
```

End-of-Job

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

Exit

```
EXIT.
```

Go To

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

If

RELATIVE TESTS

OPTION 1

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

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

OPTION 2

```
IF { alpha-data-name   } IS { GREATER THAN  ( > )    } { alpha-data-name   }
   { non-numeric-literal}   { LESS THAN     ( < )    } { non-numeric literal}
                            { EQUAL TO      ( = )    }
                            { NOT EQUAL TO  ( NOT= ) }

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

ACCUMULATOR TESTS

OPTION 1

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

OPTION 2

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

ZERO TESTS

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

FORMS LIMIT TEST

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

ACCUMULATOR FLAG TESTS

```
IF { numeric-data-name } { NFLAG } [ { AND } { NFLAG } ... ] THEN { statements }
   { ACCUMULATOR   } { SFLAG }     { OR  } { SFLAG }          { NEXT SENTENCE }
                     { CFLAG }             { CFLAG }
                     { MFLAG }             { MFLAG }
  [ELSE statements]
```

SWITCH TESTS

Group A Switches

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

Group B Switches

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

OPERATION CONTROL KEYS (OCK) FLAG TESTS

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

Move

OPTION 1

```
MOVE { alpha-data-name   } TO alpha-data-name
     { non-numeric literal }
```

OPTION 2

```
MOVE { numeric-data-name } TO numeric-data-name
     { integer          }
     { ZERO             }
     { ACCUMULATOR      }
```

OPTION 3

```
MOVE { numeric-data-name } TO ACCUMULATOR
     { integer          }
     { ZERO             }
```

OPTION 4

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

OPTION 5

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

OPTION 6

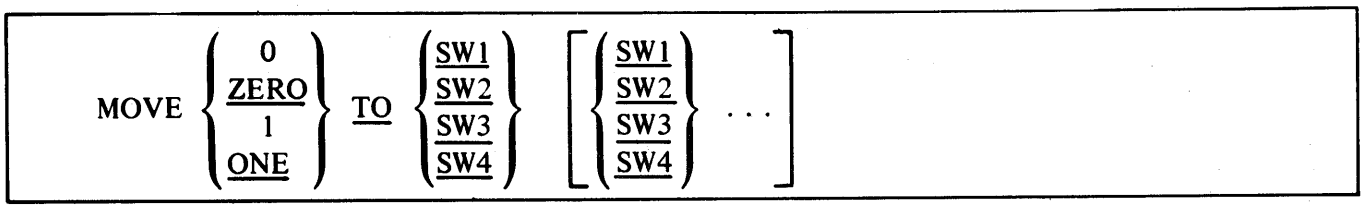
```
MOVE REMAINDER TO { ACCUMULATOR    }
                  { numeric-data-name }
```

OPTION 7

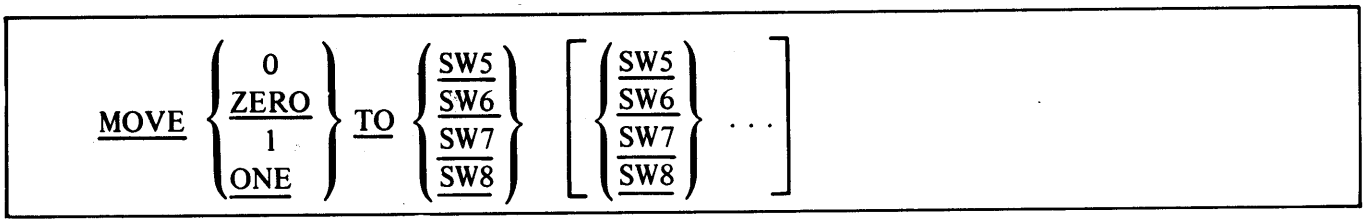
```
MOVE { 0    } TO { NFLAG } [ { NFLAG } ... ]
     { ZERO }    { SFLAG }   { SFLAG }
     { 1    }    { CFLAG }   { CFLAG }
     { ONE  }    { MFLAG }   { MFLAG }
```

