IBM Technical Newsletter

File Number	1410/7010-22
Re: Form No.	C28-0326-2
This Newsletter No.	N27-1223
Date	June 24, 1965
Previous Newsletter Nos.	None

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 DCW statement, the chaining of instructions and the use of zoned switches, and to correct minor errors.

The attached replacement pages (11-14, 29-30, and 41-42) should be substituted for the pages currently in the publication. Text changes are indicated by a vertical line at the left of the affected text.

In addition, the following changes should be made to the publication:

Page	Amendment
22	Change the Meaning of the mnemonic "HEADR" (under "Assembly Control Codes") to read Header Line
26	Under the heading "Assignment of Data Areas in COMMON", the first paragraph refers to "Figures 61 and 62". Change this reference to read: Figures 35 and 36
27	In the section "Use of Labels Referencing COMMON," there are two references to "Figure 61". Change these references to read: Figure 35
47	The first complete sentence on the page ends with "skip to statement 004". Change this to read: skip to the statment labeled L.
51	The second sentence in Note 2 ends with the word "common". The word "common" should be capitalized to read: COMMON

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

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

PRINTED IN U.S.A.

1. SEQNO – Sequence Number: The sequence number of statements as they appear in the assembly listing.

2. PGLIN – Page and Line Number: The page and line number as it appears in columns 1 through 5 of the cards in the source deck. Page and line numbers must consist of five non-blank characters and must appear in ascending sequence.

Statements generated by the macro generator will have a page and line number in this field supplied by the generator. These numbers have no relationship to the numbers of the hand-coded statements; they represent the order in which the statements appear in the Macro Library.

The space between the SEQNO and PGLIN columns of the listing are used by the processor to contain either an "S" or a "G," under the following conditions.

S – The page and line number of the statement is not in ascending sequence in relation to the preceding source statement. This is only a warning to the programmer that his source statements may be out of sequence.

G – This character differentiates statements produced by the macro generator from the hand-coded source statements.

3. LABEL – Label: The contents of the label field, columns 6 through 15, of the Autocoder statement.

4. OPCOD: The Operation Code, columns 16 through 20, of the Autocoder statement.

5. OPERAND: The contents of the operand field, columns 21 through 72, of the Autocoder statement.

6. REL – Relocation Indicator: This is a code character that indicates to the Linkage Loader the type of relocation to be applied to the element(s) in the statement.

7. CT – Character Count: The length in characters of the assembled imperative statement, or the number of core-storage locations reserved for a constant defined in a declarative statement.

8. ADDRS: The relative address assigned by the processor to the instruction or constant. This address is subject to relocation.

9. INSTRUCTION: The assembled machine-language instruction or constants from which the object deck is constructed.

10. CARD – Card Number: The sequence number of the card in which the associated constants or instructions appear in the object deck. This sequence number is automatically computed and placed in columns 73-75 of each card in the object deck, in ascending order.

11. FLAG: An alphabetic character indicating an actual or possible programming error. As many as five flags can be assigned to one Autocoder statement. The flags provided are as follows:

F – invalid statement Format

M – Multiple definition of a label

N – macro generation Note

O – invalid Operation code

R – Restricted operation code (if not generated by a macro)

U – Unidentified label in the operand

W – Warning, general classification of error

Details concerning the above flags can be found in Appendix A. The total number of flagged statements is indicated at the end of the assembly listing, followed by a line which contains the sequence number of each flagged statement, to a maximum of 20 numbers. The presence of any flag except "R" causes the processor to set the "no-go" switch during assembly. This setting of the "no-go" switch can cause a bypassing of all the source cards up to the next job. See the *System Monitor* publication.

The assembly listing can be supplemented by a cross reference listing at the option of the user, by means of the PST statement. This listing analyzes the subprogram(s) just assembled, and lists each label, followed by the sequence number of the statement in which it was defined, and the sequence number of each statement in which the label is used as a reference address. See "PST – Print Symbol Table," in the subsection "Control Operation Codes," for a more detailed explanation.

NOTE: The system symbol /LIN/ controls the line count on the listing page. However, if this system symbol calls for the printing of less than 30 lines per page the processor will reject this direction and print the assembly listing at the normal 55 lines per page. See the *System Monitor* publication for details concerning this system symbol.

