IBM Technical Newsletter

File Number	1410/7010-22
Re: Form No.	C28-0326-2
This Newsletter No.	N27-1267
Date	December 30, 1966
Previous Newsletter Nos.	N27-1223

IBM 1410/7010 AUTOCODER

This Technical Newsletter amends the publication IBM 1410/7010 Operating System; Autocoder, Form C28-0326-2, to include new information concerning the NOTE and MACRO statements, and to make other necessary changes and additions.

The attached replacement pages (9-10, 41-44, 47-48) should be substituted for the corresponding pages now in the publication. Text changes are indicated by a vertical line to the left of the affected text; figure changes are indicated by a bullet (•) to the left of the affected figure caption.

Please file this cover letter at the back of the publication. It provides a method of determining if all changes have been received and incorporated into the publication.

IBM Corporation, Programming Publications, Dept. 637, Neighborhood Road, Kingston, N.Y. 12401

PRINTED IN U.S.A.

as the reference address of COMMON during the assembly process. All relocatable addresses of data in COMMON are relative to 99999. For example, the 15th location downward in COMMON is assigned the value 99985, and appears as the same relative address in all subprograms. Labels referencing COMMON are assigned downward relocation indicators for absolute adjustment by the Linkage Loader.

Absolute adjustment involves changing the relative values of the labels (assigned to them by the processor) to absolute values in the relocated COMMON data area. The adjustment factor applied is the difference between the value of COMMON in the assembly process (99999) and the absolute value of COMMON determined by the Linkage Loader. Normally, the Linkage Loader will place COMMON at the location represented by the value of the system symbol /AMS/ (Absolute Memory Size). However, the programmer can specify a different absolute location for COMMON by means of a BASE2 statement. (The interested reader will find a fuller discussion of this subject in the publication, *System Monitor.*)

The steps necessary to use COMMON in a subprogram are discussed under "DAV – Define Area in COMMON," in the subsection "Declarative Operation Codes."

Processing Options

There are four processing options which can be exercised by the user:

1. He can suppress the printing of the assembly listing (on the Standard Print Unit).

2. He can suppress the punching of the object deck (on the Standard Punch Unit).

3. If there are no macro statements in the source deck, he can speed up the assembly process by indicating this fact.

4. He can suppress the diagnostic generation of an "M" flag for uses of index registers 14 and 15 when there is no true multiple definition. (See NOTE 1, under "Indexing with Address Adjustment.")

These options are indicated by means of additional parameters in the EXEQ card that calls the Autocoder processor.

The four parameters are:

NOPRT – Suppress printing
NOPCH – Suppress punching
NOMAC – No macros present
NOFLG – Suppress "M" flag

Any or all of these parameters may be used in the EXEQ card. They can appear in any order immediately following the EXEQ parameters required by the System Monitor. (See the publication, *System Monitor,* for details concerning the EXEQ card.)

Specification of parameters in the EXEQ card is concluded by the first blank encountered in the operand field. The following examples illustrate the format:

LABEL	OPERATION CODE	OPERAND
MON$$	EXEQ	AUTOCODER, SOF, SIU, NOPRT
MON$$	EXEQ	AUTOCODER, , , NOMAC, NOPCH, NOFLG
MON$$	EXEQ	AUTOCODER, , , NOFLG, NOPRT, NOMAC, NOPCH

Autocoder Multiple Compilation

Autocoder can compile any number of programs with a single MON$$ EXEQ AUTOCODER card. The output is the same as if it were produced by several separate compilations.

Input for a multiple compilation consists of the MON$$ EXEQ AUTOCODER card followed by the source decks to be compiled. No control cards are necessary between the END statement of one program and the first card of the next program if the programmer wants the subsequent compilation to receive standard treatment; that is, printing, punching, and normal macro and flag processing.

A different set of processing options (NOPRT, NOPCH, NOMAC or NOFLG) can be specified for an ensuing program in a multiple compilation by placing an Option card after the preceding END statement. This card has the same requirements and options as the MON$$ EXEQ AUTOCODER card except that the label and operation fields, card columns 6-20, must contain blanks (instead of MON$$ EXEQ). The processing options specified in this Option card will be applied until the next Autocoder END card is read by the processor.

Autocoder multiple compilation has two potential advantages:

1. It enables the programmer to process a series of source decks from the Alternate Input Unit as well as the Standard Input Unit.

