UCID-16140

REFER QUESTIONS TO: Waldo Magnuson, Jr.	LAWRENCE RADIATION LABORATORY	ELECTRONICS ENGINEERING **REPORT** LIVERMORE, CALIFORNIA	UNIVERSITY OF CALIFORNIA	**LER** 72-103401	
ORIG. W. G. Magnuson, Jr.	ASSEMBLER PROGRAM FOR THE INTEL MCS-8 8008 CPU			DATE 2/15/73	REV.
APPROVED				PAGE 1 OF 24	B*

1.0 SUMMARY

This report is both a user's manual for the CDC-7600 computer program MCS8 as well as a reference manual for the INTEL 8008 symbolic assembly language. The MCS8 program accepts as input, programs written in INTEL 8008 CPU assembly language using mnemonic symbols for the instruction operations and creates two output files: A standard assembly output (symbol table, object, and source code) and a formatted object code output. The program will also write a magnetic tape (a program option) in formatted object code in 1601 format (PN tape) which can then be used to punch a paper tape utilizing the CDC 160A computer in building 113. The reference manual for the symbolic assembly language appears as appendix A which is essentially a duplication of Section II of the INTEL writeup titled "MC8 8008 Assembler."

2.0 Reading MCS8 Program From Photostore

The MCS8 program is stored in the ELEPHANT photostore under the "take" directory: 558850:INTEL.

After logging onto the OCTOPUS system on a CDC-7600 computer system read MCS8 from the photostore as follows:

```
ELF / .5 .1
.RDS .558850:INTEL:MCS8
.END
RDS

 ALL DONE
```

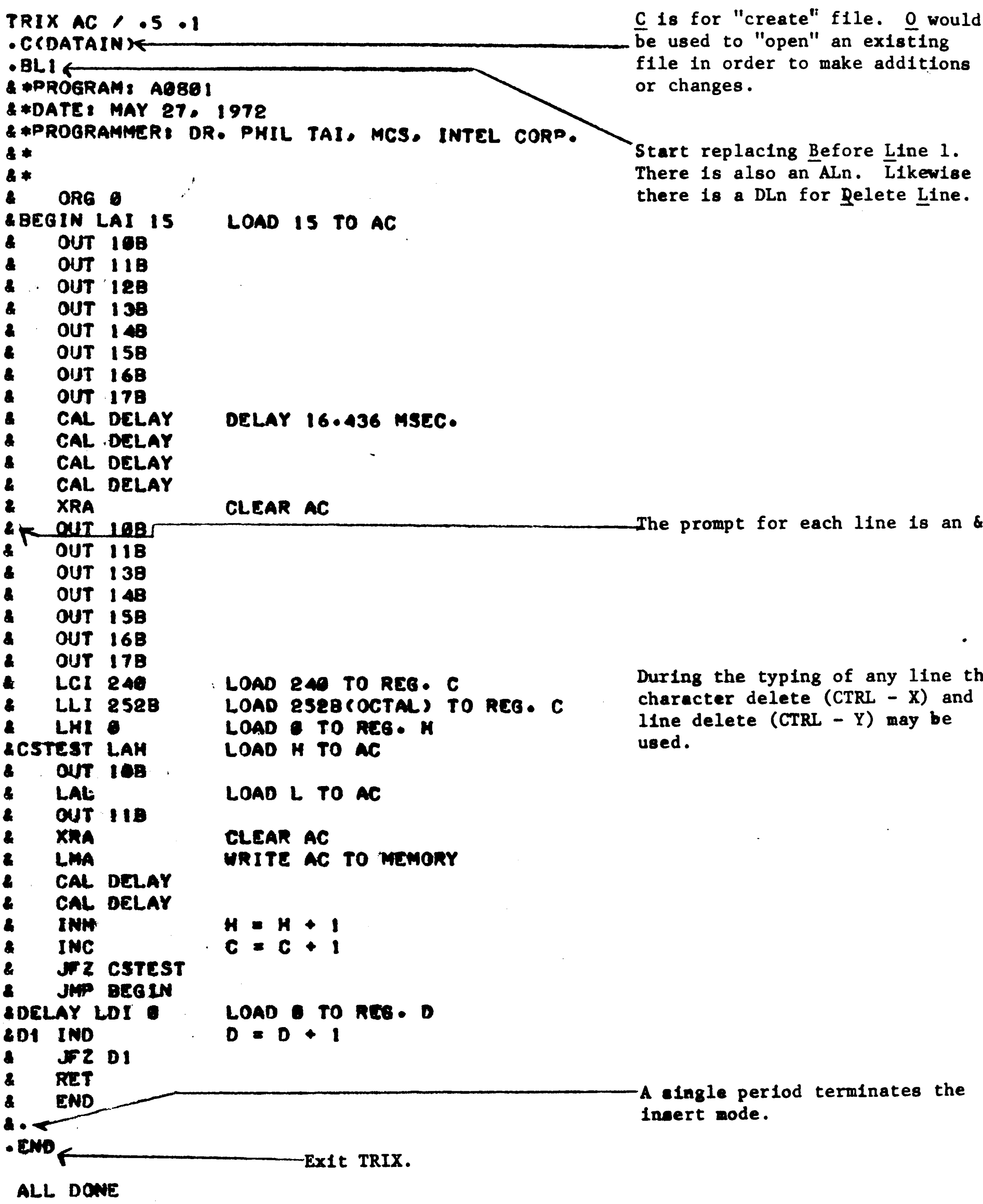
After the above operations the program will exist on disk file available for use as described in the following section.