OPTION 8

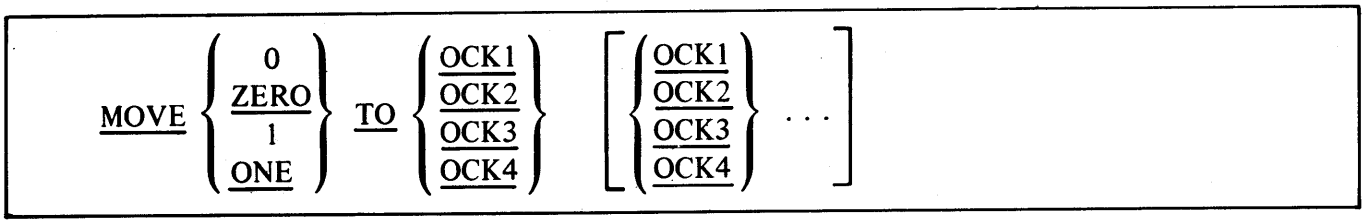
Group A



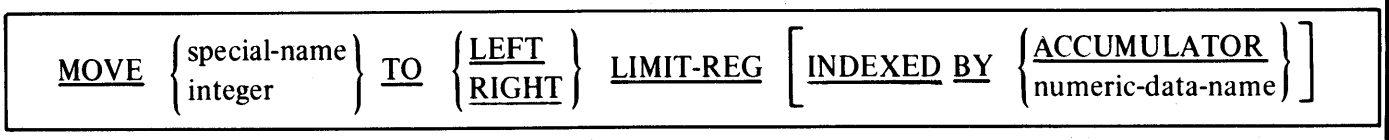
Group B



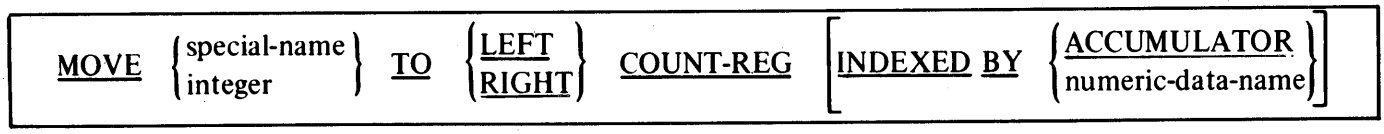
OPTION 9



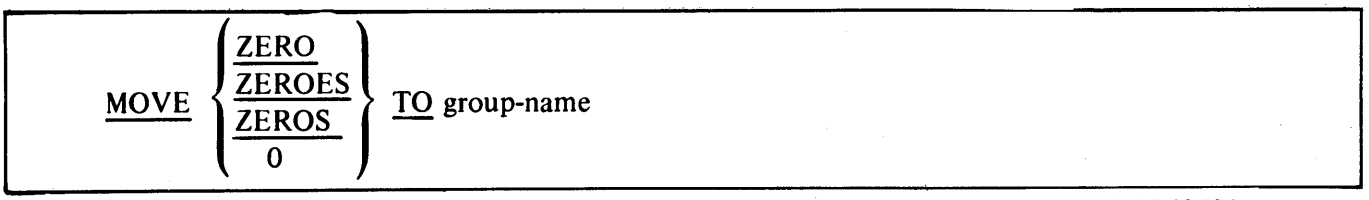
OPTION 10



OPTION 11



OPTION 12



Multiply

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

No-op

```
NO-OP
```

```
NOTE sentence.
```

```
NOTE. paragraph
```

Open

```
OPEN HANDLER [{integer | special-name}] [INDEXED BY {ACCUMULATOR | numeric-data-name}]
```

Perform

```
PERFORM procedure-name
```

Position

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

Red-Ribbon

```
RED-RIBBON
```

Stop Run

```
STOP RUN
```

Subtract

OPTION 1

```
SUBTRACT {numeric-data-name-1} FROM {numeric-data-name-2}
         {integer              }      {ACCUMULATOR      }

    [GIVING numeric-data-name-3 [ROUNDED]] [ON SIZE ERROR statements]
```

OPTION 2

```
SUBTRACT ACCUMULATOR FROM numeric-data-name-1 [GIVING numeric-data-name-2
    [ROUNDED]] [ON SIZE ERROR statements]
```

Use

OPTION 1

```
USE FOR PK-TABLE table-name
```

OPTION 2

```
USE FOR SUBROUTINE procedure-name
```

PART B: PAPER TAPE I/O

Accept

OPTION 1

```
ACCEPT alpha-data-name FROM { 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 { numeric-data-name                          } FROM { KEYBOARD-PCH
       { INTO ACCUMULATOR numeric-data-name }              { RDR
                                                           { RDR-PRNTR
                                                           { RDR-PCH
                                                           { RDR-PRNTR-PCH
                                                           { KYBRD-PRNTR-PCH }
```

OPTION 4

```
ACCEPT INTO ACCUMULATOR FROM RDR
```

Display

OPTION 1

```
DISPLAY { alpha-data-name   } UPON { PRNTR-PCH }
        { "non-numeric literal" }  { PCH       }
```

OPTION 2

```
DISPLAY numeric-data-name UPON { PRNTR-PCH }
                               { PCH       }
```

