UNIVAC 1050 SYSTEMS

MAGNETIC TAPE SYSTEM

PROVISIONAL DOCUMENT

PAL TAPE ASSEMBLER

UP 3940.5

This document is provisional in nature and is intended as a vehicle for meeting immediate needs with regard to system familiarization and orientation. UNIVAC® Division of Sperry Rand Corporation reserves the right to change and/or modify such information contained herein as may be required by subsequent system developments.

ILLUSTRATIONS

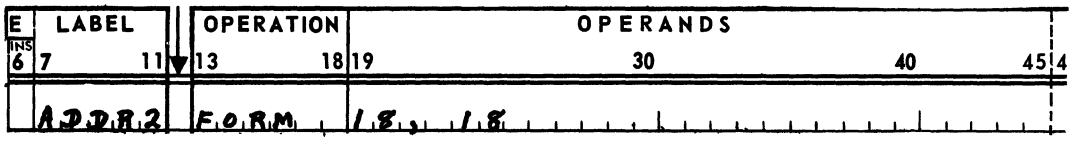
The PAL Tape Assembler is an extension of the PAL Card Assembler. It uses the same instruction repertoire. Because of the tape control system, however, additional operations are required when using the Halt, Then Jump and Jump Display instructions. These operations are explained in the description of the Executive Routine.

The data generation procedure FORM provides the programmer with the facility of arranging data in any desired manner. The rules for the use of the procedure FORM are as follows:

- The FORM line must have a label.
- The word FORM must appear in the operation field.
- The operands field of the FORM line may contain from one to eight expressions. Each expression is either a decimal number not greater than 48, the letter T, or the letter X. Each expression specifies the number of bit positions required for the value that is represented by the expression in the line to which FORM will apply.
- The total of the expressions entered in the operands field must not exceed 96 and must be a multiple of 6. Each T has a value of 6, and each X has a value of 3 (see following page).
- The FORM line must precede all lines to which it is applicable.
- A succeeding line to which a FORM line is to apply must contain:
  a. the label entry of the FORM line in its operation field;
  b. as many expressions in its operands field as there are expressions in the operands field of the FORM line. The labels appearing in this operands field reference the <u>least significant</u> character of a field.

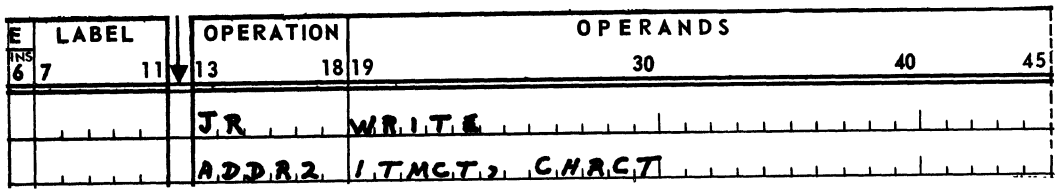
To illustrate the use of a FORM line, assume that a program is using a subroutine that writes a block on tape whenever a JR instruction to the label WRITE is executed. The subroutine requires that whenever such a JR instruction is written, the following line must specify the addresses of two counters which are contained in a six character field. The first 18 bits of the 36 bit field must contain the address of the counter the program uses to count the number of records in each block; the following 18 bits must contain the address of the counter used to count the number of characters in each block.

The FORM line is written as follows:



When the assembler encounters the entry ADDR2 in the operation field of a succeeding line, it will be supplied with two expressions, the values of which are to be arranged in two contiguous 18 bit fields.

Communication with the subroutine is accomplished by writing



where ITMCT is the label of the record counter, and CHRCT is the label of the character counter. When the program is assembled, the six characters immediately following the JR instruction will contain the addresses of these two counters.

It should be noted that the subroutine whose first instruction is labeled WRITE must contain some provision to increment the M portion of that instruction by 6, in order that control be returned to the next instruction in sequence, bypassing the ADDR2 line.

The T and X entries on a FORM line specify special treatment of the corresponding expressions on any succeeding line to which the FORM line applies. If these corresponding expressions are labels, the assembler handles them as follows:

- T specifies that the value of the defined label is to be divided by four, and that a six-bit quotient is to be produced.
- X specifies that thirty-two is to be subtracted from the value of the defined label, and that the difference is to be divided by four, producing a three-bit quotient.

In both cases, after the arithmetic operations have been performed on the label, only the quotient is retained, and the remainder is ignored.

A T entry, therefore, directs the assembler to convert a label (which is equated to a fifteen-bit binary address) into a tetrad number, while an X entry directs the assembler to convert a label into an index register number.

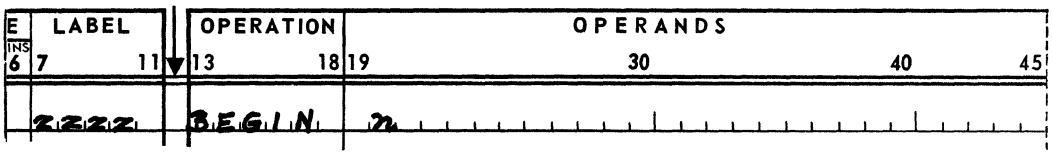
Assembler directives are devices used by the programmer to supply information to the PAL Assembler. The information supplied by these directives controls the disposition of the program, program instructions, or storage areas required by the program. Although they affect the assembly of program instructions, the lines in which these assembler directives appear are not assembled as instructions.

The following assembler directives are available with the Tape System in addition to those with the Card System - ORIG, END, EQU, and AREA.

A. BEGIN

Every program to be assembled must have the assembler directive BEGIN in the operation field of the first line. The BEGIN directive causes the start of the assembly process.

A program name of up to six characters may be in columns 7 through 12 of the BEGIN line. When the program is assembled on tape, however, only the first four characters (7 through 10) are used for program identification in the tape header block. If the program is to be loaded in segments, the program name is incremented with each segment (see Section 4-C, page 23).



zzzz may be any value that does not start with X.

n directs the loading of the program. Where n

= an absolute address of 520 or greater, the assembler assigns the value as the absolute address of the first character of a program. Only one absolute program may be loaded. The executive will not load a second program of either the absolute or relative type.

