A cross assembler and simulator for the EMMY microprogrammable processor has now been developed and will shortly be generally available for EMMY Lab users. The program is written in ALGOL W and presently exists on the Campus 360/67, however, in the future it should also be available at SLAC or the SCIP 168.
EMMY/360 CROSS ASSEMBLER

by

Thomas S. Hedges

December 1975

Technical Note No. 74

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University
Stanford, California

The work herein was supported in part by the Army Research Office-Durham under contract DAHC 04-76-G-0001.
Digital Systems Laboratory

Stanford Electronics Laboratories

Technical Note No. 74

December 1975

EMMY/360 CROSS ASSEMBLER

by

Thomas S. Hedges

ABSTRACT

A cross assembler and simulator for the EMMY microprogrammable processor has now been developed and will shortly be generally available for EMMY Lab users. The program is written in ALGOL W and presently exists on the Campus 360/67, however, in the future it should also be available at SLAC or the SCIP 168.

The work herein was supported in part by the Army Research Office-Durham under contract DAHC 04-76-G-0001.
I. THE CROSS ASSEMBLER LANGUAGE

A. CHARACTER SET: The cross assembler accepts EBCDIC characters including ASCII: ' [ , ] , { , }'.

B. IDENTIFIERS: Identifiers may be from one to eight (8) characters. Characters beyond the eighth will be ignored by the assembler. Identifiers conform to the following:

1. First character must be ALPHA(BETIC) upper or lower case; or the dollar sign '$'.

2. The second and following characters, if any, may be ALPHA, NUMERIC ('0'-'9'), the dollar sign or the underscore ('_').

C. RESERVED IDENTIFIERS: Certain identifiers are reserved for special use of the assembler and are not available to the programmer as ordinary identifiers. The reserved identifiers are all upper case alphabetic characters.

D. COMMENTS: Comments may be included on any assembler statement by prefacing the comment with a period ('.') or vertical bar ('|'). A comment may be the only item on a line, and totally blank lines may be included in the input.

E. STATEMENT FORMAT: Statements are coded entirely within card columns 1-72 and no continuation is allowed. The contents of columns 73-80 is printed on the listing but otherwise ignored.

    [<LABEL ID>:] [<T-STATEMENT>] ;[<A-STATEMENT>];[<UPDATE PTR>]]

Machine code statements follow the basic form given above, with numerous minor variations, of course. Blanks, beyond a single one used to delimit other quantities, are ignored and coding is free form within a statement.

If no a-statement is coded then it and the semicolon preceding are omitted, and this also precludes coding any <UPDATE PTR>. Likewise if the t-statement is not coded, then it is omitted with the a- statement beginning as shown with a semicolon.

One or more label identifiers may be attached to a statement by coding each before other items on a given line, and following each LABEL I.D. with a colon (':'). If a LABEL(s) is left 'hanging', (i.e. coded on a line containing no machine statement)
then the label is assigned the current value of the location counter.

F. IDENTIFIER TYPES: The assembler defines different types of identifiers as follows:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>RANGE OF VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>ABSOLUTE NUMERIC</td>
<td>$-2^i \leq n \leq 2^i - 1$</td>
</tr>
<tr>
<td>SYMB</td>
<td>SYMBOLIC LABEL</td>
<td>$0 \leq n \leq 2^{12} - 1$</td>
</tr>
<tr>
<td>REG</td>
<td>HARDWARE REGISTER</td>
<td>R0,R1,R2,R3,R4,R5,R6,R7</td>
</tr>
<tr>
<td>MASK</td>
<td>CONDITIONAL MASK</td>
<td>ALL $2^n$ of $2^9 \cdot (\overline{0}) \cdot (\overline{1}) \cdot (\overline{0})$</td>
</tr>
</tbody>
</table>

G. LITERALS: The assembler accepts literal constants coded in any of the forms below. All literal constants are like an identifier type ABS, except * which is SYMB.

- **DECIMAL** nnnnnn $(\leq 2^i - 1)$
- **HEX** X'nnnnnnnnn' $(\leq 8$ HEX DIGITS$)$
- **OCTAL** O'nnnn-n' $(\leq 2^i - 1)$
- **BINARY** B'(\overline{0})\cdot(\overline{1})\cdots(\overline{0})' $(\leq 32$ BINARY DIGITS$)$
- **CHARACTER** C'cccc' $(\leq 4$ CHARACTERS$)$

(NOTE: Character constants are converted to ASCII and packed right justified in 8 bit fields. No parity bit is given so the high bit of all bytes is zero.)

- **LOCATION COUNTER** * [TYPE IS SYMB]

(NOTE: The * gives the location counter value or the address of the current instruction being processed.)
H. EXPRESSIONS: Literals and identifiers of types ABS and SYMB may appear in expressions as follows (and only as below):

**Unary**
- `-` — AB S NEGATE
- `+` — ++ABS NULL (NO ACTION)

**Binary**
- `-` — ABS _1_ — ABS _2_ YIELDS ABS RESULT
  SYMB _1_ — SYMB _2_ YIELDS ABS RESULT
- `+` — ABS _1_ ++ ABS _2_ YIELDS ABS RESULT
  ABS ++ SYMB YIELDS SYMB RESULT
  SYMB _1_ ++ SYMB _2_ ERROR

Multiple operators are evaluated from left to right. No parentheses may be used.