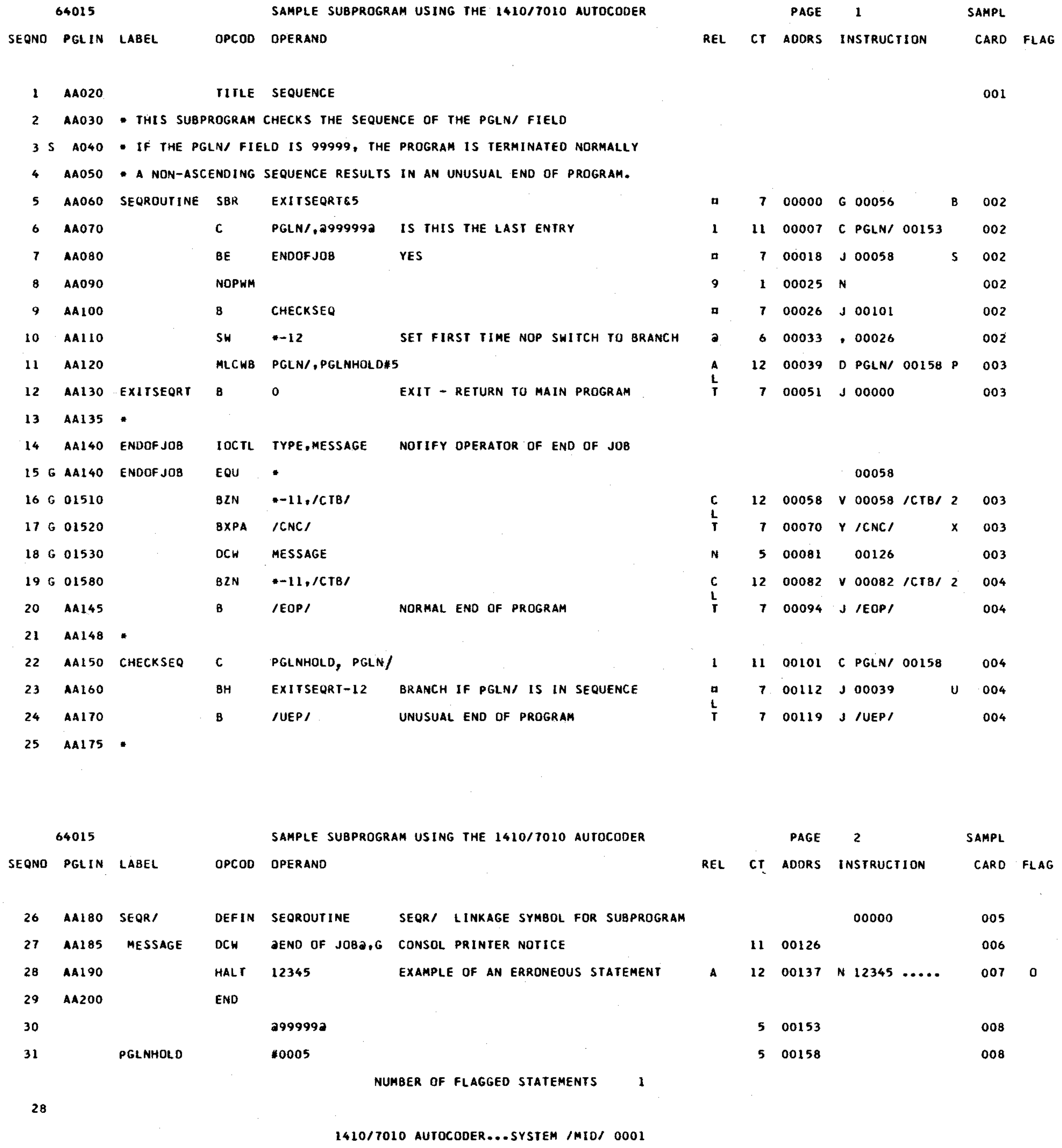
2. It bypasses the monitor processing which normally is necessary between compilations.

Terminating the Object Program

The object progam must terminate execution by means of one of the following instructions:

B	/EOP/	Normal End of Program
B	/UEP/	Unusual End of Program

Both forms of termination are shown in Figure 2. Full details can be found in the publication, *System Monitor.*



```
     64015                      SAMPLE SUBPROGRAM USING THE 1410/7010 AUTOCODER              PAGE    1                  SAMPL
SEQNO PGLIN  LABEL       OPCOD  OPERAND                                              REL  CT ADDRS  INSTRUCTION          CARD FLAG

  1   AA020              TITLE  SEQUENCE                                                                                 001
  2   AA030  * THIS SUBPROGRAM CHECKS THE SEQUENCE OF THE PGLN/ FIELD
  3 S  A040  * IF THE PGLN/ FIELD IS 99999, THE PROGRAM IS TERMINATED NORMALLY
  4   AA050  * A NON-ASCENDING SEQUENCE RESULTS IN AN UNUSUAL END OF PROGRAM.
  5   AA060  SEQROUTINE  SBR    EXITSEQRT&5                                          ¤     7 00000  G 00056         B    002
  6   AA070              C      PGLN/,@99999@   IS THIS THE LAST ENTRY               1    11 00007  C PGLN/ 00153        002
  7   AA080              BE     ENDOFJOB        YES                                  ¤     7 00018  J 00058         S    002
  8   AA090              NOPWM                                                       9     1 00025  N                    002
  9   AA100              B      CHECKSEQ                                             ¤     7 00026  J 00101              002
 10   AA110              SW     *-12            SET FIRST TIME NOP SWITCH TO BRANCH  @     6 00033  , 00026              002
 11   AA120              MLCWB  PGLN/,PGLNHOLD#5                                     A    12 00039  D PGLN/ 00158   P    003
                                                                                     L
 12   AA130  EXITSEQRT   B      0               EXIT - RETURN TO MAIN PROGRAM        T     7 00051  J 00000              003
 13   AA135  *
 14   AA140  ENDOFJOB    IOCTL  TYPE,MESSAGE    NOTIFY OPERATOR OF END OF JOB
 15 G AA140  ENDOFJOB    EQU    *                                                                     00058
 16 G 01510              BZN    *-11,/CTB/                                           C    12 00058  V 00058 /CTB/   2    003
                                                                                     L
 17 G 01520              BXPA   /CNC/                                                T     7 00070  Y /CNC/         X    003
 18 G 01530              DCW    MESSAGE                                              N     5 00081    00126              003
 19 G 01580              BZN    *-11,/CTB/                                           C    12 00082  V 00082 /CTB/   2    004
                                                                                     L
 20   AA145              B      /EOP/           NORMAL END OF PROGRAM                T     7 00094  J /EOP/              004
 21   AA148  *
 22   AA150  CHECKSEQ    C      PGLNHOLD, PGLN/                                      1    11 00101  C PGLN/ 00158        004
 23   AA160              BH     EXITSEQRT-12    BRANCH IF PGLN/ IS IN SEQUENCE       ¤     7 00112  J 00039         U    004
                                                                                     L
 24   AA170              B      /UEP/           UNUSUAL END OF PROGRAM               T     7 00119  J /UEP/              004
 25   AA175  *


     64015                      SAMPLE SUBPROGRAM USING THE 1410/7010 AUTOCODER              PAGE    2                  SAMPL
SEQNO PGLIN  LABEL       OPCOD  OPERAND                                              REL  CT ADDRS  INSTRUCTION          CARD FLAG

 26   AA180  SEQR/       DEFIN  SEQROUTINE      SEQR/  LINKAGE SYMBOL FOR SUBPROGRAM                  00000              005
 27   AA185   MESSAGE    DCW    @END OF JOB@,G  CONSOL PRINTER NOTICE                     11 00126                       006
 28   AA190              HALT   12345           EXAMPLE OF AN ERRONEOUS STATEMENT    A    12 00137  N 12345 .....        007   0
 29   AA200              END
 30                             @99999@                                                    5 00153                       008
 31          PGLNHOLD           #0005                                                      5 00158                       008
                                                        NUMBER OF FLAGGED STATEMENTS     1

 28

                                                    1410/7010 AUTOCODER...SYSTEM /MID/ 0001
```

