CONTROL DATA®
1700 COMPUTER SYSTEMS

MACRO ASSEMBLER
MASS STORE FORTRAN
INSTANT
CONTROL DATA®
1700 COMPUTER SYSTEMS

MACRO ASSEMBLER
MASS STORE FORTRAN
INSTANT
<table>
<thead>
<tr>
<th>REVISION</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>(10-72)</td>
<td>This printing obsoletes previous versions.</td>
</tr>
</tbody>
</table>

Address comments concerning this manual to:

Control Data Corporation  
Small Computer Development Division  
4455 Easigate Mall  
La Jolla, California  92037  
or use Comment Sheet in the back of this manual.
**1700 MACRO ASSEMBLER**

The Macro Assembler provides a symbolic machine language especially well suited for process control applications as well as for standard general purpose programming. The powerful macro capability feature greatly expedites program development and offers a distinct advantage for process control applications. Running under control of the 1700 Operating System, the Macro Assembler offers the following features:

- Paper tape input
- Relocatable program, data, and common storage areas
- Absolute loading location option
- Free field source statement format
- Symbolic machine instructions
- Pseudo instructions, including conditional assembly and variable field definitions
- Assembly error diagnostics
- Assembly output listing on paper tape or typewriter
- Binary output on paper tape
- Library macros and user-developed macro capabilities
- Relative address optimization

---

**1700 MASS STORE FORTRAN**

Mass Store FORTRAN is a super set of USASI Basic FORTRAN with mass storage input/output features. Running under control of the 1700 Operating System, emphasis is on the compilation of efficient and compact object code.
MACRO ASSEMBLER

Three standard options determine the type of output from the assembler. All three are automatically selected if no OPT pseudo instruction is encountered before the first NAM.

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>List output on standard list unit</td>
</tr>
<tr>
<td>P</td>
<td>Punch binary output on standard punch unit</td>
</tr>
<tr>
<td>X</td>
<td>Load and go output loaded on a mass storage device for subsequent execution</td>
</tr>
<tr>
<td>M &amp;</td>
<td>Macro expansion</td>
</tr>
</tbody>
</table>

Instruction Format

<table>
<thead>
<tr>
<th>label</th>
<th>opcode</th>
<th>address</th>
<th>remark</th>
</tr>
</thead>
</table>

label
Blank or symbolic, 1 to 6 characters, beginning with a letter (excess ignored). An asterisk in column 1 identifies a remark. May be terminated by a tab.

opcode
Follows one or more blanks after the label or tab. If label is blank, opcode may begin in column 2. May be terminated by a tab.

address
Follows one or more blanks after the opcode or a tab. For machine instructions, may contain an address expression or when storage reference instructions are used, the address field may contain an address expression terminated by a comma and followed by Q, I, or B to indicate indexing. See Pseudo instructions for their address field formats. Address expression: a single operand terminated by a tab, blank or comma, or a string of operands joined by the arithmetic operators:

+ addition
- subtraction
* multiplication
/ division (integer)

I always refers to the storage index (core location FF16).
Indirect addresses are enclosed in parentheses.
As an operand, * signifies the current value of the location counter. The address expression is evaluated mod 2^15-1.

remark
After the first blank or a tab in the address field, contents of the statement are considered a comment.
On paper tape, the instruction must be terminated by by a carriage return.
MACRO ASSEMBLER ERROR MESSAGE

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>Doubly defined symbol</td>
</tr>
<tr>
<td>UD</td>
<td>Undefined symbol</td>
</tr>
<tr>
<td>EX</td>
<td>Illegal expression</td>
</tr>
<tr>
<td>OP</td>
<td>Illegal operation code</td>
</tr>
<tr>
<td>RL</td>
<td>Illegal relocation</td>
</tr>
<tr>
<td>OV</td>
<td>Numeric operand overflow</td>
</tr>
<tr>
<td>SQ</td>
<td>Sequence error</td>
</tr>
<tr>
<td>MD</td>
<td>Error in macro definition</td>
</tr>
<tr>
<td>MC</td>
<td>Error in macro instruction</td>
</tr>
<tr>
<td>PP</td>
<td>Error in previous pass (compilation)</td>
</tr>
</tbody>
</table>

SYMBOL TABLE

A table containing the location symbols, locations, and relocation values is printed at the end of pass 3 if the L option is selected. Columns not specified below contain spaces.