= 3 a relative program will be loaded. If the 80 column version executive is used, the first program (A) will be loaded at the beginning of the first available row following the Executive Routine. If the 90 column version executive is used, it will be loaded, behind the executive, at the beginning of the first available even numbered row.

If a concurrent version of the executive is used and two programs (A and B) are to be run, the second program (B), loaded with zzzz BEGIN 3, will be placed in the high end of storage.

= 5 program A will be loaded just as described for 3. However, program B, loaded with zzzz BEGIN 5 is placed in storage directly behind program A.

B. PROC, DO, and NAME

The user is urged to devote careful attention to this section. The judicious construction and use of procedures which reflect particular installation computational needs enable the user to build an installation-oriented assembler-compiler. The primary significance of this feature is the reduction of the time expended in coding, thereby freeing programmers for the more creative tasks of problem analysis and problem solving.

The assembler directives PROC, DO, and NAME allow the programmer to incorporate procedures in a program. A procedure is a series of instructions which the programmer anticipates writing several times. To save himself the time and effort required to write a series of instructions repeatedly, the programmer writes them once in a procedure definition. Each time that the same series of instructions is required, the programmer simply writes a procedure reference line. At that point, the assembler inserts the coding lines contained within the procedure.

By means of a procedure definition, the programmer gives the assembler the fixed portions of an operation only once; whenever the operation is required, the programmer calls it into the program by writing a reference line in which he supplies the variable portions. The assembler then combines the fixed portions with the variable portions to produce instructions which are then inserted into the program at the point where the reference line appears.

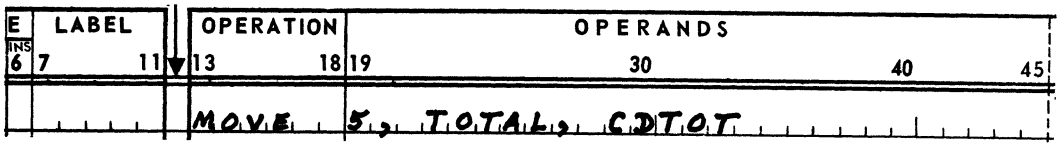
The programmer may write the reference line calling a procedure anywhere in the program as frequently as desired. The procedure definition, however, appears only once.

1. PROC

The procedure <u>reference line</u> contains

- a label (optionally) in the label field,
- the name of the called procedure in the operation field, and
- the appropriate parameters in the operands field.

For example, a reference line for a procedure named MOVE is



The procedure definition consists of a number of coding lines, the first of which contains

- the procedure name in the label field,
- the assembler directive PROC in the operation field, and
- an expression in the operands field (not required).

The last line of the procedure definition is an END directive.

The PROC line identifies all following lines, up to the first END line, as being part of the same procedure named in the label field. The purpose of the expression in the operands field of the PROC line will be explained under the discussion of the directive NAME.

The following limits are placed on the use of the PROC assembler directive.

a. A procedure definition must have been defined before it may be referenced.

b. A procedure definition may not appear within another procedure definition.

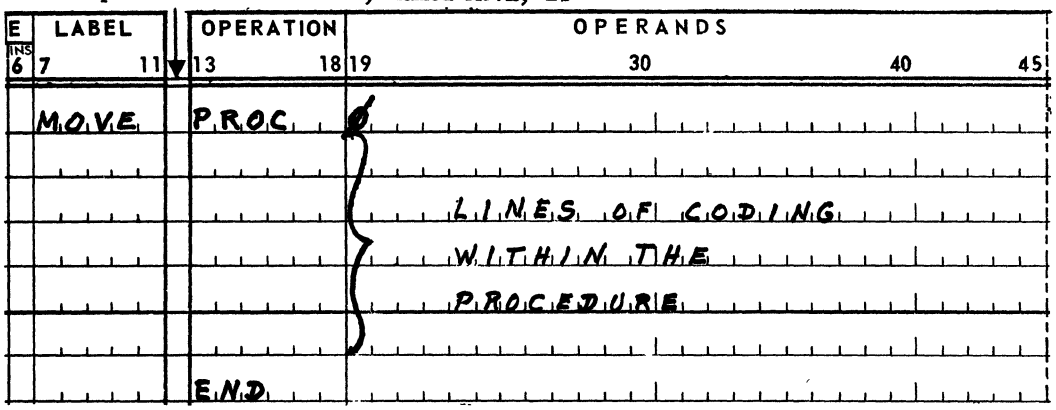
c. A procedure definition may not appear within the range of a DO line (see Section 4-B-2).

d. A full comments line, i.e., one with a period in column 7, may appear within a procedure definition, as well as within the range of a DO line. Comments may be written after the last expression on a line, but will not appear in the listing of generated coding lines.

e. Reference to a library routine or a procedure may not appear in the operation field of any line within a procedure definition.

f. A FORM definition within a procedure definition may be referenced only by lines within the procedure definition. However, lines within the procedure may reference FORM statements external to the procedure definition.

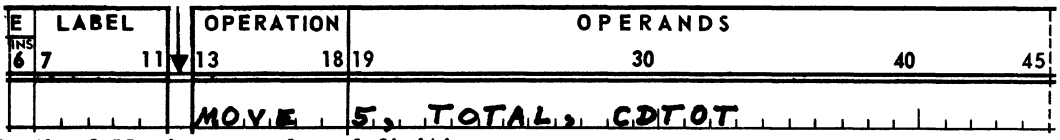
An example of a procedure definition, named MOVE, is



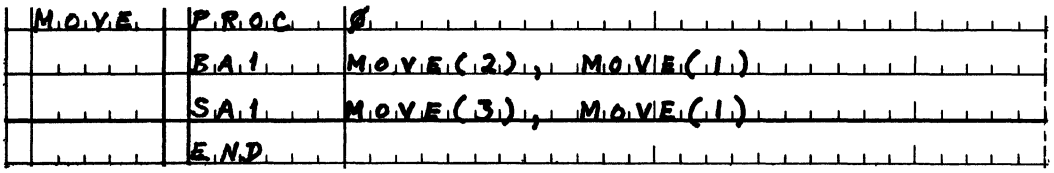
PROVISIONAL

Fixed portions of a procedure definition are written exactly as the programmer wishes them to appear in the generated coding. Variable portions, which in the generated coding are to be replaced by the parameters in the reference line, are represented by indexed labels. An indexed label is the procedure name followed by a decimal number enclosed in parentheses. This number defines which parameter supplied by a reference line is to replace the indexed label. For example, the indexed label MOVE (3) is replaced in the generated coding by the third expression on a reference line.