●Figure 2. A Page from an Assembly Listing

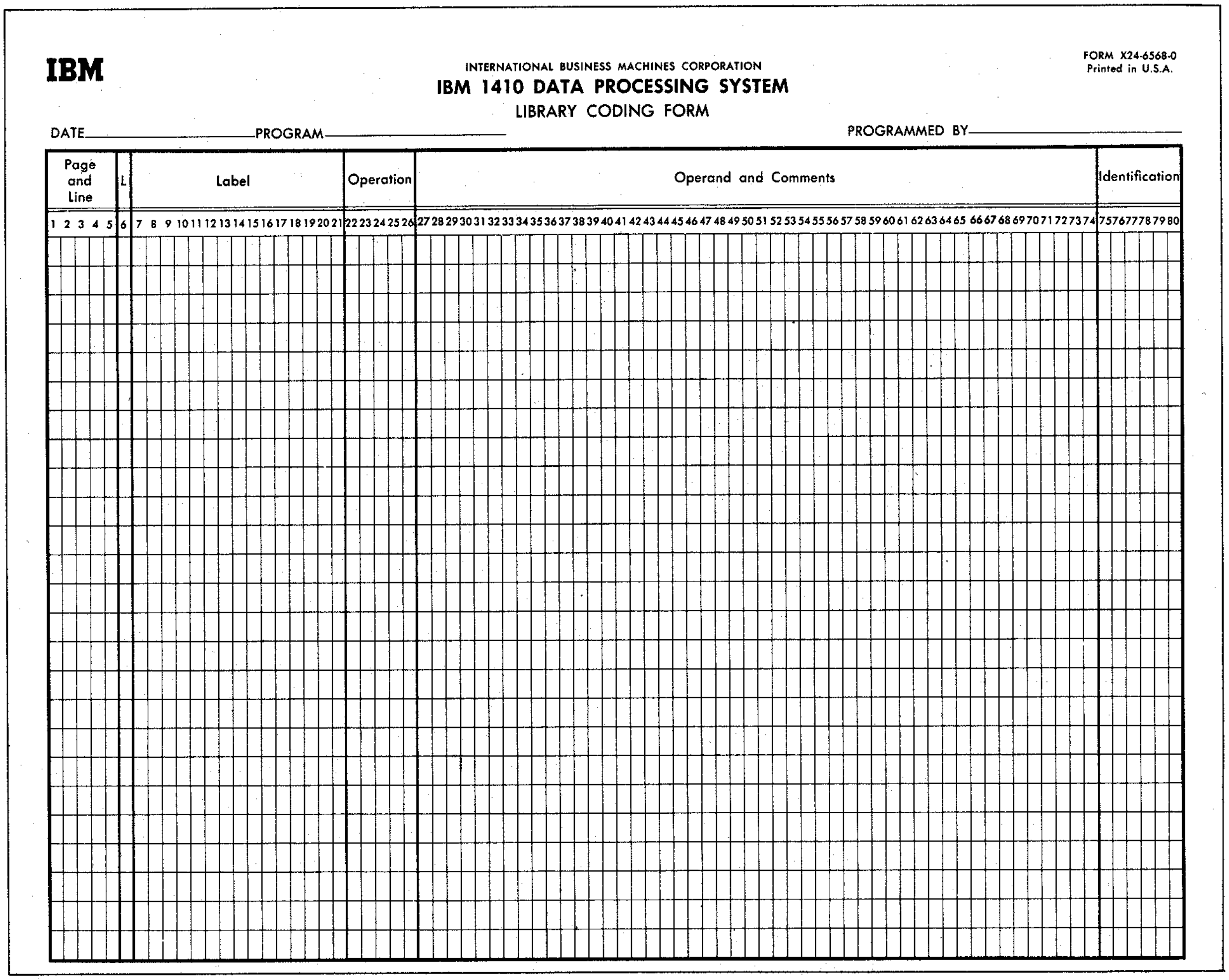
Assembly Listing

Each page of the assembly listing contains a page heading line and a column heading line.