Replacement Codes

The Autocoder processor utilizes a second line (normally blank) in the assembly listing, for the representation of non-printable characters. Each of these characters is represented by two characters, one printed above the other, at the appropriate place in the listing. These two-character substitutions are called replacement codes, and they appear most frequently as relocation indicators or operation modifiers.

The two-character replacement codes with their conventional graphic representations, card codes, and names are listed in Figure 3.

Replacement Code	Graphic	Card Code	Name
$\dot{0}$	?	12-0	Plus Zero
$\bar{0}$	!	11-0	Minus Zero
GM	‡	12-7-8	Group Mark
QT	+++	0-7-8	Segment Mark
WS	⌒	0-5-8	Word Separator
DL	Δ	11-7-8	Delta
CT	¢ or ♭	2-8	Cent Sign or Substitute Blank
LP	[	12-5-8	Left Bracket
RP	]	11-5-8	Right Bracket
TM	√‾	7-8	Tape Mark
LT	<	12-6-8	Less Than
GT	>	6-8	Greater Than
;	;	11-6-8	Semicolon
:	:	5-8	Colon
"b	\	0-6-8	Backslash

Figure 3. Replacement Codes

Coding Sheet

The Autocoder Coding Sheet (Figure 4) provides a convenient form for coding source program statements. Column numbers on the coding sheet have a one-for-one correspondence to the columns on the card used to punch the source statements (*Autocoder Input Card*, Form A36199).

Each line of the coding sheet is punched into a separate card. The source deck, therefore, consists of a sequenced set of punched cards containing a line-by-line representation of the coding sheets.

The following paragraphs explain the function of each field. The heading information, *Program*, *Programmed By*, and *Date*, are only for documentation, and are not punched.

Identification (Card Columns 76-80)

This five-position field can contain a name created by the programmer to identify the program. This identification will be punched into 76-80 of the object deck only if it appears in a HEADR or RESEQ control card. (See "Control Operation Codes.") However, the identification is not checked on the other Autocoder statements, and serves only to identify the program to which the card belongs. Special, as well as alphameric, characters are permitted.

Page Number and Line Number (Card Columns 1-5)

The page number (columns 1 and 2), in conjunction with the line number (columns 3-5), provides a means of sequencing the cards in the source deck. This enables the programmer to identify and correlate the entries on the coding sheet and assembly listing with the entries in the source deck. Alphabetic, as well as numeric, characters can be used. (If the standard collating sequence is not followed, the processor will place a sequence (S) flag next to the PGLIN field in the assembly listing, as previously explained.)

Label (Card Columns 6-15)

This field, if used, contains the label being defined in this statement.

Operation Code (Card Columns 16-20)

This field contains the operation code.

Operand (Card Columns 21-72)

This field, if used, contains the operand element(s) of the statement.

NOTE: Columns 73-75 should be left blank.

COMMENTS

Comments are remarks or notes written by the programmer in the operand field. At least two blank spaces must separate a comment from the last character of the statement. The comment, punched in the source deck, appears in the assembly listing but is not contained in the object deck, and has no effect on the object program.

COMMENTS CARD

It may, at times, be helpful to insert an entire line of descriptive information. This is done by placing an asterisk in column 6 and using the balance of the line (up to column 72) for comments. When this line of information is punched into a card of the source deck, the asterisk will identify it to the processor as a comments card. The comments will be printed in a single line of the assembly listing at the point of encounter, which can be anywhere in the source deck, except as