A complete procedure reference line and the procedure definition that it calls is written as follows. The reference line



calls the following procedure definition



Line 1 is a PROC line, identifying all succeeding lines up to the next END line as being part of the MOVE procedure.

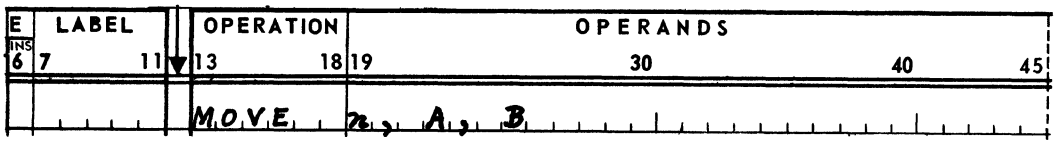
Lines 2 and 3 are instructions written in generalized form. The M and L expressions of these instructions are indexed labels.

Line 4 is an END directive, indicating that there are no more lines in this procedure.

When the reference line is assembled, the assembler replaces every MOVE (1) in the definition with 5, every MOVE (2) with the label TOTAL, and every MOVE (3) with the label CDTOT, and the following coding is generated.

```
BA1     TOTAL, 5
SA1     CDTOT, 5
```

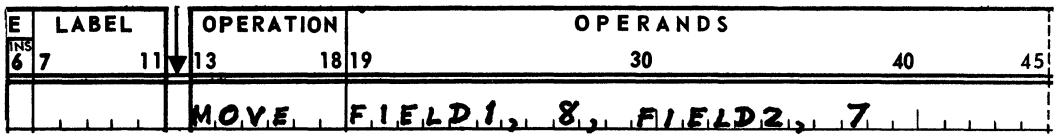
Every time the programmer wishes to transfer a field of n characters from A to B using AR1, therefore, he does not have to code each instruction line; instead, all he has to do is write



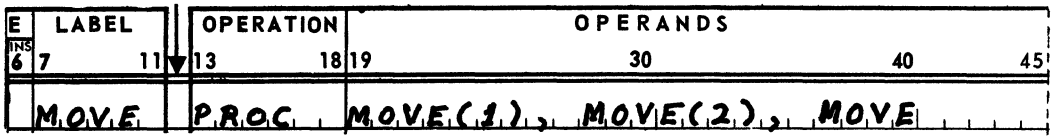
and the assembler supplies the correct coding in the object program and on the output listing.

Another way that a variable may be provided in a procedure definition is by putting the unindexed procedure name in the operands field. When the procedure is assembled the name is replaced by a number that equals the number of parameters in the reference line.

For example, the reference line



with the procedure definition



causes the following coding to be generated.

```
BA1          FIELD1, 8, 4
```

2. DO

The DO directive causes a number of lines to be repeated a certain number of times.

A DO directive contains

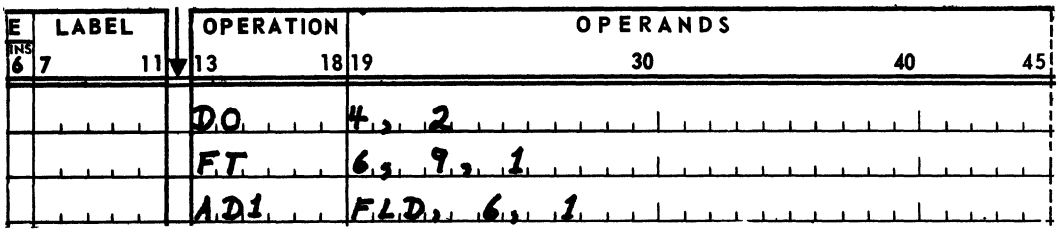
- an A (optionally) in the label field which acts as a counter. A further explanation of the label is given separately.
- the assembler directive DO in the operation field.
- two expressions in the operands field:
  - The first expression specifies the number of times that the DO is to be executed.
  - The second expression specifies the number of lines immediately following the DO line that will be repeated.

The DO directive may exist either inside or outside of a procedure. However, when used in conjunction with a PROC directive, it imparts a greater degree of flexibility to a procedure.

The following limits are placed on the use of the DO directive.

- The maximum number of nested DO's, i.e., DO's within DO's, within DO's, is ten. The assembler will not process more than ten levels of DO's at one time.
- Labels may not appear in the operands field of a DO line, with the exception of the label of the same or another DO line.
- References to a library routine or a procedure may not appear in the operation field of any line within a DO.

In the following example the DO line directs the assembler to repeat the next two lines, four times.



After the assembler has processed the above DO line, the DO line and the next two lines will be replaced by

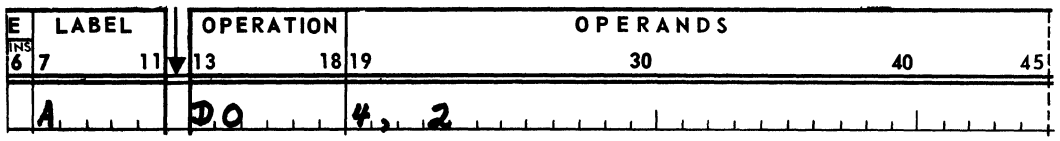
```
FT      6,      9,      1
AD1     FLD,    6,      1
FT      6,      9,      1
AD1     FLD,    6,      1
FT      6,      9,      1
AD1     FLD,    6,      1
FT      6,      9,      1
AD1     FLD,    6,      1
```

a. The Label Field

The DO line may have an entry in the label field, however, it is not treated as a label. The "label" of a DO line serves as a counter. Its use is to allow the programmer to vary the number of parameters which he supplies to a single procedure, and to allow the procedures to handle a variable number of parameters.