The page heading line contains the following information, from left to right:

1. The date contained at location /DAT/ (the system symbol for the five-position date field in the Resident Monitor)
2. Information supplied via HEADR card
3. Page number in the listing
4. The identification supplied by HEADR or RESEQ cards

The column heading line is illustrated in Figure 2, which shows the assembly listing of a subprogram assembled by the 1410/7010 Autocoder processor. The subprogram contains a deliberate error contrived to exhibit Autocoder's diagnostic flagging system. Figure 2 illustrates the following items, going from left to right in the column heading line:



IBM

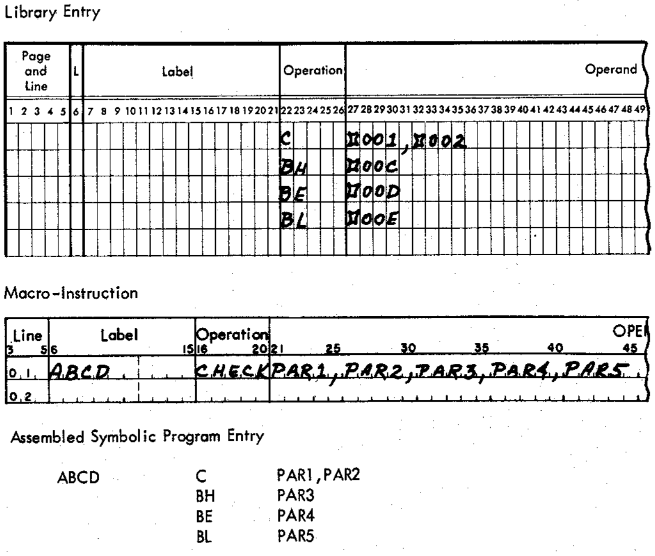
INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1410 DATA PROCESSING SYSTEM
LIBRARY CODING FORM

FORM X24-6568-0
Printed in U.S.A.

DATE________ PROGRAM________ PROGRAMMED BY________

Page and Line	L	Label	Operation	Operand and Comments	Identification
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	75 76 77 78 79 80

Figure 78. IBM 1410/7010 Library Coding Form



Library Entry

Page and Line	L	Label	Operation	Operand
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
			C	□001,□002
			BH	□00C
			BE	□00D
			BL	□00E

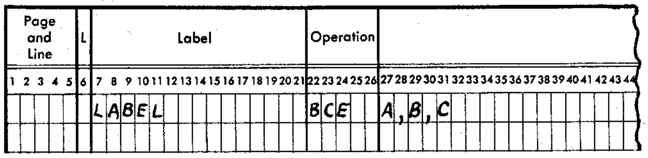
Macro-Instruction

Line 3 5	Label 6 15	Operation 16 20	OPE 21 25 30 35 40 45
0 1	ABCD	CHECK	PAR1,PAR2,PAR3,PAR4,PAR5
0 2			

Assembled Symbolic Program Entry

```
ABCD        C       PAR1,PAR2
            BH      PAR3
            BE      PAR4
            BL      PAR5
```

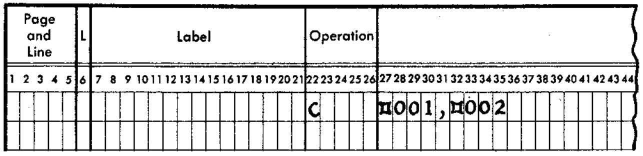
Figure 79. Macro Operations



Page and Line	L	Label	Operation	
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
		LABEL	BCE	A,B,C

Figure 80. Model Statement for a Complete Instruction

that a corresponding parameter from the macro-instruction operand field must be inserted in its place. This code is a □ followed by a number from 001 to 199, that indicates the position of the parameter in the macro-instruction. The macro-instruction in the source program will give the parameter entries to be inserted in the object routine. The model statement is illustrated in Figure 81.

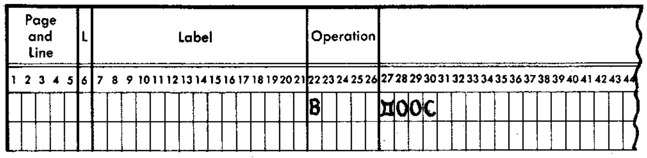


Page and Line (1–5)	L (6)	Label (7–21)	Operation (22–26)	(27–44)
			C	□001,□002

Figure 81. Model Statement for an Incomplete Instruction with Required Parameters

c. If the entry is incomplete, the programmer writes a □ followed by a number from 001 to 199 with AB bits over the units position (parameter 001 is □00A, parameter 2 is □00B, etc.). This indicates that the entry is to be included in the object routine only if the parameter is specified by the macro-instruction. For example, if parameter 003 does not appear in the macro-instruction, the instruction shown in Figure 82 will be deleted from the object routine.

Note: If parentheses are used, the programmer cannot use zoned switches in a MATH or BOOL statement.



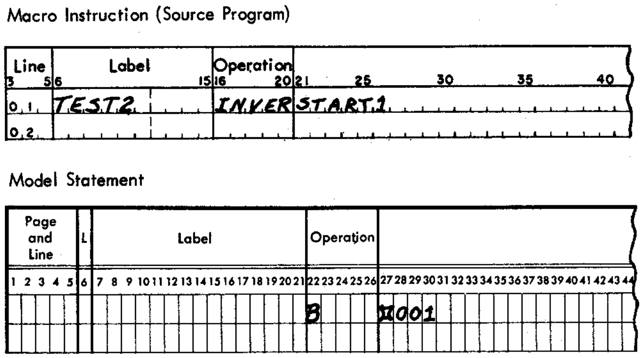
Page and Line (1–5)	L (6)	Label (7–21)	Operation (22–26)	(27–44)
			B	□00C

Figure 82. Model Statement for an Incomplete Instruction with Conditional Parameters

Labeling: If the model statement represents an instruction entry point for a branch instruction elsewhere in the program, it should have a label.