OPTION 3

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

OPTION 4

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

OPTION 5

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

If

OPTION 1

```
IF { RDR-ERR    } THEN { statements     } ELSE statements
   { RDR-COND   }      { NEXT SENTENCE }
```

OPTION 2

```
IF { PCH-ERR    } THEN { statements     } ELSE statements
   { PUNCH-OFF  }      { NEXT SENTENCE }
   { NO-MEDIA   }
   { LOW-TAPE   }
```

Move

OPTION 1

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

OPTION 2

```
MOVE integer TO PCH-REG
```

Open

```
OPEN MEDIA-CLAMP
```

PART C: 80-COLUMN CARD I/O

Accept

OPTION 1

```
ACCEPT alpha-data-name FROM { KEYBOARD-PCH
                            { KYBRD-PRNTR-PCH
```

OPTION 2

```
ACCEPT integer CHARACTERS FROM KYBRD-PRNTR-PCH
```

OPTION 3

```
ACCEPT numeric-data-name FROM { KEYBOARD-PCH
                              { KYBRD-PRNTR-PCH
```

OPTION 4

```
ACCEPT INTO ACCUMULATOR numeric-data-name { KEYBOARD-PCH
                                          { KYBRD-PRNTR-PCH
```

Display

OPTION 1

```
DISPLAY { alpha-data-name } UPON { PRNTR-PCH }
        { "literal"       }      { PCH      }
```

OPTION 2

```
DISPLAY numeric-data-name UPON { PRNTR-PCH }
                               { PCH      }
```

OPTION 3

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

OPTION 4

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

OPTION 5

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

If

OPTION 1

```
IF RDR-ERR THEN { statements / NEXT SENTENCE } ELSE statements
```

OPTION 2

```
IF PCH-ERR THEN { statements / NEXT SENTENCE } ELSE statements
```

OPTION 3

```
IF PCH-OFF THEN { statements / NEXT SENTENCE } ELSE statements
```

Read

```
READ file-name
```

Select

OPTION 1

```
SELECT ALTERNATE STACKER
```

OPTION 2

```
SELECT SKIP FUNCTION TO { special-name }
                        { integer      }
```

OPTION 3

```
SELECT REPEAT FUNCTION { THROUGH } { special-name }
                       { THRU    } { integer      }
```

NO WORK-AREA DECLARED IN FILE CONTROL

Display

```
DISPLAY alpha-file-data-name [FROM BUFFER] UPON { PRNTR     }
                                               { PRNTR-PCH }
                                               { PCH       }
```

Move

OPTION 1

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

OPTION 2

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

USE WORK-AREA DECLARED IN FILE-CONTROL

Fill

```
FILL record-name.
```

PART D: DATA COMMUNICATIONS CAPABILITY

Accept

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

Display

```
DISPLAY alpha-file-data-name [FROM BUFFER] UPON PRNTR
```

Fill

```
FILL record-name.
```

If

OPTION 1

```
IF XMT-RDY THEN statements [ELSE statements]
```

OPTION 2

```
IF RCV-RDY THEN statements [ELSE statements]
```

OPTION 3

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

OPTION 4

```
IF DC-ERROR THEN statements [ELSE statements]
```

OPTION 5

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

Locate

```
LOCATE file-name
```

Move

OPTION 1

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

OPTION 2

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

OPTION 3

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

OPTION 4

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

OPTION 5

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

OPTION 6

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

OPTION 7

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

OPTION 8

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

OPTION 9

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

OPTION 10

```
MOVE (a)ab(a) TO DATA-COMM
```

Read

```
READ file-name
```

Stop Machine

```
STOP MACHINE
```

Write

```
WRITE record-name
```

PART E CHECK DIGIT CAPABILITIES

Convert

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

If

```
IF { ACCUM data-name } CHECK-DIGIT [(integer)] [FROM data-name-2]
   { data-name 1     }
   [TRUNCATED] THEN statements [ELSE statements]
```

PART F STERLING CAPABILITIES

Convert

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

Display

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

If

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

Round

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

PART G TC 700

If

```
IF { TELLER-1
   | TELLER-2
   | SUPERVISOR
   | PB-FIRST-LINE
   | PB-LAST-LINE
   | PB-FOLD
   | PB-NOT-PRESENT } THEN statements [ELSE statements]
```

Move

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

SERIES L/TC RESERVED WORD LIST

ACCEPT
ACCEPTING
ACCESS
ACCUM
ACCUMULATOR
ACUM
ADD
ADVANCE
ALARM
ALIGN
ALPHA
ALPHANUMERIC
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
COLUMNS
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
REVERSE
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
SPECIFIED
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
TRANSPORT
TRUNCATED
TWO-WIRE-CNTL