When the procedure is first entered, the "label" of a DO line originally has a value of 1. Each time the DO is executed, this value is incremented by 1.

When the assembler processes the line



the first time A has a value of 1. In the second repetition of the DO, A has a value of 2; in the third repetition, its value is 3; and in the fourth repetition, its value is 4. After the fourth repetition of the DO, the assembler has completed the DO. The value of the DO "label" is available for examination, i.e., the programmer can test the value of the counter at any time prior to the occurrence of another DO line that has an entry in the label field.

If the indexed label MOVE(A+3) appears in the procedure, this indexed label is MOVE(4) in the first iteration of the DO; MOVE(5) in the second; and so on.

This example illustrates the use of a DO within a procedure and the DO label .

ENCE LINE 4 5	INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
			MOVE	2, 5, TOTAL, CDTOT, GRPAY, NTPAY

Figure 1. PROC Reference Line.

01	MOVE	PROC	0
02	A	DO	MOVE(1), 2
03		BA1	MOVE(A+A+1), MOVE(2)
04		SA1	MOVE(A+A+2), MOVE(2)
05		END	

Figure 2. PROC Definition Called by the Reference Line.

01	BA1	TOTAL, 5
02	SA1	CDTOT, 5
03	BA1	GRPAY, 5
04	SA1	NTPAY, 5

Figure 3. Resultant Lines Produced from the PROC Reference.

This procedure generates the coding required to transfer any number of fields of the same length (up to sixteen characters).

The parameters required by a reference line calling this PROC definition are:

(1) The first parameter specifies the number of Bring-Store instruction pairs required;

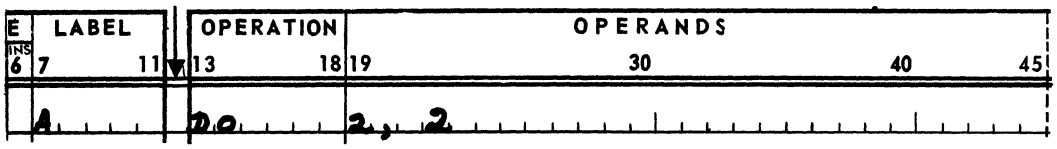
(2) the second parameter specifies the size of the fields which must all be the same length; and

(3) as many pairs of addresses (label of the sending field followed by the label of the receiving field) as there are moves specified.

When the assembler encounters the procedure reference line (Fig. 1) the MOVE procedure (Fig. 2) is referenced. The second line of the procedure is a DO statement with a label A. The value of the label at this point is one.

The DO statement directs the assembler to repeat the next 2 lines, MOVE(1) times. The reference line supplies 2 for MOVE(1). For the purpose of this reference line, the DO

statement is equivalent to



The first time that the DO is executed, since A has a value of 1, the indexed label MOVE (A+A+1) is equivalent to MOVE(3), which is the label TOTAL. The first line of Figure 3 is generated. The indexed label MOVE(A+A+2) is equal to MOVE(4), and the second line of Figure 3 is generated.

In the second iteration of the DO, A has a value of 2. When the assembler processes MOVE (A+A+1) and MOVE(A+A+2), the indexed labels have the values MOVE(5) and MOVE(6), respectively, resulting in the generation of lines 3 and 4 of Figure 3.

Note that the first indexed labels appearing on lines 3 and 4 of Figure 3 used A+A to represent twice the value of A. This is the only legitimate way to represent a multiple of the value of a DO label; 2A is illegal.

Figure 4 shows the output listing from an actual assembly of the same procedure call.

```
ERROR  LOCATION  SIZE  Ix        DATA          LINE  LABEL  OP     OPERANDS              COMMENTS  PROGRAM-ID
         000600                                00101  MOVER  BEGIN  0600                            MOVER
         000005                                00102  IN1    EQU    5                               MOVER
         000012                                00103  IN2    EQU    10                              MOVER
         000017                                00104  OUT1   EQU    15                              MOVER
         000024                                00105  OUT2   EQU    20                              MOVER
                                               00106  MOVE   PROC   0                               MOVER
                                               00107  A      DO     MOVE (1), 2                     MOVER
                                               00108         BAI    MOVE (A+A+1), MOVE (2)          MOVER
                                               00109         SAI    MOVE (A+A+2), MOVE (2)          MOVER
                                               00110         END                                    MOVER
                                               00111         MOVE   2,5,IN1,OUT1,IN2,OUT2           MOVER
         000600                 5600000505       *           BAI    IN1,5
         000605                 5200001205       *           SAI    OUT1,5
         000612                 5600001705       *           BAI    IN2,5
         000617                 5200002405       *           SAI    OUT2,5
                                3000060000    00112         END    0600                            MOVER
```

Figure 4. Sample Procedure and Reference Line Listing.

b. Conditional DO

A form of the DO directive, known as a conditional DO, is executed only if a specified condition exists. The first expression of such a DO line is a combined expression, consisting of two expressions related by one of the following symbols:

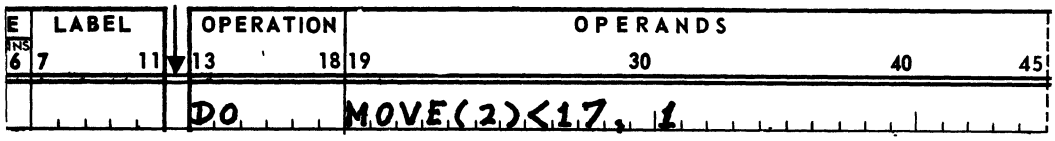
= (the first expression is equal to the second)

≠ (the first expression is not equal to the second)

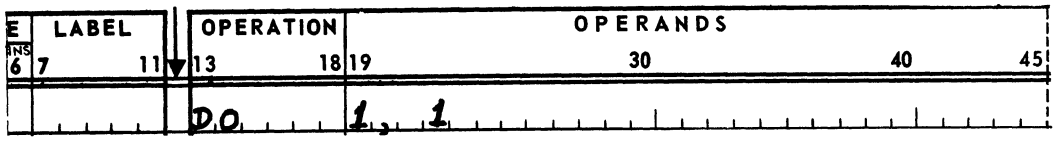
> (the first expression is greater than the second)