If additional external labels are required and specified as parameters in the macro-instruction they can be inserted in the label field of the symbolic program entry by using the □001-199 code.

The label of the macro-instruction causes the generation of an equate statement in the assembled object routine. The label is equated to an *, as shown in Figure 83.



Macro Instruction (Source Program)

Line (3–5)	Label (6–15)	Operation (16–20)	(21–40)
0.1	TEST2	INVER	START1
0.2			

Model Statement

Page and Line (1–5)	L (6)	Label (7–21)	Operation (22–26)	(27–44)
			B	□001

Assembled Symbolic Program Entry

```
TEST2          EQU     *
               B       START1
```

Figure 83. Labeling

Another example is shown in Figure 84.

Symbolic Addressing within the Library Routine: To allow a symbolic reference to other instructions in a library routine a □followed by a number from 001 to 199 with a B bit over the units position (□00J = symbolic address 1, □00K = symbolic address 2, etc.) can be used. For example, the processor generates the symbolic address if the code □00J is used as a label for one entry and as an operand of at least one other entry in the same library routine.

Internal labels within flexible routines are generated in the form □nnnmmm, where nnn is the code (00J-09R), and mmm is the number of the macro within the source program. This is done to avoid duplicate address assignments for labels.

Example: Use the generated symbolic address of □00J as an operand for entry 3 and as the label for entry 6. UPDAT is the 23d macro encountered in the source program (Figure 85).

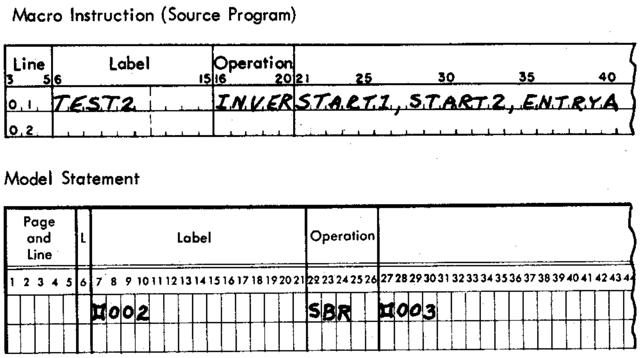
Address Adjustment and Indexing: The parameters in a macro-instruction and the operands in partially complete instructions in a library routine can have address adjustment and indexing.

If address adjustment is used in both the parameter and the instruction, the assembled instruction will be adjusted to the algebraic sum of the two. For example, if the address adjustment on one is +7 and the other is −4, the assembled instruction will have address adjustment equal to +3.

Model statement operands can be indexed. This indexing takes precedence over any indexing of a parameter supplied by a macro-instruction. The model statement index is used.

Literals: Operands of instructions in library routines may use literals as required. However, these literals may not contain the @ symbol within an alphameric literal.

Note: Area defining literals and area defining constants cannot be used in a MACRO statement.



Macro Instruction (Source Program)

Line (3–5)	Label (6–15)	Operation (16–20)	(21–40)
0.1	TEST2	INVER	START1,START2,ENTRYA
0.2			

Model Statement

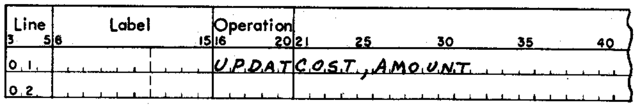
Page and Line (1–5)	L (6)	Label (7–21)	Operation (22–26)	(27–44)
		□002	SBR	□003

Assembled Symbolic Program Entry

```
TEST2          EQU     *
START2         SBR     ENTRYA
```

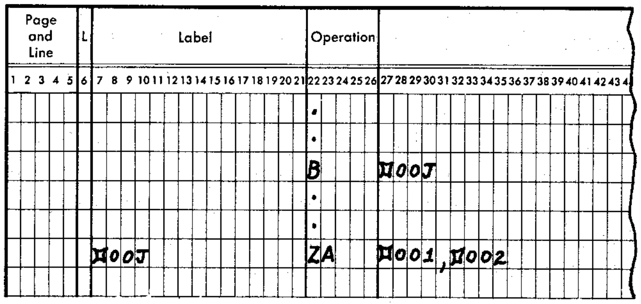
Figure 84. Additional External Labels

Macro Instruction (Source Program)



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		UPDAT	COST,AMOUNT
0 2			

Model Statement



Page and Line 1 2 3 4 5	L 6	Label 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	Operation 22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 4
			.	
			.	
			B	□00J
			.	
			.	
		□00J	ZA	□001,□002

Assembled Symbolic Program Entry

```
                    .
                    .
                    B         □00J023
                    .
                    .
     □00J023        ZA        COST,AMOUNT
```

Figure 85. Internal Labels

NOTE 1: A model statement in the library routine for a macro-instruction may not be another macro-instruction.

NOTE 2: END statements cannot be used in library routines.

NOTE 3: A comments card can be included in the model statements. It must be written with an asterisk in column 7.

The Processor enters model statements in the library tape immediately following the header statement during System Generation.

Result: Any library routine can be extracted by writing the associated macro-instruction in the source program.

Figure 86 is a summary of the codes that can be used in the model statements of library routines.

