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.
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 BASE 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 “DAA — 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:

<table>
<thead>
<tr>
<th>LABEL</th>
<th>OPERATION</th>
<th>OPERAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON$</td>
<td>EXEQ</td>
<td>AUTOCODER, SOF, SIU, NOPRT</td>
</tr>
<tr>
<td>MON$</td>
<td>EXEQ</td>
<td>AUTOCODER, M, NMAC, NOPCH, NOFLG</td>
</tr>
<tr>
<td>MON$</td>
<td>EXEQ</td>
<td>AUTOCODER, M, NOFLG, NOPRT, NMAC, NOPCH</td>
</tr>
</tbody>
</table>

**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 program 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.
### 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:
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.
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 \( B \) followed by a number from 001 to 199 with a B bit over the units position (parameter 001 is \( \text{B}001 \), parameter 2 is \( \text{B}002 \), 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.

**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 \( \text{B}001-199 \) 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)**

<table>
<thead>
<tr>
<th>Line</th>
<th>Label</th>
<th>Operation</th>
<th>12</th>
<th>20</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>TEST2</td>
<td>INTSTART1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Model Statement**

<table>
<thead>
<tr>
<th>Page and Line</th>
<th>Label</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>TEST2</td>
<td>INTSTART1</td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Assembled Symbolic Program Entry**

`TEST2 EQU * B  START1`

Figure 83. Labeling
2. Writes the label that is to reference the first assembled model statement. A \texttt{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 \texttt{001} or \texttt{000A} 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 \texttt{@} symbols. The \texttt{@} symbol itself cannot appear between \texttt{@} symbols. Also, the \texttt{@} symbol can be used only in pairs as a literal identifier. It cannot be used in any other way; e.g., a single \texttt{@} symbol could not be used to represent the \texttt{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 \texttt{EXACT} is used. \texttt{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 \texttt{BOOL} or \texttt{COMP} in the model statements.
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.

### 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

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

### 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.

![Figure 97. bool Pseudo-Macro](image)

Figure 97 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](image)

**SWITCHES**

<table>
<thead>
<tr>
<th>001 * 002 + 003 * 004</th>
<th>LOGICAL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF OFF OFF OFF</td>
<td>0</td>
</tr>
<tr>
<td>ON ON OFF OFF</td>
<td>1</td>
</tr>
</tbody>
</table>

![CONDITIONS](image)

**CONDITIONS**

<table>
<thead>
<tr>
<th>001 * 002 + 003 * 004</th>
<th>LOGICAL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF OFF ON ON</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Figure 99 is a table showing all conditions that will cause the `COMP` statement shown in Figure 98 to be false.

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, alphanumeric 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.)

![Figure 100. Format for COMP Pseudo-Macro](image)

**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.

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.

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.

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.

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.

Figure 103. Checking for a Single Character

Figure 105. MEND Pseudo-Macro