< (the first expression is less than the second)

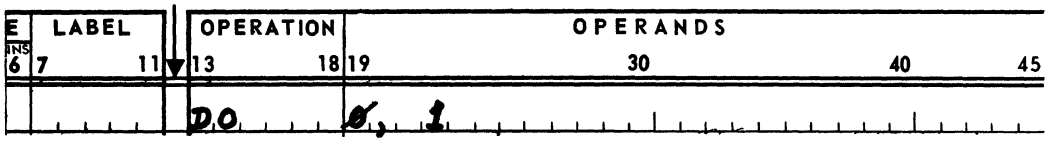
An example of a conditional DO in a PROC definition is



If MOVE(2) is less than 17, the condition is true and the first expression is replaced by a 1, effectively making the line



which results in the generation of the one line immediately following it. If MOVE(2) is greater than or equal to 17, however, the condition is not true, and the line becomes



and the DO is not executed, since the assembler is directed to copy the next one line, zero times.

The following example is an expansion of the MOVE procedure given above. By means of a conditional DO, the procedure determines whether the length of the operands is greater than 16. If not, the transfer(s) will be effected through AR1; if the operands are 17 or more characters in length, coding for Block Transfer(s) will be generated.

The reference lines calling this procedure are of the same format as that shown in Figure 1.

If the reference line is

SEQUENCE LINE 4 5	INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
			MOVE	3, 8, ACCT1, ACCTA, ACCT2,

and the procedure is

SEQUENCE LINE 4 5	INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
0.1		MOVE	PROC	Ø
0.2			DO	MOVE(2)<17, 1
0.3		A	DO	MOVE(1), 2
0.4			BA1	MOVE(A+A+1), MOVE(2)
0.5			SA1	MOVE(A+A+2), MOVE(2)
0.6			DO	MOVE(2)>16, 2
0.7			FT	MOVE(2), 18
0.8		B	DO	MOVE(1), 2
0.9			FT	MOVE(B+B+1), 17
1.0			TTR	MOVE(B+B+2)
1.1			END	

The following coding is generated:

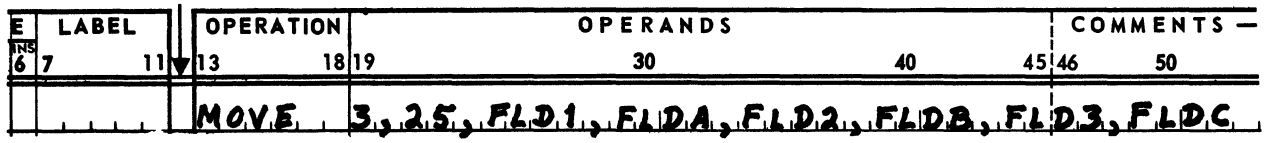
```
BA1     ACCT1,    8
SA1     ACCTA,    8
BA1     ACCT2,    8
SA1     ACCTB,    8
BA1     ACCT3,    8
SA1     ACCTC,    8
```

Line 2 of the procedure is a conditional DO. The assembler examines parameter 2, and since the condition is satisfied, this expression is replaced by a 1. The procedure therefore directs the assembler to DO the next line once.

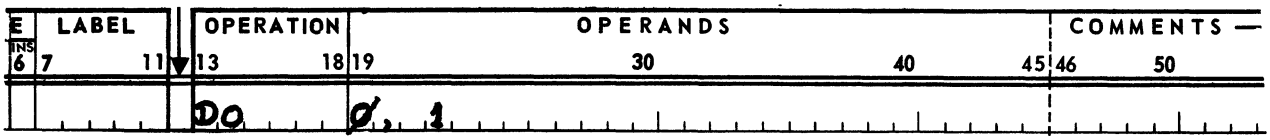
The next line is an unconditional DO. All of the coding lines falling within the range of this DO, i.e., the next two lines, are counted as part of the conditional DO. Although the conditional DO specifies repeat one line, it actually controls three lines due to the following unconditional DO.

The next conditional DO appears on line 6. Again the assembler compares the second parameter of the reference line against 16. It is less than 16 and therefore is replaced with zero(0). The two lines falling within the range of this DO, therefore, are not copied. The second line, within the range of this conditional DO, is another DO which controls two more lines.

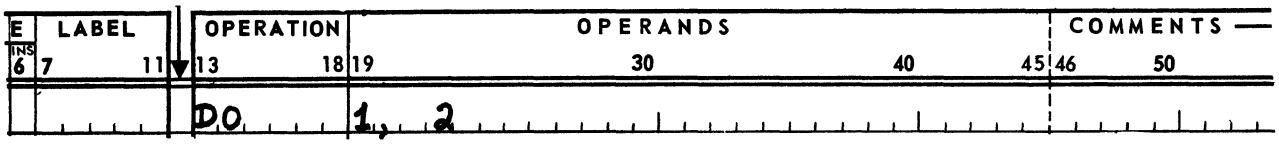
If the reference line is



the conditional DO on line 2 becomes



and everything within its range is skipped, taking the assembler down to the conditional DO on line 6. This becomes

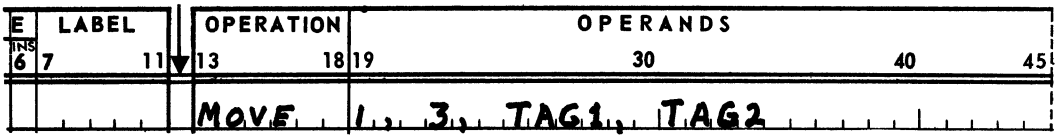


because the second parameter of the reference line is greater than 16, and the following is generated

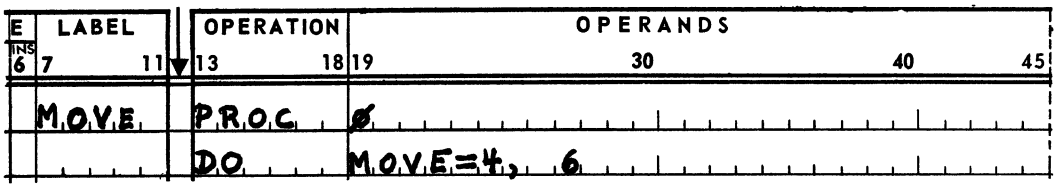
```
FT      25, 18
FT      FLD1, 17
TTR     FLDA
FT      FLD2, 17
TTR     FLDB
FT      FLD3, 17
TTR     FLDC
```

A conditional DO can also be written within a procedure through the use of the unindexed procedure name. As already explained in Section 4-B-1, an unindexed procedure name in the procedure definition is treated as a number equal to the number of parameters in the procedure reference line.