I. **PSEUDO-OPS:**

1. **DC — DEFINE CONSTANT**

   DC <EXPRESSION>

   This statement reserves one word of storage. The statement may have a label, whose value would be the address of the constant. The <EXPRESSION> may be ABS or SYMB.

2. **BLK — BLOCK**

   BLK <ABS-EXPR>

   Reserve <ABS-EXPR> words of storage. The label, if coded, is the address of the first word. The expression must be ABS.

3. **ORG — ORIGIN LOCATION COUNTER**

   ORG <EXPRESSION>

   Begin assembling code (starting with next statement) at location <EXPRESSION>, which may be ABS or SYMB. It is poor form to label an ORG statement although the assembler probably would allow it.
4. EQU EQUATE SYMBOL

\[ <\text{IDENTIFIER}> \text{ EQU } \{ \langle \text{REG} > \text{ \{ \langle \text{EXPRESSION} > \} } \langle \text{MASK} > \langle \text{IDENTIFIER} > \} \]

**NOTE:** MASKS are defined using 'MASK' function as follows:

**MASK(]**

The four fields are all ABS and **EIGHTBITS** is 0 ≤ 255 while the other fields are 0 or 1 only.

**EIGHTBITS** IS THE TEST MASK

\[ <\text{NOT}> \equiv 1 \text{. INVERTED SENSE} \]

\[ <\text{ZERO}> \equiv 1 \text{. TEST INVERTED (FOR ZERO)} \]

\[ <\text{CODES}> \equiv 1 \text{. TEST INDICATOR CODES} \text{} \]

(INSTEAD OF CONDITION CODES)

The EQU **IDENTIFIER** must not have been used as a label identifier nor may it appear at the left in another EQU. The **IDENTIFIER** is given the type and value of the quantity on the right of the EQU. A register, mask define, expression, or identifier may appear on the right in the EQU. A restriction exists that any identifier appearing at the right, either alone or in an expression, must have been given its value earlier in the program.

5. END

**END** [\langle \text{LABEL} > \]

The END PSUEDO-OP marks the last physical statement of the program. The optional **LABEL** specifies a point to transfer control to begin execution.
J. T - STATEMENT

1. ARITHMETIC

To store result & set codes:

\[
Raf := Raf \{\begin{array}{c}
+ \\
- \\
\end{array}\} \begin{array}{c}
Rbf \\
\text{LITERAL} \\
\end{array} \}
\]

To only set codes:

\[
Raf \{\begin{array}{c}
+ \\
- \\
\end{array}\} \begin{array}{c}
Rbf \\
\text{LITERAL} \\
\end{array} \}
\]

2. LOGICAL

\[
Raf := \begin{cases}
0 \text{ or } -1 \\
\text{LITERAL or LABEL} \\
\text{NOT } \langle \text{OP2} \rangle \\
\text{Raf } \langle \text{LOG} \rangle < \text{OP2} \\
Rbf
\end{cases}
\]

"-" MAY BE USED FOR 'NOT'
\(<\text{OP2}>: \text{Rbf } \text{or LITERAL} \]
\(<\text{LOG}>: \text{AND } \text{or OR } \text{or NAND } \text{or}
\text{NOR } \text{or XOR } \text{or XNOR}

NOTE: \(Raf := Rbf\) DOES A 'LTR'. THAT IS LOADS AND SET CONDITION CODES

*1* LITERAL is assembled short if \(0 < \text{LIT} < 7\) and an A-STATEMENT is coded, otherwise it is long.

*2* LITERAL or LABEL with value 0 or -1 are assembled short, other values generate a long literal form.
3. **SHIFT.ROTATE**

**SINGLE:**

\[
\text{Raf} \quad \text{<S-OP> \{Rbf \{LITERAL\}}
\]

**DOUBLE:**

\[
\text{Raf, Raf} \quad \text{<S-OP> \{Rbf \{LITERAL\}}
\]

\[
\text{<S-OP>:} \quad \text{<< LEFT LOGICAL} \\
\quad \text{<O LEFT ROTATE} \\
\quad \text{<< RIGHT LOGICAL} \\
\quad \text{@< RIGHT ARITHMETIC}
\]

4. **EXTENDED**

**TRANSFER**

\[
\text{Raf} = \text{Rbf} \quad \text{[NOTE: CONDITION CODES NOT SET]}
\]

**DIVIDE STEP**

\[
\text{DIV} \ (\text{Raf, Rbf})
\]

**MULTIPLY STEP**

\[
\text{MUS} \ (\text{Raf, Rbf})
\]

**EXCESS SIX**

\[
\text{XS6} \ (\text{Raf, Rbf})
\]