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>symbol name</td>
</tr>
<tr>
<td>9-12</td>
<td>location</td>
</tr>
<tr>
<td>13</td>
<td>relocation of location</td>
</tr>
<tr>
<td>15-20</td>
<td>symbol name</td>
</tr>
<tr>
<td>23-26</td>
<td>location</td>
</tr>
<tr>
<td>27</td>
<td>relocation of location</td>
</tr>
<tr>
<td>29-34</td>
<td>symbol name</td>
</tr>
<tr>
<td>37-40</td>
<td>location</td>
</tr>
<tr>
<td>41</td>
<td>relocation of location</td>
</tr>
</tbody>
</table>
## ASSEMBLY LISTING

The assembly list consists of 18 columns of descriptive information related to the source statement, followed by a maximum of 80 columns listing the source statement.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>Line number; truncated to 3 decimal digits</td>
</tr>
<tr>
<td>4</td>
<td>Period</td>
</tr>
<tr>
<td>5</td>
<td>Space</td>
</tr>
<tr>
<td>6</td>
<td>Relocation designator for location</td>
</tr>
<tr>
<td></td>
<td>P  program relocation</td>
</tr>
<tr>
<td></td>
<td>D  data relocation</td>
</tr>
<tr>
<td>7-10</td>
<td>Location in hexadecimal</td>
</tr>
<tr>
<td>11</td>
<td>Space</td>
</tr>
<tr>
<td>12-15</td>
<td>Machine word in hexadecimal</td>
</tr>
<tr>
<td>16-17</td>
<td>Relocation designator for word</td>
</tr>
<tr>
<td></td>
<td>P  program relocation</td>
</tr>
<tr>
<td></td>
<td>-P negative program relocation</td>
</tr>
<tr>
<td></td>
<td>C  common relocation</td>
</tr>
<tr>
<td></td>
<td>-C negative common relocation</td>
</tr>
<tr>
<td></td>
<td>D  data relocation</td>
</tr>
<tr>
<td></td>
<td>-D negative data relocation</td>
</tr>
<tr>
<td></td>
<td>X  external</td>
</tr>
<tr>
<td></td>
<td>blank absolute</td>
</tr>
<tr>
<td>18</td>
<td>Space</td>
</tr>
<tr>
<td>19-72</td>
<td>Input source statement</td>
</tr>
</tbody>
</table>
### MACHINE INSTRUCTIONS

Storage Reference

<table>
<thead>
<tr>
<th>t</th>
<th>Addressing</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>One-word relative—$e_a$ max. value $\pm 7F_{16}$</td>
</tr>
<tr>
<td>blank</td>
<td>Two-word relative or constant addressing</td>
</tr>
<tr>
<td>-</td>
<td>One-word absolute—$e_a$ max. value $FF_{16}$</td>
</tr>
<tr>
<td>+</td>
<td>Two-word absolute</td>
</tr>
</tbody>
</table>

$e_a$ is the value of the address expression plus indexing

**LDA** <sub>t</sub>  
$e_a$  
$(e_a) \rightarrow A$

**LDQ** <sub>t</sub>  
$e_a$  
$(e_a) \rightarrow Q$

**ADD** <sub>t</sub>  
$e_a$  
$(e_a) + (A) \rightarrow A$

**SB** <sub>t</sub>  
$e_a$  
$(A) - (e_a) \rightarrow A$

**ADQ** <sub>t</sub>  
$e_a$  
$(Q) + (e_a) \rightarrow Q$

**AND** <sub>t</sub>  
$e_a$  
$(A) \land (e_a) \rightarrow A$

**EOR** <sub>t</sub>  
$e_a$  
$(A) \lor (e_a) \rightarrow A$

**MULT** <sub>t</sub>  
$e_a$  
$(A) \times (e_a) \rightarrow QA$

**DIV** <sub>t</sub>  
$e_a$  
$(QA) / (e_a) \rightarrow A$

remainder $\rightarrow Q$

In the above instructions, the effective address may be replaced by an address constant:

- $N \pm dddd$ decimal or $N \pm Sshhh$ hexadecimal
- $ACC$  
  \[ cc = \text{two alphanumeric characters to be converted to ASCII} \]
- $Xe_a$  
  $e_a = \text{address expression; if parenthesized will be indirect (sign bit set)}$
STA蔺 e蔺 (A) → e蔺
STQ蔺 e蔺 (Q) → e蔺
JMP蔺 e蔺 Jump to e蔺
RTJ蔺 e蔺 Return jump to e蔺
RAO蔺 e蔺 (e) + 1 → e蔺
SPA蔺 e蔺 (A) → e蔺 Parity → A