3.0 Creating a Input Data File

Before running MCS8, an ASCII disk data file containing the 8008 program in terms of the 8008 symbolic assembly language instructions must be available. This data file can be created in a number of ways. For example cards can be punched and read onto disk through an online card-reader or RJET, or one of the OCTOPUS editor routines (TRIX, NAB, MICROPUS) can be used online to create a data file. The routines TRIX and NAB will be used for illustrative purposes. Figure 1 shows a data file created using TRIX and Figure 2 with NAB. The figures have been annotated to help show what was going on. The TRIX example was taken from the INTEL MCS-8 CPU manual and the NAB example is a program with deliberate errors to show some of the online errors when running MCS8.

The details for using both TRIX and NAB are in the references at the end of this report. TRIX is by far the most powerful and flexible of the two editors and well worth the time learning how to use it.

*Changed pages 1, 4 and 6.



```
TRIX AC / .5 .1
.C(DATAIN)
.BL1
&*PROGRAM: A0801
&*DATE: MAY 27, 1972
&*PROGRAMMER: DR. PHIL TAI, MCS, INTEL CORP.
&*
&*
&    ORG 0
&BEGIN LAI 15     LOAD 15 TO AC
&    OUT 10B
&    OUT 11B
&    OUT 12B
&    OUT 13B
&    OUT 14B
&    OUT 15B
&    OUT 16B
&    OUT 17B
&    CAL DELAY    DELAY 16.436 MSEC.
&    CAL DELAY
&    CAL DELAY
&    CAL DELAY
&    XRA          CLEAR AC
&    OUT 10B
&    OUT 11B
&    OUT 13B
&    OUT 14B
&    OUT 15B
&    OUT 16B
&    OUT 17B
&    LCI 240      LOAD 240 TO REG. C
&    LLI 252B     LOAD 252B(OCTAL) TO REG. C
&    LHI 0        LOAD 0 TO REG. H
&CSTEST LAH       LOAD H TO AC
&    OUT 10B
&    LAL          LOAD L TO AC
&    OUT 11B
&    XRA          CLEAR AC
&    LMA          WRITE AC TO MEMORY
&    CAL DELAY
&    CAL DELAY
&    INH          H = H + 1
&    INC          C = C + 1
&    JFZ CSTEST
&    JMP BEGIN
&DELAY LDI 0      LOAD 0 TO REG. D
&D1 IND           D = D + 1
&    JFZ D1
&    RET
&    END
&.
.END
 ALL DONE
```

C is for "create" file. O would be used to "open" an existing file in order to make additions or changes.

Start replacing Before Line 1. There is also an ALn. Likewise there is a DLn for Delete Line.

The prompt for each line is an &

During the typing of any line th character delete (CTRL - X) and line delete (CTRL - Y) may be used.

A single period terminates the insert mode.

Exit TRIX.

Figure 1.

```
NAB / .5 .1
TYPE NAME OF FILE.                                   Length ØØ implies we are creating
DATABAD                                              a new file.
L=      00
OK
R 0                                                  Start replacing at location Ø

*EXAMPLE TO SHOW MCS ERROR DIAGNOSTICS
OKR
*
OKR
*
OKR                                                  Prompt for each line.
   ORG 178B
OKR
   LA$
OKR
JMP1 OUT 100
OKR
JMP1 OUT 11B
OKR
   CAL DELAY
OKR
   TFX
OKR
   OUT
OKR
   JMP STOP
OKR
STOP END
OKR                                                  Terminates the insertation mode.
!
OK
T 0 20                                               Type starting at location Ø the
  00000000*EXAMPLE TO SHOW MCS ERROR DIAGNOSTICS     next 20 lines.
  00000004*
  00000005*
  00000006   ORG 178B
  00000010   LA$
  00000011JMP1 OUT 100
  00000013JMP1 OUT 11B
  00000015   CAL DELAY
  00000017   TFX
  00000020   OUT
  00000021   JMP STOP
  00000023STOP END
  00000024 END OF FILE
OK
END                                                  Exit NAB

 ALL DONE
```

Figure 2

4.0 Running MCS8 From a Teletype

Once we have obtained MCS8 from the Photostore and have created a data file we are able to run MCS8.