---

\[#1# LITERAL is assembled short if 0 < LIT < 7 and an A-STATEMENT is coded, otherwise it is long.\]
5. **EXTRACT/INSERT**

\[ Raf(\langle \text{LIST}\rangle) = Rbf(\langle n_{i} : n_{o}\rangle) \{\text{CLEAR}\} \{\text{INSERT}\} \]

\(<\text{LIST}\rangle\) is one or more of the following separated by commas

a. \(<i_{i} : i_{o}\rangle\) position \(i_{o}\) through \(i_{i}\)

b. \(<i_{2}\rangle\) position \(i_{2}\)

\(Rbf\) is rotated assuming the \(<n_{o}\rangle\) of \(Rbf\) will be matching the last \(i\) argument (furthest to right) of \(<\text{LIST}\rangle\) for \(Raf\).

If 'CLEAR' is coded an EXTRACT is done otherwise an INSERT. The ACTION is to assign the bit field given by \(<n_{i} : n_{o}\rangle\) in \(Rbf\) to \(Raf\) into those bits given by \(<\text{LIST}\rangle\) and leave the others the same (or clear them if 'CLEAR').

6. **CONDITIONAL**

\([\text{NOT}] \langle \text{MASK} \rangle \Rightarrow [.; ] \langle \text{A-STATEMENT}\rangle\)

The entire statement is surrounded by a set of parentheses, including the \(<\text{A-STATEMENT}\rangle\). The \(<\text{MASK}\rangle\) must be either the 'MASK(\(i_{j} i_{j} i_{j}\))' or an IDENTIFIER with TYPE MASK. The test may be inverted by specifying 'NOT'. NORMAL if the test is satisfied the \(<\text{A-STATEMENT}\rangle\) is EXECUTED, otherwise it is skipped.

\[\#1\]

\*1\* If \(i_{j} > i_{i}\) then positions 31 to \(i\) and \(i\) to 0 are selected (it wraps around). Remember not all possible masks can be represented in the 18 bit literal field.
K. A-STATEMENT

1. STORE REGISTER
   \[ M(\text{<EXPRESSION>}) = \text{Ref} \]

2. LOAD REGISTER
   \[ \text{Ref} = M(\text{<EXPRESSION>}) \]

3. LOAD IMMEDIATE
   \[ \text{Ref} = \text{<EXPRESSION>} \]
   \[ \text{[this includes <LABEL>'s and type SYMB]} \]

4. INDIRECT ACCESS
   \[ M(\text{Ref}) = X(\text{Rdf}) \]
   \[ \text{Ref} = X(\text{Rdf}) \]
   \[ X(\text{Ref}) = \text{Rdf} \]
   \[ X(\text{Ref}) = M(\text{Rdf}) \]
   \[ \text{Ref} = M(\text{Rdf}) \]
   \[ M(\text{Ref}) = \text{Rdf} \]
   \[ \text{Ref} = \text{Rdf} \]

   NOTE: To do POINTER UPDATE include

   Either
   \[ ; \text{Ref} = \text{Ref} \{1\} \text{LIT} \text{or} \]
   \[ ; \text{Rdf} = \text{Rdf} \{2\} \text{LIT} \text{or} \]
   \[ ; \text{Ref} = \text{Ref} \{2\} \text{LIT} \text{; Rdf} = \text{Rdf} \{1\} \text{LIT} \]

5. POINTER-MOD AND LOOP

   INC \text{Ref}
   DEC \text{Ref}
   \[ \text{Ref} = \text{Rcf}\{ }\text{Rdf} \]

   NOTE: To do CONDITIONAL LOOP include

   \[ \left[ \left[ \text{NOT} \{ \right\} \left\{ \right\} \right] \{=\} \{\Rightarrow \} \{\text{R01 }\text{LIT} \{\text{<LABEL>}} \} \right] \]

   If \text{<LABEL>} is used must be equivalent to \text{R01 }\text{LIT},
   where \(-8 \leq \text{LIT} \leq 7\)
6. CONDITIONAL BRANCH

\[ \text{[NOT] <MASK> [=\Rightarrow] \{ RO(\#)Lit \rangle <LABEL> } \]

Branch part is above. <MASK> the same as for condition T-STATEMENT.
APPENDIX

A. PREDEFINED MASKS

ZERO
NEGATIVE
POSITIVE
OVERFLOW
CARRY CARRY BIT = 1
HIGH HIGH BIT = 1
LOW LOW BIT = 1
SAME ALL BITS 0 or 1
ODD PARITY IS ODD
BUSY CPU BUS ACCESS IS BUSY

B. RESERVED WORDS

R0 INSERT
R1 CLEAR
R2
R3 MUS
R4 DIV
R5 X65
R6
R7 DEC
INC

AND ZERO
OR POSITIVE
NAND NEGATIVE
NOR OVERFLOW
XOR CARRY
NOT HIGH
LOW

M0 SAME
E0 ODD
BLK BUSY
END
DC
ORG
A+ + only when directly followed by '('
M+