Register Reference

Δ value e蔺 is evaluated 2^{15} - 1 and truncated to 8 bits

SLS e蔺 Halt if STOP switch ON

P INP e蔺 Word from device specified by Q → A; next instruction:
  P + 1 if device sends reply
  P + 1 + Δ if device sends reject
  P + Δ if internal reject

P OUT e蔺 Word from A to device specified by Q; next instruction:
  P + 1 if device sends reply
  P + 1 + Δ if device sends reject
  P + Δ if internal reject

ENA e蔺 e蔺 → A_{7-0}, sign extended
ENQ e蔺 e蔺 → A_{7-0}, sign extended
INA e蔺 (A) + e蔺 → A, sign extended
INQ e蔺 (Q) + e蔺 → Q, sign extended
NOP e蔺 No operation
The following instructions are legal only if program protection switch is off or if the instructions are protected.

- **IN**<sub>e</sub><sub>d</sub> Enable interrupt after execution of next instruction
- **IIN**<sub>e</sub><sub>d</sub> Inhibit interrupt
- **SPB**<sub>e</sub><sub>d</sub> Set program protect bit in Q
- **CPB**<sub>e</sub><sub>d</sub> Clear program protect bit in Q
- **EXI**<sub>e</sub><sub>d</sub> Exit from interrupt state; e<sub>d</sub> should be a value from 2nd column of table below.

### Interrupt State Definition:

<table>
<thead>
<tr>
<th>Interrupt state&lt;sub&gt;10&lt;/sub&gt;</th>
<th>Value of Δ&lt;sub&gt;10&lt;/sub&gt; to exit state&lt;sub&gt;16&lt;/sub&gt;</th>
<th>Location of return address&lt;sub&gt;16&lt;/sub&gt;</th>
<th>Location of first instruction after interrupt occurs&lt;sub&gt;16&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>01</td>
<td>04</td>
<td>0104</td>
<td>0105</td>
</tr>
<tr>
<td>02</td>
<td>08</td>
<td>0108</td>
<td>0109</td>
</tr>
<tr>
<td>03</td>
<td>0C</td>
<td>010C</td>
<td>010D</td>
</tr>
<tr>
<td>04</td>
<td>10</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>05</td>
<td>14</td>
<td>0114</td>
<td>0115</td>
</tr>
<tr>
<td>06</td>
<td>18</td>
<td>0118</td>
<td>0119</td>
</tr>
<tr>
<td>07</td>
<td>1C</td>
<td>011C</td>
<td>011D</td>
</tr>
<tr>
<td>08</td>
<td>20</td>
<td>0120</td>
<td>0121</td>
</tr>
<tr>
<td>09</td>
<td>24</td>
<td>0124</td>
<td>0125</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>0128</td>
<td>0129</td>
</tr>
<tr>
<td>11</td>
<td>2C</td>
<td>012C</td>
<td>012D</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>0130</td>
<td>0131</td>
</tr>
<tr>
<td>13</td>
<td>34</td>
<td>0134</td>
<td>0135</td>
</tr>
<tr>
<td>14</td>
<td>38</td>
<td>0138</td>
<td>0139</td>
</tr>
<tr>
<td>15</td>
<td>3C</td>
<td>013C</td>
<td>013D</td>
</tr>
</tbody>
</table>
Shift

$e_k$ is value of address expression; it will be evaluated mod $2^{15}-1$ and truncated to $31_{10}$ maximum.

ARS $e_k$ Shift (A) right end-off $e_k$ places, sign extended
QRS $e_k$ Shift (Q) right end-off $e_k$ places, sign extended
LRS $e_k$ Shift (QA) right end-off $e_k$ places, sign extended
ALS $e_k$ Shift (A) left end-around $e_k$ places
QLS $e_k$ Shift (Q) left end-around $e_k$ places
LLS $e_k$ Shift (QA) left end-around $e_k$ places

Skip if Condition Exists

$e_s$ is value of address expression; it will be evaluated mod $2^{15}-1$ and error flagged if result is greater than $F_{16}$. If the skip condition is met, the next instruction is $P + 1 + e_s$; otherwise the next instruction is $P + 1$.

SAZ $e_s$ $(A) = 0$
SAN $e_s$ $(A) \neq 0$
SAP $e_s$ $(A) = +$
SAM $e_s$ $(A) = -$
SQZ $e_s$ $(Q) = 0$
SQN $e_s$ $(Q) \neq 0$
SQP $e_s$ $(Q) = +$
SQM $e_s$ $(Q) = -$
SWS $e_s$ Switch set
SWN $e_s$ Switch not set
SOV $e_s$ Overflow on
SNO $e_s$ Overflow off
SPE $e_s$ Parity error
SNP $e_s$ No parity error
SPF $e_s$ Program protect fault
SNF $e_s$ No program protect fault
Inter-Register

The destination register expression \( e_r \) may be any combination of A, and M separated by commas, or zero.