For example, with



the condition of the following DO directive is met because there are four parameters in the procedure reference line



3. NAME

The assembler directive NAME allows the programmer to assign alternate procedure names to a procedure definition. The purpose of this is twofold: first, a pseudo-operation, i.e., the operation field entry in a reference line, can be rendered more meaningful; and second, the assembler evaluates a parameter within the procedure definition, rather than on the reference line, as a basis for processing conditional DO's. The parameter that is evaluated is parameter 0, which was mentioned in the discussion of PROC.

Parameter 0 is the single expression which appears on a PROC line. Whenever a procedure is referenced by means of the label of the PROC line, the value of parameter 0 is the expression appearing in the operands field. Whenever the same procedure is referenced by the label of a NAME line, parameter 0 of that reference line is the expression appearing in the operands field of the NAME line.

A NAME line is written immediately following the PROC line of the procedure for which it is an alternate name; or immediately following a NAME line naming the same procedure. A maximum of ten NAME lines are allowed within a procedure definition. A NAME line consists of

- an alternate name for a procedure, written in the label field;
- the entry NAME in the operation field; and
- an expression in the operands field.

Figure 5 shows a procedure definition for a procedure which may be referenced by any of the following names: MOVE, TRFR, TRFI, TRTR, and TRTI. Whenever this procedure is referenced by a line containing MOVE in the operation field, the value of MOVE(0) is 0; whenever it is referenced by the entry TRFR in the operation field, the value of MOVE(0) is 1; and so on.

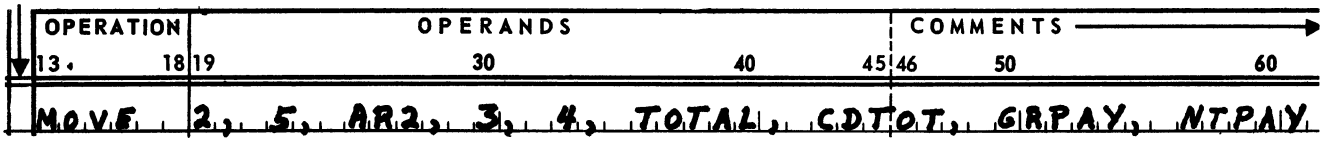
SEQUENCE PAGE 1 3	LINE 4 5	INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45	COMMENTS 46 50 60 70
	0.1		MOVE	PROC	Ø	
	0.2		TRFR	NAME	1	TRANSFER FROM, RESET BASE
	0.3		TRFI	NAME	2	TRANSFER FROM, INCREMENT BASE
	0.4		TRTR	NAME	3	TRANSFER TO, RESET BASE
	0.5		TRTI	NAME	4	TRANSFER TO, INCREMENT BASE
	0.6			DO	MOVE(Ø)=Ø,2	
	0.7			DO	MOVE(3)=AR1, 1	
	0.8		A	DO	MOVE(1), 2	
	0.9			BA1	MOVE(A+A+4),MOVE(2),MOVE(4)	
	1.0			SA1	MOVE(A+A+5),MOVE(2), MOVE(5)	
	1.1			DO	MOVE(3)=AR2, 1	
	1.2		B	DO	MOVE(1), 2	
	1.3			BA2	MOVE(B+B+4),MOVE(2)MOVE(4)	
	1.4			SA2	MOVE(B+B+5),MOVE(2)MOVE(5)	
	1.5			DO	MOVE(Ø)≠Ø, 7	
	1.6			FT	MOVE(1),18	SET CHARACTER COUNT IN TET. 18
	1.7			DO	MOVE(Ø)<3, 1	
	1.8			FT	MOVE(3), 16, MOVE(5)	SET DESTINATION ADDR
	1.9			DO	MOVE(Ø)=1, 1	
	2.0			TFR	MOVE(2),,MOVE(4)	TRANSFER FROM, RESET BASE
	2.1			DO	MOVE(Ø)=2, 1	
	2.2			TFI	MOVE(2),,MOVE(4)	TRANSFER FROM, INCREMENT BASE
	2.3			DO	MOVE(Ø)>2, 1	
	2.4			FT	MOVE(2), 17, MOVE(4)	SET ORIGIN ADDR
	2.5			DO	MOVE(Ø)=3, 1	
	2.6			TTR	MOVE(3),,MOVE(5)	TRANSFER TO, RESET BASE
	2.7			DO	MOVE(Ø)=4, 1	
	2.8			TTI	MOVE(3),,MOVE(5)	TRANSFER TO, INCREMENT BASE
	2.9			END		

Figure 5. MOVE Procedure

The format of a reference line calling this procedure by the name MOVE is as follows:

- MOVE(0) is 0;
- MOVE(1) is a decimal integer indicating the number of times that a move is to be executed;
- MOVE(2) is the number of characters, up to a maximum of sixteen, that will be moved on each transfer;
- MOVE(3) is an expression of the form ARn, where n is 1 or 2, specifying the arithmetic register to be used in effecting the transfer;
- MOVE(4) is the index register modifying all origin addresses;
- MOVE(5) is the index register modifying all destination addresses; and
- all succeeding pairs of parameters are pairs of addresses naming the origin and destination addresses of each transfer.