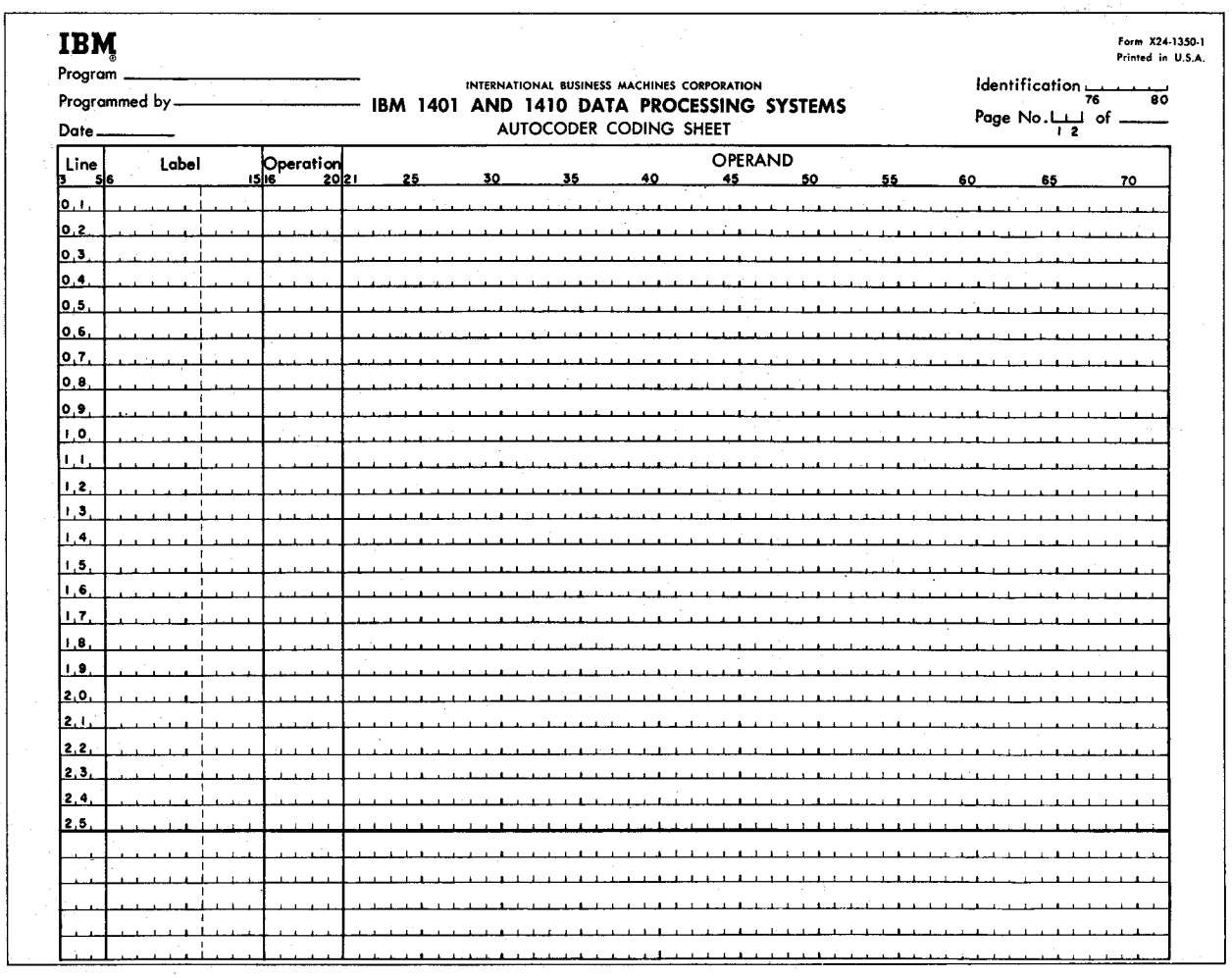


Figure 4. The Coding Sheet Form

noted under "Implied DCW Operation Codes." Comments cards have no effect on the object program and are not included in the object deck. However, a comments card inserted in a series of chained Autocoder operation codes will break the chain. It is, therefore, necessary to restate the operation code and to provide the proper operand on the first source card statement following the comments card, in order to resume a chained action. In summary, a chain which is broken by a comment must be made into two chains.

Types of Operand Entries

This section explains the form and use of the various entries permitted in the operand field of imperative, declarative, Linkage Loader, and control statements.

The operand field of an Autocoder statement is used to specify a variety of information to the processor. The function of a specific entry is dependent upon the type of Autocoder statement in which it appears. The normal operand usage with each of the five types of Autocoder statements is as follows.

STATEMENT TYPE	OPERAND CONTENTS
IMPERATIVE	Symbolic address(es) to be operated upon by the machine instruction, and a d-modifier, when required
DECLARATIVE	Constants, symbols, and/or control parameters necessary to declare the desired fields
LINKAGE LOADER	Symbolic (or actual) addresses and/or control parameters required to convert the object deck into absolute format
CONTROL	Symbolic (or actual) addresses and constant information indicated by the operation code
MACRO	Parameters of the macro statement (These parameters are discussed in the section entitled, "The Macro System.")