- SET \( e_r \) Ones \( \rightarrow \) register(s)
- CLR \( e_r \) Zeros \( \rightarrow \) register(s)
- TRA \( e_r \) \( (A) \rightarrow \) register(s)
- TRM \( e_r \) \( (M) \rightarrow \) register(s)
- TRQ \( e_r \) \( (Q) \rightarrow \) register(s)
- TRB \( e_r \) \( (Q) \lor (M) \rightarrow \) register(s)
- TCA \( e_r \) \( (\overline{A}) \rightarrow \) register(s)
- TCM \( e_r \) \( (\overline{M}) \rightarrow \) register(s)
- TCQ \( e_r \) \( (\overline{Q}) \rightarrow \) register(s)
- TCB \( e_r \) \( (\overline{Q}) \lor (\overline{M}) \rightarrow \) register(s)
- AAM \( e_r \) \( (A) + (M) \rightarrow \) register(s)
- AAQ \( e_r \) \( (A) + (Q) \rightarrow \) register(s)
- AAB \( e_r \) \( (A) + (Q) + (M) \rightarrow \) register(s)
- EAM \( e_r \) \( (A) \lor (M) \rightarrow \) register(s)
- EAQ \( e_r \) \( (A) \lor (Q) \rightarrow \) register(s)
- EAB \( e_r \) \( (A) \lor (Q) \lor (M) \rightarrow \) register(s)
- LAM \( e_r \) \( (A) \land (M) \rightarrow \) register(s)
- LAQ \( e_r \) \( (A) \land (Q) \rightarrow \) register(s)
- LAB \( e_r \) \( (A) \land (Q) \lor (M) \rightarrow \) register(s)
- CAM \( e_r \) \( (\overline{A}) \land (M) \rightarrow \) register(s)
- CAQ \( e_r \) \( (\overline{A}) \land (Q) \rightarrow \) register(s)
- CAB \( e_r \) \( (A) \land (\overline{Q}) \lor (M) \rightarrow \) register(s)
PSEUDO INSTRUCTIONS

NAM s  Identify source language sub-
program; symbolic name s is  
onoptional

END s  Last statement in source language  
subprogram; optional s specifies  
initial entry point of program

ENT s1, s2, . . . , sn  Symbolic entry point names with-  
in subprogram

EXT s1, s2, . . . , sn  Symbolic entry point names in  
external subprograms

EXT *s1, s2, . . . , sn  Symbolic relative entry point  
names in external subprograms

BSS s1(e1), s2(e2), . . . , sn (en)  Storage segment allocation:  
si = symbolic name, ei = number of  
words in block

BZS s1 (e1), s2(e2), . . . , sn (en)  Storage segment allocation set to  
zero: si = segment name, ei = number  
of words in block

COM s1 (e1), s2(e2), . . . , sn (en)  Common storage allocation:  
si = symbolic name, ei = number  
of words in block

DAT s1(e1), s2(e2), . . . , sn (en)  Data common storage allocation:  
si symbolic name, ei number of  
words in block
ADC $e_1, e_2, \ldots, e_n$

Address constant definitions $e_i$; stored beginning in s; ($e_1$) indicate indirect addressing

DC* $e_1, e_2, \ldots, e_n$

Address constant definitions $e_i$; relative to address counter, stored beginning in s; no indirect addressing

ALF $n$, message

Message definition, stored beginning in s, two eight-bit characters per word; $n$ is an unsigned integer specifying number of words in message, or a non-integer specifying end of message

Typewriter characters may be ALF input:

:R Carriage return ($D_{16}$)
:T Horizontal tab ($9_{16}$)
:L Line feed ($A_{16}$)
:B Bell ($7_{16}$)
:F Top of form ($C_{16}$)
:V Vertical tab ($B_{16}$)

NUM $k_1, k_2, \ldots, k_n$

Integer constant specification, stored beginning in s

$k_i$

$\$hhhh_{16}$

$dddd_{10}$ $< 32767$

DEC $k_1, k_2, \ldots, k_n$

Decimal constant specification, stored beginning in s

$k_i = fD_{10} + dB_{10}, f \times 10^{d_1} 2^{d_2} < 2^{15} - 1$

VFD $m_1n_1/v_1, m_2n_2/v_2, \ldots, m_n n_n/v_n$

Variable field definition
<table>
<thead>
<tr>
<th>$m_i$</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Numeric constant</td>
</tr>
<tr>
<td>A</td>
<td>ASCII character</td>
</tr>
<tr>
<td>X</td>
<td>Expression</td>
</tr>
</tbody>
</table>