CODE	POSITION	FUNCTION
□001 - □199	Statement	Substitute parameter (parameter must be present)
□00A - □191	Statement	Substitute parameter (if parameter is missing, delete statement)
□00J - □19R	Label Field and Operand Field	Assign internal label

Figure 86. Model Statement Codes

General Description: A macro-instruction is the entry in the source program that causes a series of instructions to be inserted in a program.

The Programmer:

1. Writes the name of the library routine in the operation field.

2. Writes the label that is to reference the first assembled model statement. A LABEL EQU * is generated to do this.

3. Writes the parameters that are required for the particular object routine desired. These parameters, used by the model statements, are written as follows:

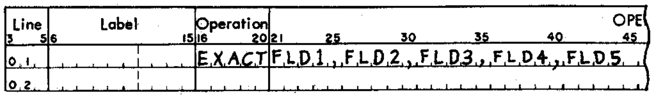
   a. Parameters must be written in the sequence in which they are to be used by the codes in the model statements. For example, if cost is parameter 001, it must be written first so that it will be substituted wherever a □001 or □00A appears as a label, operation code, or operand of a model statement.

   b. As many parameters may be used as can be contained in the operand fields of five or fewer coding sheet lines. If more than one line is needed for a macro-instruction, the label and operation fields of the additional lines must be left blank. Parameters must be separated by a comma. They cannot contain blanks or commas unless they appear between @ symbols. The @ symbol itself cannot appear between @ symbols. Also, the @ symbol can be used only in pairs as a literal identifier. It cannot be used in any other way; e.g., a single @ symbol could not be used to represent the d modifier of a macro-instruction. If parameters for a single macro-instruction require more than one coding sheet line, the last parameter in each line must be followed immediately by a comma. The last parameter in a macro-instruction should not be followed by a comma.

   c. Parameters that are not required for the particular object routine desired can be omitted from the operand field of the macro-instruction. However, if a parameter is omitted, the comma that would have followed the parameter must be included, unless the omitted parameter is behind the last parameter which is included in the macro-instruction. These commas are necessary to count parameters up to the last included parameter. All parameters between the last included parameter and parameter 199 are assumed by the processor to be absent.

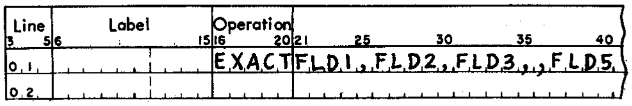
Figures 87, 88, 89 and 90 show how parameters can be omitted. The hypothetical macro-instruction called EXACT is used. EXACT can have as many as nine parameters.

The Processor extracts the library routine and selects the model statements required for the object routine as specified by the parameters in the macro-instructions, and by substitution and switches set by BOOL or COMP in the model statements.



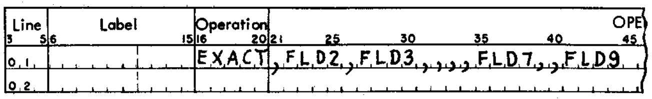
Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPE 45
0 1		EXACT	FLD1,FLD2,FLD3,FLD4,FLD5
0 2			

Figure 87. Parameter for EXACT. 006-199 Missing



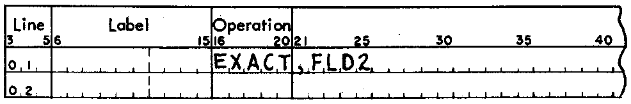
Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		EXACT	FLD1,FLD2,FLD3,,FLD5
0 2			

Figure 88. Parameters 004 and 006-199 Missing



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPE 45
0 1		EXACT	,FLD2,FLD3,,,,FLD7,,FLD9
0 2			

Figure 89. Parameters 001, 004-006, 008 and 010-199 Missing



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		EXACT	,FLD2
0 2			

Figure 90. Parameters 001 and 003-199 Missing

Result: The resulting program entries are merged with the source program entries behind the macro-instruction.

Pseudo-Macro Instructions

These statements never appear in a user's source program or in the output listing of an assembled Autocoder program. However, they are used in library routines to signal the processor that certain conditions exist which can affect the assembly of an object routine. For example, the presence of a pseudo-macro-instruction in a library routine can cause a group of model statements to be deleted. Thus, pseudo-macros provide the writer of library routines with a coding flexibility which exceeds the limitations of the substitution and condition codes described previously.

Pseudo-macro-instructions may be written anywhere in a library routine. The five pseudo-macros incorporated in the Autocoder processor are MATH, BOOL, COMP, NOTE, and MEND.

Permanent and Temporary Switches

The MATH, BOOL, and COMP pseudo-macros use internal indicators (switches) to signal the processor of existing status conditions.

There are 099 permanent and 199 temporary switches available for recording status conditions. Each switch occupies one core-storage position during the macro generator phase of Autocoder. If a storage position contains the character A (AB 1 bits), the switch is ON; if it contains a ? (CAB 82 bits), the switch is OFF. At the beginning of assembly all switches are OFF.

Permanent Switches: Permanent switches retain status conditions during the entire macro generator phase unless changed by a pseudo-macro. They are addressed by using a # symbol followed by the three-digit number of the switch to be set or tested. For example, #001 addresses permanent switch 001; #002 addresses switch 002; and #099 addresses switch 099.