All permissible operand entries are explained and illustrated under the following headings:

Basic Addresses
Address Adjustment
Indexing
Literals
Linkage Symbols
System Symbols
Miscellaneous

Basic Addresses

Basic addresses contained in the operand field of an Autocoder statement are the primary elements of information conveyed to the processor. They can be altered or modified by means of additional elements contained in the operand field.

A basic address is the symbolic or actual representation of a core-storage location of the data field or instruction referred to by the Autocoder statement.

A basic address can be in one of three forms:

Symbolic
Asterisk
Actual

Symbolic

A symbolic address is an operand entry that appears elsewhere in the source program as a label. As a rule, this symbol can be defined as a label either before or after the Autocoder statement in which it appears as an address. The exceptions to this rule are as follows:

1. All symbolic operands appearing in ORG, LTORG, and EQU statements must have been *previously* defined within the same program.

2. The symbolic address appearing in an RSV statement must *precede* any other use of this symbol in a program. (See "RSV – Reserve.")

3. The symbolic representations of index registers (X0, X1-X15) and the common data area (COMMON), must never appear in the label field. They cannot be defined by the user because they are predefined labels in the symbol table maintained by the Autocoder processor.

The instruction in Figure 5 illustrates the use of symbolic addresses. The symbols TOTAL and ACCUMULATE are defined as labels elsewhere in the program. The assembled instruction will cause the contents of the core-storage area labeled TOTAL to be moved to the area labeled ACCUMULATE.

NOTE: A symbolic address will receive upward, downward, or NO relocation, depending on the manner in which the symbol is defined.

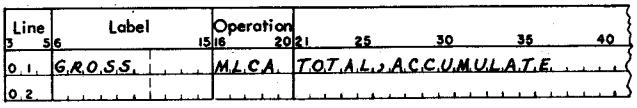


Figure 5. Autocoder Instruction with Symbolic Addresses

Asterisk (*)

An asterisk (11-4-8 punch) can be used as a basic address in an Autocoder statement. When compiling the object program, the processor will replace the asterisk with the relative core-storage address of the last character of the instruction or data field created by the statement in which it appears. However, if an asterisk address is used in a statement that does not cause the generation of an instruction or data area in the object program, the value substituted for the asterisk will be the current location in the object program.

Blank Constants

A field of blanks can be reserved by placing a # character (3-8 punch) in column 21, followed by a number indicating how many consecutive blank core-storage positions are to be defined (Figure 40). A word mark is set in the high-order position of this field.

NOTE: The number of successive blank constants that can be reserved by a DCW statement is limited to 500 positions of core storage. If this limit is exceeded, the processor will reserve only the maximum (500 positions), and attach an "F" flag to the statement on the assembly listing.

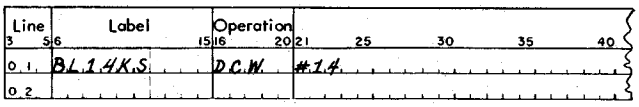


Figure 40. Field of 14 Blanks Defined in a DCW Statement

Address Constants

A DCW statement can be used to define an address constant. The constant is the address of the field whose label is written in the operand. For example (Figure 41), assume that the label MANNO is used in the symbolic program, and that it was assigned the address 00500 by the processor. The programmer can refer to the address of MANNO by using the symbolic label of the DCW statement.

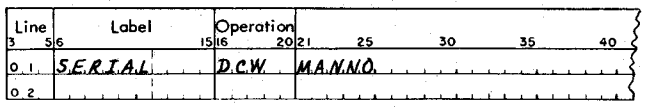


Figure 41. Address Constant

The five-character data field labeled SERIAL (Figure 41) will contain the address of the label MANNO (00500). The Linkage Loader will recognize address constants and adjust them by the proper relocation factor. Thus, SERIAL will contain the relocated address of MANNO.

If an address constant is address adjusted in a DCW statement, the constant is adjusted before it is assigned a storage location. In Figure 42, MANNO (actual address 00500) has been address adjusted by +12. Thus, the location labeled FICA will contain the address constant 00512.