$n_i$ Bit count
$v_i$ Value

$\text{EQU } s_1(e_1), s_2(e_2), \ldots, s_n(e_n)$
Equate each symbolic name $s_i$ to the expression $e_i$, evaluated modulo $2^{15} - 1$

$\text{ORG } e$
Sets location counter to address expression $e$, evaluated $2^{15} - 1$

$\text{ORG}^*$
Returns location counter to setting before first ORG $e$ pseudo instruction

$s \text{ IFA } e_1, c, e_2$
Skip to ELF, if $c$ is not true

<table>
<thead>
<tr>
<th>c</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>$e_1 = e_2$</td>
</tr>
<tr>
<td>NE</td>
<td>$e_1 \neq e_2$</td>
</tr>
<tr>
<td>GT</td>
<td>$e_1 &lt; e_2$</td>
</tr>
<tr>
<td>LT</td>
<td>$e_1 &gt; e_2$</td>
</tr>
</tbody>
</table>

$\text{EIF } s$
Terminate effect of IFA or IFC

$\text{OPT}$
Enter assembler options from typewriter
L  List output
P  Punch output
X  Execute output
M  List macros
A  Return control to operating system

MON  Return control to operating system after last assembler source program
NLS  Do not list output
LST  List output (after NLS)
SPC e  Skip e absolute spaces between output listing lines
EJT  Eject page
MACROS
Definition

$s \text{ MAC } p_1, p_2, \ldots, p_n$  
Identify macro definition named $s$ with one- or two-character parameters $p_1, p_2, \ldots, p_n$

$\text{EMC}$  
End macro definition

$\text{LOC } s_1, s_2, \ldots, s_n$  
Macro local symbol definition, one or two characters each

$s \text{ IFC } a_1, c, a_2$  
Skip to pseudo instruction ELF, in Macro definition, if $c$ is not true

<table>
<thead>
<tr>
<th>$c$</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>$a_1 = a_2$</td>
</tr>
<tr>
<td>NE</td>
<td>$a_1 \neq a_2$</td>
</tr>
</tbody>
</table>

Macro Instruction

$s \text{ n } p_1, p_2, \ldots, p_n$  
Assemble macro instruction beginning in location $s$ with actual parameters $p_1, p_2, \ldots, p_n$

Macro Library

$\text{MAC}_1 \ldots \text{EMC}$  
Assemble macro definitions $\text{MAC}_1$, $\text{MAC}_2$, into assembler library

$\text{MAC}_2 \ldots \text{EMC} \ldots \text{ENDMAC}$
MASS STORE FORTRAN
FORTRAN ELEMENTS

Constants

Integer

\[ n_1 n_2 \cdots n_m \]
\[ 1 \leq m \leq 5 \]

Hexadecimal

\[ \$n_1 n_2 \cdots n_m \]
\[ 1 \leq m \leq 4 \]

Real

\[ n_1 n_2 \cdots n_m \pm E \exp_{10} \]
\[ 1 \leq m \leq 7 \]

Examples

2
247
31415

\$1
\$AB1
\$FFFF

3.14
+.0749
314E-05
-.3E01

SUBSCRIPTS

For DIMENSION A(L,M,N), where L, M, and N are integer constants the location of A(i,j,k) with respect to the first element of A is

\[ A + (i - 1 + L(i - 1 + M(k - 1)) \times E \]

For DIMENSION A(L,M) the location of A(i,j) is

\[ A + (i - 1 + L(j - 1)) \times E \]

E is the number of words occupied by each element of A.