Temporary Switches: When the processor encounters a macro-instruction, the temporary switches are set to the condition (presence or absence) of the parameters in the operand of the macro field. If the parameter is present, the corresponding switch is set ON. If the parameter is missing, the switch is set OFF. For example, if parameter 001 is present, temporary switch 001 is turned ON. If parameter 002 is missing from the macro-instruction, temporary switch 002 is OFF. Temporary switches retain status throughout the processing of a macro-instruction unless changed by a pseudo-macro. After the macro-instruction has been completely processed, all temporary switches are set OFF. Temporary switches are addressed by using a ⌑ symbol followed by the three-digit number of the switch to be set or tested. For example, ⌑001 addresses temporary switch 001; ⌑002 addresses switch 002; and ⌑199 addresses switch 199.

If a macro with a maximum of nine parameters is encountered, the processor sets the first nine temporary switches to indicate the presence or absence of these nine parameters. Temporary switches 010-199, which are OFF, can be used by the pseudo-macros to communicate conditions to the processor while it is working on this particular macro-instruction. This use of temporary switches is recommended because it reserves the permanent switches for communicating information from one macro to another.

MATH — For Solving Algebraic Expressions

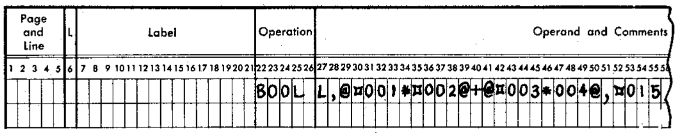
A MATH pseudo-macro contains as operands: sum boxes, arithmetic expressions, and sign switches.

Sum Boxes: A sum box is a group of five core-storage positions used to store the result of an arithmetic expression. Autocoder makes available 20 such sum boxes. A sum box is addressed by using a # symbol followed by the three-digit number (ending in zero or five) of the sum box to be referenced. For example, the address of the first sum box is #005; the address of the second sum box is #010; and the address of the twentieth sum box is #000.

At the beginning of the macro phase, a sum box contains 00000. Any number may be placed in a sum is OFF, the statement is false. Therefore, set temporary switch 015 OFF and skip to statement labeled L.

The example shown in Figure 97 states:

1. If both temporary switches 001 and 002 or both temporary switches 003 and 004 are ON, the statement is true. Therefore, set temporary switch 015 ON.
2. However, if either temporary switch 001 or 002 and either temporary switch 003 or 004 is OFF, the statement is false. Therefore, set temporary switch 015 OFF and skip to the model statement whose label is L.



Page and Line	L	Label	Operation	Operand and Comments
			BOOL	L,@¤001*¤002@+@¤003*004@,¤015

Figure 97. BOOL Pseudo-Macro

Figure 98 is a table showing all conditions that will cause the BOOL statement shown in Figure 97 to be true.

Figure 99 is a table showing all conditions that will cause the BOOL statement shown in Figure 97 to be false.

COMP — To Compare Two Fields

General Description: The COMP pseudo-macro compares an A-field to a B-field (maximum of 15 characters), and sets permanent or temporary switches to indicate the result of the comparison.

The Programmer:

1. Writes the name of the pseudo-macro (COMP) in the operation field.

SWITCHES

CONDITIONS

001	*	002	+	003	*	004		LOGICAL VALUE
ON 1	*	ON 1	+	OFF 0	*	OFF 0	=	1
OFF 0	*	OFF 0	+	ON 1	*	ON 1	=	1
ON 1	*	ON 1	+	ON 1	*	ON 1	=	1
ON 1	*	ON 1	+	ON 1	*	OFF 0	=	1
OFF 0	*	ON 1	+	ON 1	*	ON 1	=	1
ON 1	*	ON 1	+	OFF 0	*	ON 1	=	1
ON 1	*	OFF 0	+	ON 1	*	ON 1	=	1

TRUE

Figure 98. True Conditions

SWITCHES

CONDITIONS

001	*	002	+	003	*	004	=	LOGICAL VALUE
OFF 0	*	OFF 0	+	OFF 0	*	OFF 0	=	0
ON 1	*	OFF 0	+	OFF 0	*	OFF 0	=	0
OFF 0	*	ON 1	+	OFF 0	*	OFF 0	=	0
OFF 0	*	OFF 0	+	ON 1	*	OFF 0	=	0
OFF 0	*	OFF 0	+	OFF 0	*	ON 1	=	0
OFF 0	*	ON 1	+	OFF 0	*	ON 1	=	0
ON 1	*	OFF 0	+	ON 1	*	OFF 0	=	0
OFF 0	*	ON 1	+	ON 1	*	OFF 0	=	0
ON 1	*	OFF 0	+	OFF 0	*	ON 1	=	0

FALSE

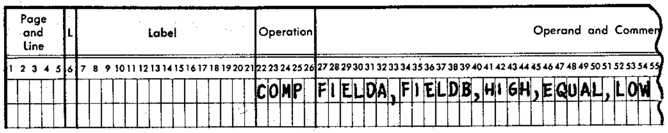
Figure 99. False Conditions

2. Writes the operand field in the format shown in Figure 100. The first and second entries are the A- and B-fields. The A- and B-fields may be any of the parameters 001-199, sum boxes #005-#000, or literals. They cannot be switches.

NOTE 1: For the COMP pseudo-macro, alphameric literals are not enclosed by @ symbols. Entries 3, 4, and 5 are high, equal, and low switches.

NOTE 2: The codes for the two fields to be compared must be present in all COMP pseudo-macro-instructions. Codes for the switches may be omitted if they are not needed to store the result of the compare operation. However, if a switch is omitted, the comma that would have followed it must be included in the operand field.

NOTE 3: B-field controls compare. (High-order position of B-field ends compare.)