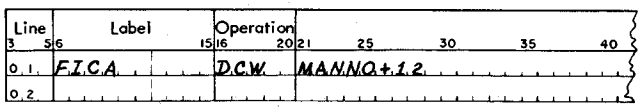


Figure 42. Address Constant with Address Adjustment Defined in DCW Statement

Address constants defined in a DCW statement can be indexed. The zone bit(s) indicating the specified index register becomes part of the constant.

NOTE 1: All address constants receive the same relocation indicators that were assigned to the symbol specified in the operand field.

NOTE 2: An address constant of a linkage or system symbol can be specified, and the desired address will be automatically supplied by the Linkage Loader. However, this form of address constant cannot be address adjusted or indexed.

Signed Address Constants

An address constant defined in a DCW statement can be signed. A and B bits will be generated by the processor over the units position, if the plus (+) sign was placed before the operand. The units position will contain a B bit if the minus (−) sign was used (Figure 43).

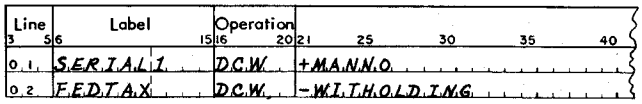


Figure 43. Signed Address Constants Defined in DCW Statement

Implied DCW Operation Codes

If several constants are to be defined in succession, only the first statement (or any statement preceded by a comments card) requires the mnemonic DCW in the operation field (Figure 44).

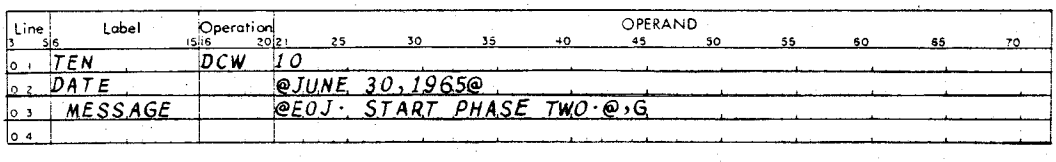


Figure 44. Successive DCW Statements with Blank Operation Columns

DC — Define Constant (no word mark)

The function performed by the DC statement, and the permissible forms of the constants, are identical to those described for the DCW statement. The only difference is that the word mark is absent when the constant is assigned to core storage (Figure 45).

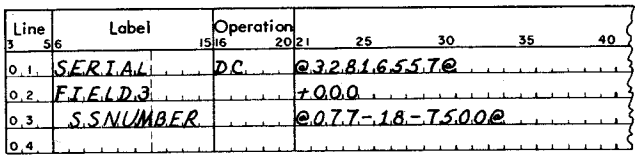


Figure 45. Successive DC Statements with Blank Operation Columns

NOTE: The restriction on the use of an initial word separator character in the DCW statement defining an alphameric constant does not apply to the DC statement.

DS — Define Symbol

The DS statement is used to label and define an area within the subprogram. No information is entered into the area, no word mark is assigned by the processor, and the area is not cleared prior to reservation. The programmer specifies the size of the area, and designates the symbolic label by which it will be referenced. The number of desired consecutive positions of core storage is written in the operand field (Figure 46). The label refers to the low-order position of the area. However, if the label is indented one place, that is, if it begins in column 7, the label will refer to the high-order position. A label is not mandatory.

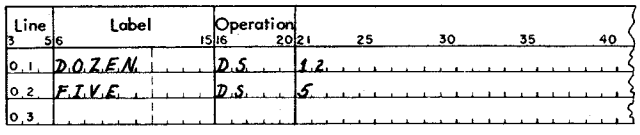


Figure 46. Defining Twelve-Position and Five-Position Areas in DS Statements

Figure 46 illustrates the form of the DS statement. The first entry, labeled DOZEN, defines an area twelve positions long. The second entry, labeled FIVE, defines an area five positions in length.

EQU — Equate

The EQU statement is used to define either a second symbol to reference a specific location, or a symbol for a location not previously labeled. The symbol to be defined is specified in the label field, and the representation of the location to be "equated" is specified in the operand field.

An EQU statement can be used to assign a symbolic label to each of the following:

Actual or symbolic address
Adjusted or modified address
Index register
Asterisk address

Actual or Symbolic Address

The symbol to be defined is specified in the label field. The operand field can contain an actual or symbolic address. If a symbolic address is specified in the operand field, it must have appeared as a label prior to this point in the subprogram. If this condition is not met, the label will *not* be defined.