Form

Examples

(c \times l \pm d)

(l, J)

(l \pm d)

(2, J + 3, 2 \times K + 1)

(c \times l)

(14)

(l)

(K, J + 5)

(c)

c and d are unsigned integer constants; l, J, K are simple integer variables.
<table>
<thead>
<tr>
<th>Variables</th>
<th>Form</th>
<th>Examples</th>
</tr>
</thead>
</table>
| Simple integer            | $a_1 a_2 \ldots a_m$     | N
|                           | $1 \leq m \leq 6$        | I2504        |
|                           | $a_1$                     | M 58         |
|                           | $l, j, k, l, m$ or N      |              |
|                           | $a_2-a_6$                 |              |
|                           | alphanumeric              |              |
| Simple real               | $a_1 a_2 \ldots a_m$     | VECTOR      |
|                           | $1 \leq m \leq 6$        | SPOILS       |
|                           | $a_1$                     | A65          |
|                           | alphabetic other          |              |
|                           | than $l, j, k, l, m, m$   |              |
|                           | or N                      |              |
|                           | $a_2-a_6$                 |              |
|                           | alphanumeric              |              |
| Subscripted integer       | $a_1 a_2 \ldots a_m(i,j,k)$ | NERVE(6,8,6) |
|                           | $1 \leq m \leq 6$        | LO(J)        |
|                           | $a_1$                     | JEL(1,M,3)   |
|                           | $l, j, k, l, m, N$        |              |
|                           | $a_2-a_6$                 |              |
|                           | alphanumeric              |              |
| Subscripted real          | $a_1 a_2 \ldots a_m(i,j,k)$ | TIME(J,K,L) |
|                           | $1 \leq m \leq 6$        | QL(1)        |
|                           | $a_1$                     | ROGER(2,2,1) |
|                           | alphabetic other          |              |
|                           | than $l, j, k, l, m, N$   |              |
|                           | $a_2-a_6$                 |              |
|                           | alphanumeric              |              |

Variables defined by type declarations begin with any letters.
**EXPRESSION Operator**

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Relational</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>.EQ.</td>
<td>.AND.</td>
</tr>
<tr>
<td></td>
<td>Equal to</td>
<td>Conjunctiv</td>
</tr>
<tr>
<td>Subtraction</td>
<td>.NE.</td>
<td>.OR.</td>
</tr>
<tr>
<td></td>
<td>Not equal to</td>
<td>Disjunctiv</td>
</tr>
<tr>
<td>Multiplication</td>
<td>.GT.</td>
<td>.NOT.</td>
</tr>
<tr>
<td></td>
<td>Greater than</td>
<td>Negative</td>
</tr>
<tr>
<td>/ Division</td>
<td>.GE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Greater than or equal to</td>
<td></td>
</tr>
<tr>
<td>** Exponentiation</td>
<td>.LT.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Less than</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.LE.</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

**Elements of Expression**

- **Constants**
- **Variables**
- **Functions**

**Arithmetic expression:** Elements separated by arithmetic operators

**Relational expression:** Arithmetic expressions separated by relational operators

**Logical expression:** Relational expressions separated by logical operators

**Example**

\[
\begin{align*}
A & \quad A \ .GT. \ 16. \\
3.14159 & \quad (D-Q(I) \ * \ Z) \ .LE. \ 3.14159 \\
+16.8946 & \quad (B+C) \ .LT. \ (A-D) \ .OR. \ 1. \ .EQ. \ 0 \\
(A-B(I,J+2)) & \quad B^C(J+4.1/(Z(J,3*K)) \ ) \ * \ SIN(V)
\end{align*}
\]
FORTRAN STATEMENTS

Subprogram Statements

BLOCK DATA
SUBROUTINE name \((p_1, p_2, \ldots, p_n)\)
FUNCTION name \((p_1, p_2, \ldots, p_n)\)
RELATIVE name_1, name_2, \ldots, name_n
EXTERNAL name_1, name_2, \ldots, name_n
CALL name \((p_1, p_2, \ldots, p_n)\)
RETURN
END

Data Declaration and Storage Allocation

REAL list
INTEGER list
DIMENSION \(v_1, v_2, \ldots, v_n\)
COMMON/name/list
COMMON list
EQUIVALENCE \((a_1, b_1, \ldots), (a_2, b_2, \ldots), \ldots, (a_n, b_n, \ldots)\)
DATA list_1/data, \ldots, list_n/data
BYTE/SIGNED BYTE \((a_1, b_1(i_1)(k_1=m_1)), \ldots, (a_n, b_n(i_n)(k_n=m_n))\)

a Variable or array name
b Variable, array or subscripted (i) array element name;
i, k, m \text{ are integer constants;}
\(1 \leq k, m \leq 0\)
k High order bit position of byte
m Low order bit position of byte

Replacement Statements

\(a = \text{arithmetic expression}\)
Control Statements

ASSIGN n TO a
TO integer variable
GO TO n
GO TO (n₁, n₂, . . ., nₘ) integer variable
IF(arithmetic expression) n₁, n₂, n₃
IF(logical expression) statement
DO n i=m₁, m₂, m₃
CONTINUE
PAUSE n
STOP n