Page and Line	L	Label	Operation	Operand and Comments
			COMP	FIELDA,FIELDB,HIGH,EQUAL,LOW

Figure 100. Format for COMP Pseudo-Macro

The Processor:

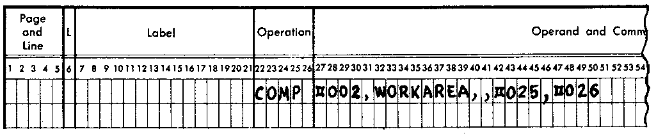
1. Compares the A-field to the B-field.
2. Sets one status switch ON and two switches OFF to reflect the result of the comparison.
   a. The first switch is set ON, if the value of the B-field is greater than that of the A-field.

b. The second switch is set ON, if the B-field is equal to the A-field.

c. The third switch is set ON, if the value of the B-field is less than that of the A-field.

Examples: Figure 101 shows a COMP pseudo-macro which states:

1. Compare parameter 002 of the macro statement to WORKAREA.
2. If parameter 002 is equal to WORKAREA, turn on temporary switch 25.
3. If WORKAREA is less than parameter 002, turn on temporary switch 26.

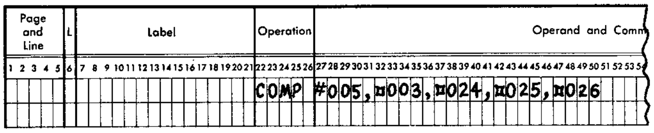

Page and Line
L
Label
Operation
Operand and Comm

```
                     COMP ¤002,WORKAREA,,¤025,¤026
```

Figure 101. COMP Pseudo-Macro

Figure 102 shows a COMP pseudo-macro which states:

1. Compare the contents of sum box 005 to parameter 003 of the macro statement.
2. If the result is HIGH, set temporary switch 024 ON.
3. If the result is EQUAL, set temporary switch 025 ON.
4. If the result is LOW, set temporary switch 026 ON.


Page and Line
L
Label
Operation
Operand and Comm

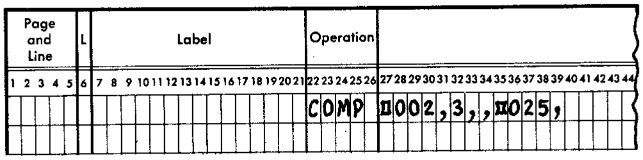
```
                     COMP #005,¤003,¤024,¤025,¤026
```

Figure 102. Comparing a Parameter to the Contents of a Sum Box

NOTE: Standard 1410/7010 collating sequence determines HIGH, EQUAL, or LOW conditions. Comparisons are controlled by the B-field. Thus, the statement shown in Figure 103 will cause temporary switch 025 to be set ON if the low-order position of parameter 002 is a 3.

NOTE—To Produce a Message

General Description: The NOTE pseudo-macro is used to write messages concerning conditions that can arise during the processing of a macro-instruction.


Page and Line
L
Label
Operation

```
                     COMP ¤002,3,,¤025,
```

Figure 103. Checking for a Single Character

The message is printed in line on the output device (tape or on-line printer). In addition, an "N" will be automatically inserted in the flag field of the assembly listing.

The Programmer:

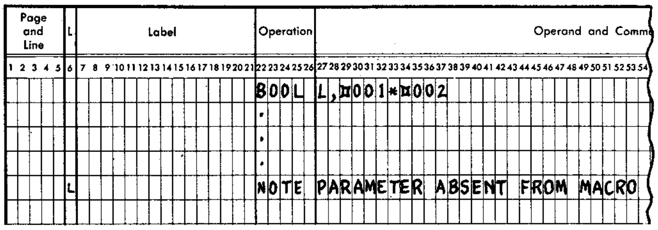
1. Writes the name of the pseudo-macro (NOTE) in the operation field.
2. Writes the message in the operand field.

NOTE: Two successive blanks terminate the operand of a NOTE statement.

The Processor: Prints the message on the Standard Print Unit (tape or on-line printer).

Examples: Figure 104 shows how the NOTE pseudo-macro can be used in combination with the BOOL pseudo-macro. The BOOL pseudo-macro tests to ensure that parameters 001 and 002 are present in the macro-instruction. If either parameter is missing, the processor skips to the NOTE pseudo-macro and prints:

PARAMETER ABSENT FROM MACRO.


Page and Line
L
Label
Operation
Operand and Comm

```
                     BOOL L,¤001*¤002
                     .
                     .
                     .
     L               NOTE PARAMETER ABSENT FROM MACRO
```

Figure 104. NOTE Pseudo-Macro

MEND — End of Routine

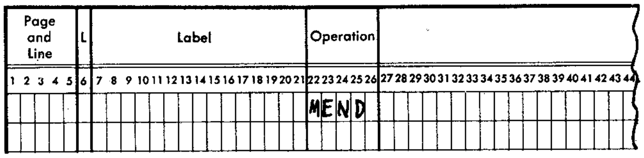
General Description: This pseudo-macro signals the end of generation for a macro-instruction. It may appear anywhere in a library routine.

The Programmer:

1. Writes the name of the pseudo-macro (MEND) in the operation field.
2. Leaves the operand field blank.

The Processor: Stops processing the macro-instruction when it encounters a MEND statement. Figure 105 shows a MEND pseudo-macro.

NOTE: A BOOL pseudo-macro can be used to skip over a MEND pseudo-macro which appears within the library routine if conditions indicate that more model statements must be processed.


Page and Line
L
Label
Operation

```
                     MEND
```

Figure 105. MEND Pseudo-Macro