SYMBOLIC ADDRESS

The EQU statement in Figure 47 will cause the processor to assign the same address to the label INDIVIDUAL that is assigned to the symbol MANNO. Thus, INDIVIDUAL has been equated to MANNO — both labels refer to the same core-storage location and are assigned the same relocation indicator by the processor.

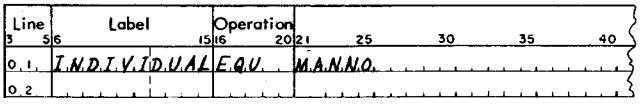


Figure 47. Equating a Symbolic Address

ACTUAL ADDRESS

The EQU statement in Figure 48 will cause the processor to assign the label ACCTNO to machine location 25000.

NOTE: Labels equated to actual addresses will be treated as absolute values and given a NO relocation indicator.

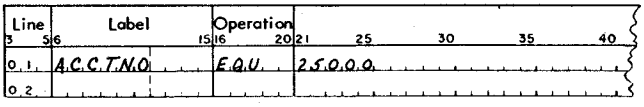


Figure 48. Equating an Actual Address

Adjusted or Modified Address

The operand of an EQU statement can be address adjusted or indexed. The same relocation indicators assigned to the address adjusted and/or indexed operand will be given to the defined label.

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	Label	Operation	OPE
3 5 6		15 16 20	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.

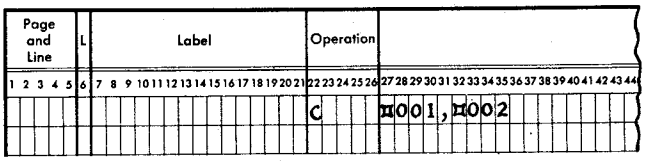


Figure 81. Model Statement for an Incomplete Instruction with Required Parameters

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.

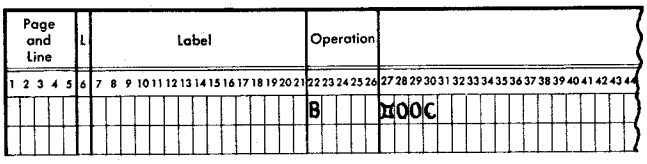


Figure 82. Model Statement for an Incomplete Instruction with Conditional Parameters

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.

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).

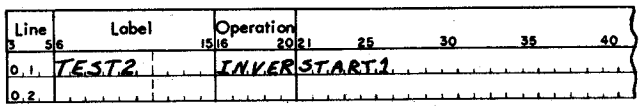
Address Adjustment and Indexing: The parameters in a macro-instruction and the operands in partially complete instructions in a library routine can have address adjustment and indexing.

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.

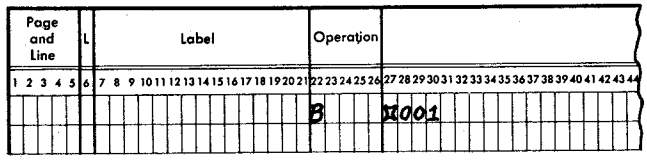
Model statement operands can be indexed. This indexing takes precedence over any indexing of a parameter supplied by a macro-instruction. The model statement index is used.

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

Macro Instruction (Source Program)



Model Statement

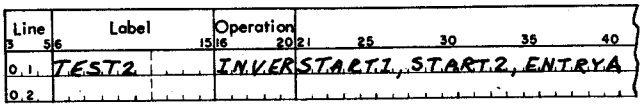


Assembled Symbolic Program Entry

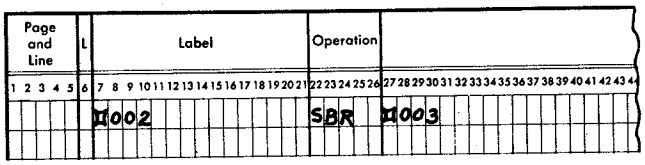
```
TEST2        EQU     *
             B       START1
```

Figure 83. Labeling

Macro Instruction (Source Program)



Model Statement



Assembled Symbolic Program Entry

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

Figure 84. Additional External Labels