Input/Output

READ (u,f)k
READ (u)k
WRITE (u,f)k
WRITE (u)k
READ (α(n),f)k
READ (α(n))k
WRITE (α(n),f)k
WRITE (α(n))k

u Logical unit number
f Format specifier
k List
α File number
i Record size in sectors
j Maximum number of records in file
x Starting sector address; if omitted, assigned by compiler at load time
n Record number of mass storage file
END FILE u
REWIND u
BACKSPACE u

u Logical unit number

Format Specifications

Conversion

Ew.d Single precision floating point with exponent
Fw.d Single precision floating point without exponent
Iw Decimal integer
$w Hexadecimal integer
Aw Alphanumeric, partial words left justified
Rw Alphanumeric, partial words right justified

Editing

wX Intra-line spacing
wH Heading and labeling
* . . . * Heading and labeling
/ Beginning of new record

Standard Printer Carriage Control

<table>
<thead>
<tr>
<th>Character in first column</th>
<th>Action before printing</th>
<th>Action after printing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>single space</td>
<td>single space</td>
</tr>
<tr>
<td>1</td>
<td>page eject</td>
<td>single space</td>
</tr>
<tr>
<td>other</td>
<td>none</td>
<td>single space</td>
</tr>
</tbody>
</table>
LIBRARY FUNCTION

Trigonometric and Exponential

SIN (x)  Sine x radians  Real to real
COS (x)  Cosine x radians  Real to real
ATAN (x) Arctangent x radians  Real to real
SQRT (x) Square root of x  Real to real
ALOG (x) Natural logarithm of x  Real to real
EXP (x)  e to power x  Real to real

Arithmetic

ABS (x)  Absolute value  Real to real
IABS (i)  Integer to integer
FLOAT (i) Conversion  Integer to integer
IFIX (x)  Real to integer
ISIGN (i_1, i_2) Sign of i_2 times |i_1|  Integer to integer
SIGN (x_1, x_2) Sign of x_2 times |x_1|  Real to real

Input/Output Condition

EOF (i)  Test for end-of-file on unit i  Integer to integer
IOCK (i)  Test for parity error on unit i  Integer to integer
Machine Condition

IFault (i) Check overflow, underflow, divide fault

Integer by integer

MASKING FUNCTIONS

NOT (a) Complement

AND (a,b) Logical product

OR (a,b) Inclusive OR

EOR (a,b) Exclusive OR

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>NOT(a)</th>
<th>AND(a,b)</th>
<th>OR(a,b)</th>
<th>EOR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Hollerith to ASCII Conversion