To start the program in execution, type:

```
MCS8 /  .5 .1
PLEASE TYPE INPUT FILE NAME (A7)
DATAIN
TYPE LINE-FEED OR TAPE VAULT NO.
DC2Ø3
======================
8ØØ8 INTEL ASSEMBLER
======================

 ALL DONE
```

During execution four disk files are created by MCS-8. They are:

BUFFER — Used by the program as a ENCODE-DECODE buffer.
MCSMID — Used for temporary storage during execution.
MCSBIN — The formatted object code in 1601 format.
MCSOUT — The standard output which contains a symbol table and source and object code.

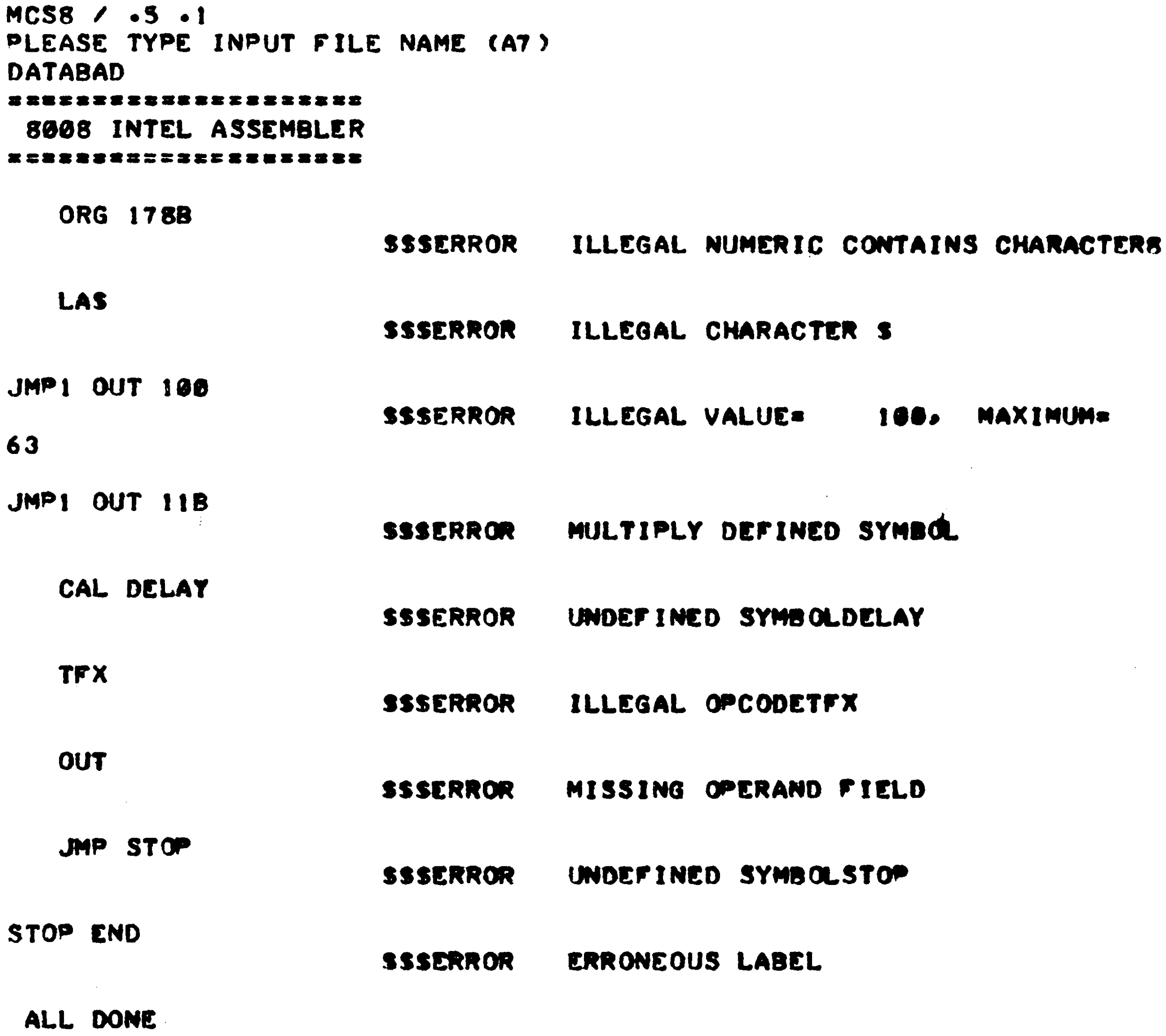
If a line-feed response is used when running the program, no magnetic will be written. The first two files can be ignored (or destroyed). The file MCSBIN can be punched as cards and the cards converted to paper tape on the PDP-1 computer using the HAT routine. To first punch the cards on the OCTOPUS system, type:

```
PUNCH MISCBIN / .5 .1
BOX&ID?
BOX NØØ PROGRAM:  A:  AØ8Ø1

 ALL DONE
```