When the following reference line is written



the following coding is generated:

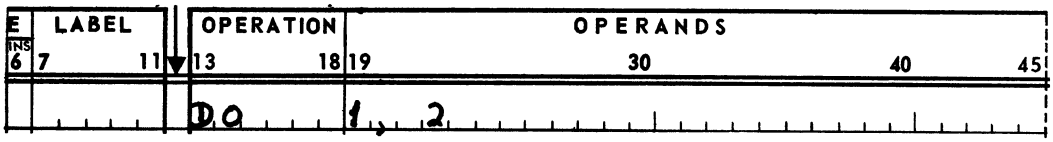
```
(1)  BA2    TOTAL,  5,  3
(2)  SA2    CDTOT,  5,  4
(3)  BA2    GRPAY,  5,  3
(4)  SA2    NTPAY,  5,  4
```

Figure 6. Coding Generated from the Illustrated MOVE Call.

The generation proceeds as follows:

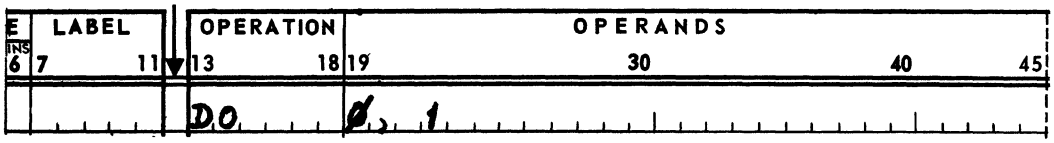
a. When the programmer calls for the procedure by the name MOVE, the assembler is directed to equate parameter 0 to 0.

b. When line 6 of the procedure definition is evaluated, parameter 0 is found to be equal to 0, effectively making this line



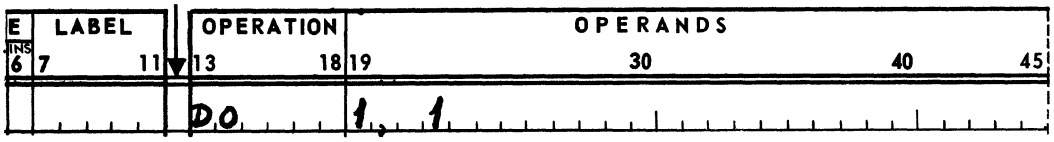
The two lines controlled by this DO are line 7 and line 11. Line 7 controls the one line on line 8, which in turn controls the two lines on lines 9 and 10. As far as line 6 is concerned, lines 8, 9, and 10 are a part of line 7.

c. Line 7 is evaluated next. The assembler is directed to test parameter 3 against AR1. Parameter 3 is not equal to AR1, which makes line 7 equivalent to

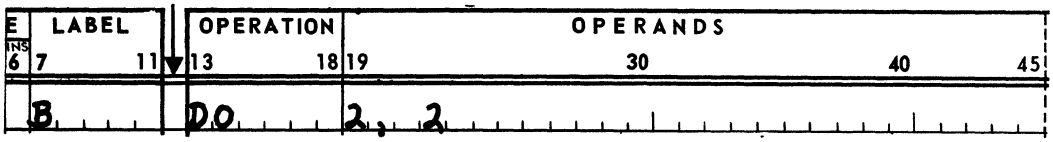


The assembler therefore bypasses line 7, as well as lines 8, 9, and 10, which are all treated as one line.

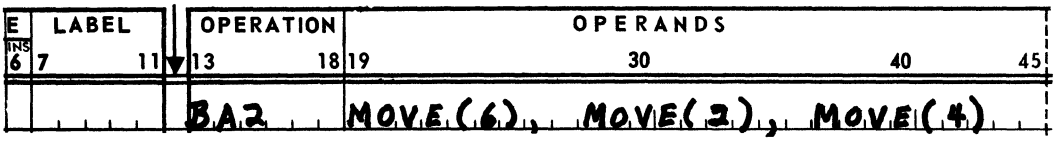
d. Upon evaluating line 11, parameter 3 is found to be AR2, making this line



e. Line 12 is then executed. It is an unconditional DO, directing the assembler to copy the next two lines, MOVE(1) times. As this reference line is processed, MOVE(1) is replaced by the supplied parameter, making this line



f. In the first iteration of the DO, B has a value of 1. Line 13, therefore, is processed as if it read



and when the indexed labels are replaced by the parameters supplied on the reference line, line 1 of Fig. 6 is generated. Line 14 becomes

E INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
		SA2	MOVE(7), MOVE(2), MOVE(5)

resulting in the generation oi line 2 of Fig. 6.

g. In the second iteration of the DO, B has a value of 2, and lines 13 and 14 become

E INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
		BA2	MOVE(8), MOVE(2), MOVE(4)
		SA2	MOVE(9), MOVE(2), MOVE(5)

and lines 3 and 4 of Fig. 6 are generated. The DO has now been executed MOVE(1) times. The assembler proceeds to the next line of the procedure definition.

h. Line 15 also calls for an evaluation of MOVE(0). Since parameter 0 is equal to 0, the condition of inequality is not satisfied, and line 15 becomes

E INS 6	LABEL 7 11	OPERATION 13 18	OPERANDS 19 30 40 45
		DO	ø, 7

The assembler therefore bypasses the next seven lines. These next seven lines comprise lines 16, 17, 19, 21, 23, 25, and 27. Line 18 is considered part of line 17, line 20 is considered part of line 19, etc.

The assembler thereupon encounters the END directive, which terminates the processing of this reference line.

Whenever a reference line calls this procedure via any of the alternate names assigned to it by the directive NAME, the coding generated effects a block transfer. This block transfer will be either a transfer from or a transfer to, with or without incrementing the base address of the sending or receiving field, depending upon the call.

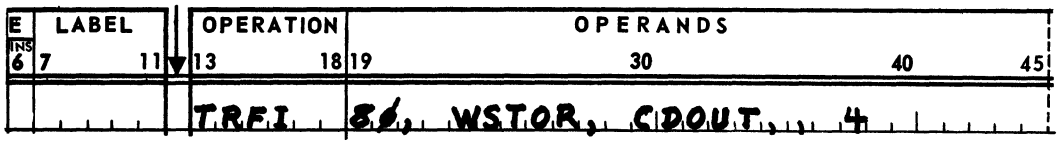
If the reference line contains TRFR, a transfer from store, resetting the base address, will be generated; TRFI calls for a transfer from, incrementing base address; TRTR calls for a transfer to, resetting the base address; and RTRI calls for a transfer to store, imcrementing the base address.