<table>
<thead>
<tr>
<th>Punch</th>
<th>Char</th>
<th>Hex</th>
<th>Punch</th>
<th>Char</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Punch</td>
<td>Space</td>
<td>20</td>
<td>0-8-7</td>
<td>@</td>
<td>40</td>
</tr>
<tr>
<td>11-3-2</td>
<td>!</td>
<td>21</td>
<td>12-1</td>
<td>A</td>
<td>41</td>
</tr>
<tr>
<td>8-7</td>
<td>&quot;</td>
<td>22</td>
<td>12-2</td>
<td>B</td>
<td>42</td>
</tr>
<tr>
<td>12-8-7</td>
<td>#</td>
<td>23</td>
<td>12-3</td>
<td>C</td>
<td>43</td>
</tr>
<tr>
<td>11-8-3</td>
<td>$</td>
<td>24</td>
<td>12-4</td>
<td>D</td>
<td>44</td>
</tr>
<tr>
<td>0-8-5</td>
<td>%</td>
<td>25</td>
<td>12-5</td>
<td>E</td>
<td>45</td>
</tr>
<tr>
<td>8-2</td>
<td>&amp;</td>
<td>26</td>
<td>12-6</td>
<td>F</td>
<td>46</td>
</tr>
<tr>
<td>8-4</td>
<td>'</td>
<td>27</td>
<td>12-7</td>
<td>G</td>
<td>47</td>
</tr>
<tr>
<td>0-8-4</td>
<td>(</td>
<td>28</td>
<td>12-8</td>
<td>H</td>
<td>48</td>
</tr>
<tr>
<td>12-8-4</td>
<td>)</td>
<td>29</td>
<td>12-0</td>
<td>I</td>
<td>49</td>
</tr>
<tr>
<td>11-8-4</td>
<td>*</td>
<td>2A</td>
<td>11-1</td>
<td>J</td>
<td>4A</td>
</tr>
<tr>
<td>12</td>
<td>+</td>
<td>2B</td>
<td>11-2</td>
<td>K</td>
<td>4B</td>
</tr>
<tr>
<td>0-8-3</td>
<td>,</td>
<td>2C</td>
<td>11-3</td>
<td>L</td>
<td>4C</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>2D</td>
<td>11-4</td>
<td>M</td>
<td>4D</td>
</tr>
<tr>
<td>12-8-3</td>
<td>.</td>
<td>2E</td>
<td>11-5</td>
<td>N</td>
<td>4E</td>
</tr>
<tr>
<td>0-1</td>
<td>/</td>
<td>2F</td>
<td>11-6</td>
<td>O</td>
<td>4F</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>30</td>
<td>11-7</td>
<td>P</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>31</td>
<td>11-8</td>
<td>Q</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>32</td>
<td>11-9</td>
<td>R</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>33</td>
<td>0-2</td>
<td>S</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>34</td>
<td>0-3</td>
<td>T</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>35</td>
<td>0-4</td>
<td>U</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>36</td>
<td>0-5</td>
<td>V</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>37</td>
<td>0-6</td>
<td>W</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>38</td>
<td>0-7</td>
<td>X</td>
<td>58</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>39</td>
<td>0-8</td>
<td>Y</td>
<td>59</td>
</tr>
<tr>
<td>8-5</td>
<td>:</td>
<td>3A</td>
<td>0-9</td>
<td>Z</td>
<td>5A</td>
</tr>
<tr>
<td>11-8-6</td>
<td>;</td>
<td>3B</td>
<td>12-8-5</td>
<td>[</td>
<td>5B</td>
</tr>
<tr>
<td>12-8-6</td>
<td>&lt;</td>
<td>3C</td>
<td>0-8-2</td>
<td>\</td>
<td>5C</td>
</tr>
<tr>
<td>8-3</td>
<td>=</td>
<td>3D</td>
<td>11-8-5</td>
<td>]</td>
<td>5D</td>
</tr>
<tr>
<td>8-6</td>
<td>&gt;</td>
<td>3E</td>
<td>11-7-8</td>
<td>]</td>
<td>5E</td>
</tr>
<tr>
<td>12-8-2</td>
<td>?</td>
<td>3F</td>
<td>0-8-6</td>
<td>-</td>
<td>5F</td>
</tr>
</tbody>
</table>
**FORTRAN LANGUAGE FORMAT**

Each line of a FORTRAN coding form represents three fields on punched tape.

<table>
<thead>
<tr>
<th>Field</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Number</td>
<td>1 - 5</td>
</tr>
<tr>
<td>Continuation</td>
<td>6</td>
</tr>
<tr>
<td>Statement Identification</td>
<td>7 - 72</td>
</tr>
<tr>
<td></td>
<td>73 - 80</td>
</tr>
</tbody>
</table>

Statements may be identified by an integer from 1 to 32767. If C appears in column 1, the remainder of the line is ignored but printed with the source listing as a comment.

Statements are written from columns 7 to 72; blanks are ignored. A punch other than zero in column 6 identifies a line as a continuation of the statement from the preceding line. Up to five continuation lines are allowed.

### Comment Line

```
C 1 - 71 character comment C R
```

### Statement Line

```
1 - 5 decimal digits or blanks 1 - 66 character statement C R
```

### Continuation Line

```
5 blanks 1 - 66 character statement continuation C R
```

### Termination Line

```
6 blanks END C R
```
OPTIONS


OPT statement allows the user to select options from the standard input device. The selected options may exist in three ways.

1. L, X, P options assumed with omission of OPT card.

2. OPT card with desired options after column 5.

3. No options specified by OPT card. This permits options to be entered through the standard input comment device.

OPT must be in character positions 2, 3, and 4 immediately preceded by a blank in column 1. Options must be preceded by a blank in column 5. The options may begin any column after column 5.

Options:

P   Relocatable binary object program

L   Source program listing (contains the generated statement numbers)

A   Object code, listing on the list device

M   Condensed object code listing on the list device. Listing contains generated statement numbers and first word of object code generated by each statement.

R   Run-Anywhere object code. This option allows a program in absolute form to be placed anywhere in memory.

K   NSI Basic FORTRAN compatibility; integers occupy two 1700 computer words.

X   Relocatable binary object program placed on the load-and-go file. Disk or drum is used for load-and-go.

Unrecognized parameters and blanks are ignored.

MON

MON statement returns control to the operating system. MON must be in columns 2, 3, and 4 preceded by a blank. The MON card is the last compile in the stack.

25