The reference line must supply five parameters:

- MOVE(1) is the number of characters;
- MOVE(2) is the address of the sending field;
- MOVE(3) is the address of the receiving field;
- MOVE(4) is the index register modifying the address of the sending field; and
- MOVE(5) is the index register modifying the address of the receiving field.

Note that the indexed labels are always written using the name of the procedure, rather than the alternate names.

The reference line



causes the following lines to be generated:

(1) FT 80, 18

(2) FT CDOUT, 16, 4

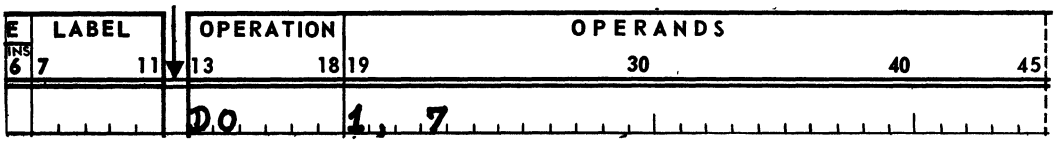
(3) TF1 WSTOR

Figure 7. Coding Generated from the Illustrated TFRI Call.

The logic of the generation is as follows:

a. Because the procedure is called by the name TRFI, the assembler is directed to equate MOVE(0) to 2. When line 6 is processed, therefore, lines 7 and 11, and everything falling within their range, are bypassed.

b. When line 15 is processed, MOVE(0) is found to be unequal to zero, thereby satisfying the condition of inequality and making this line



c. The first of the seven lines is line 16. After processing this line, line 1 of Fig. 7 is generated.

d. Since MOVE(0) is equal to 2, the condition specified by the DO on line 17 is satisfied; and line 2 of Fig. 7 is generated.

e. The condition on line 19 of the procedure is not satisfied, and the assembler proceeds to line 21.

f. MOVE(0) is equal to 2, satisfying the condition specified on line 21, resulting in the generation of line 3 of Fig. 7.

Note that MOVE(4) on this reference line is blank, indicating no index register modification of the address of the sending field. The rule governing omitted expressions applies to reference lines: if an expression other than the last one on a line is omitted, the comma ending that expression must appear on the line.

g. The conditions specified on lines 23, 25, and 27 are not satisfied, bringing the assembler to the END directive and terminating the processing of the reference line.

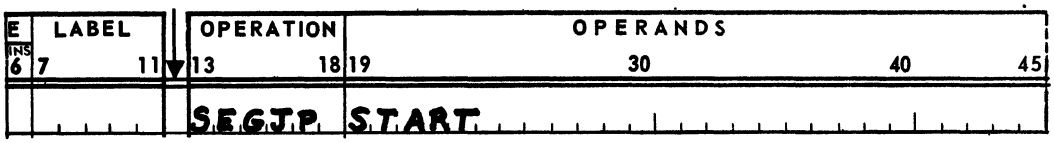
C. SEGJP

The segment jump directive enables the programmer to produce an object program divided into segments. These segments are loaded from tape and utilized one at a time.

The SEGJP assembler directive contains

- SEGJP in the operands field, and
- an integer representing an absolute address, or a label in the operands field.

When the assembler comes to the SEGJP directive



it causes a "T" block to be produced containing a jump instruction to the label (START) or address in the operands field. In this way SEGJP is similar to the END directive.

Following the "T" block, an "S" block is produced to mark the beginning of the next program segment. The "S" block is similar to the "R" block beginning the first program segment. The assembler creates the segment ID by adding a one to the program or preceding segment ID.

If the program ID (columns 7-10 of the BEGIN directive) contains a numeric value in columns 9 and 10, a decimal 01 is added. If the program ID contains an alphabetic character in column 9 and a numeric value in column 10 a decimal 1 is added.

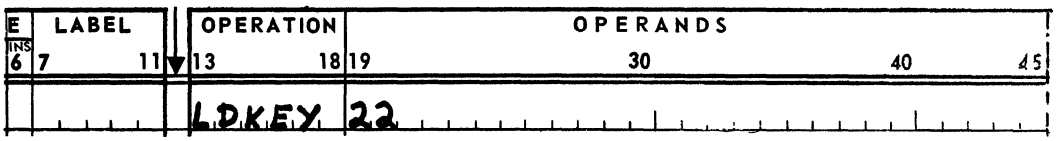
D. LDKEY

The load key directive makes it possible for the programmer to key instruction blocks containing assembled object code.

The LDKEY assembler directive contains

- LDKEY in the operation field, and
- a binary value of 0-63 in the operands field.

For example, the directive



will cause a 'LDKEY' block to be assembled. The LDKEY block is the same as an 'instruction' block except for the first character. The key obtained from the operands field of the directive (22) becomes the first character, replacing the W. This assembly continues until 112 characters of object code have been processed, an ORIG directive is used, or another LDKEY directive is encountered.

Care should be taken that the key does not duplicate any character being used by the assembler, loader, or utility routines to identify specific block types (see the description of system and library tape conventions).

ERROR CODE	DESCRIPTION
L	Duplicate or undefined label or more than 10 labels on a line.
E	Expression is too large, or has been omitted.
O	Unrecognizable OP code.
S	Card sequence error.
C	Tape block count error. Indicates block was missed during assembly.
D	More than 10 nested DO's or an incorrect expression on a DO line.
F	Incorrect form statement or form table has been exceeded. (13 characters per entry)*
M	A referenced procedure is missing from the assembler library.
P	More than 30 parameters associated with a PROC reference or the PROC name table has been exceeded. (5 characters per entry)*

* FORM and PROC use the same 360 character table in 8K storage. Therefore, the limit of the number of FORM and PROC names that can be used is interdependent.