The file MCSOUT is the standard listing file and can be either listed on the teletype or sent to a printer or an RJET using OUT or ALLOUT. For example using OUT and the printer:

```
OUT PRINTER MCSOUT / .5 .1
BOX&ID?
BOX NØØ PROGRAM:  AØ8Ø1

 ALL DONE
```

In the above example it is implied that tape DC2Ø3 will be available to the OCTOPUS computer operator (use routine SAMTOP to request a tape from the vault). If you want it then use *vault-number. You can use a tape name like *MCS8 if you can pick the tape up from the computer in a reasonable amount of time (1 hour). The CDC-160A computer in building 113 is used for punching a paper tape using the magnetic tape. The instruction book at the CDC-160A tells how to use the computer, instructions are also contained in LER71-10506 "Preparing and Verifying Punched Paper Tapes for the CLI Program."



```
MCS8 / •5 •1
PLEASE TYPE INPUT FILE NAME (A7)
DATABAD
=====================
 8ØØ8 INTEL ASSEMBLER
=====================

   ORG 178B
                    $$SERROR   ILLEGAL NUMERIC CONTAINS CHARACTERS

   LA$
                    $$SERROR   ILLEGAL CHARACTER $

JMP1 OUT 1ØØ
                    $$SERROR   ILLEGAL VALUE=    1ØØ,  MAXIMUM=
63

JMP1 OUT 11B
                    $$SERROR   MULTIPLY DEFINED SYMBOL

   CAL DELAY
                    $$SERROR   UNDEFINED SYMBOLDELAY

   TFX
                    $$SERROR   ILLEGAL OPCODETFX

   OUT
                    $$SERROR   MISSING OPERAND FIELD

   JMP STOP
                    $$SERROR   UNDEFINED SYMBOLSTOP

STOP END
                    $$SERROR   ERRONEOUS LABEL

 ALL DONE
```

Figure 3

The paper tapes produced by means of either the magnetic tape or punch cards are Intel format PN tapes in positive logic (high level = P= 1). They can be used on the PROM programmer. It is recommended that the magnetic tape produced paper tape be used for reasons of efficiency.

Figure 3 shows the teletype output produced when the file DATABAD was used as input to MCS8. Normally the input file would be corrected with a text editor and then MCS8 rerun.

Figure 4 and 5 show the listings of files MCSOUT and MCSBIN for an 8008 program which assembled with no errors.

Need Help?

If you need help running the MCS8 assembler program on the CDC-7600 system or in punching cards let me know. There are many people familiar with the use of both NAB and TRIX who can help you. I will be glad to help you with these programs as well. In addition, Terry Allison or Jack Oliver can assist with the operation of the PDP-1 computer.

Magnuson, Jr.
W. G. MAGNUSON, JR.
Electronics Engineering Department

Distribution:
W. G. Magnuson, Jr. (25 copies)
H. C. McDonald
EE Division Leaders
EE Group Leaders

```
TRIX AC / .5 .1
.O(MCSBIN)
66 LINES.
.NN T
****************************************************************
********
****************************************************************
********
      0 BNNNNNPPNF BNNNNNNNPF BNPNPNPNPF BPNPNPNNNF
        BNPNPNPPPF BNPNNNPPNF BNNNNPNPPF BNNNNNNNNF
      8 BNPNNNPNNF BNNNNNNNNF BNNNNNNNNF BNNNNNPPNF
        BNNNNNNNPF BNPNPNPPPF BNPNNNPPNF BNNPPPNPNF
     16 BNNNNNNNNF BNNNNNNNNF BNPNNNPPNF BNPNNNNNPF
        BNNNNNNNNF BPNPNPNNNF BNPNPNPPPF BNPNNNNNPF
     24 BNNNPNPPNF BPPPPPPPPF BPNPNPNPNF BNPNPNPNPF
        BNNPNNPPNF BPPPPPNNNF BNPNNNPPNF BNNPPPNPNF
     32 BNNNNNNNNF BNPNNNNNPF BNNNPNPPNF BPPPPPPPPF
        BPNPNPNPNF BNPNPNPNPF BNNNPPNPNF BPPNNNNNPF
     40 BNNNPPNPNF BPPNNPNNNF BNNPNNNNNF BNPNNPNNNF
        BNNNPPPPNF BNNNNNNNNF BPPNNNNNPF BNPNPNNPPF
     48 BNNNPNPNNF BPNNNNNNNF BPPNNPNNNF BNPNNNPPNF
        BNNPPPNPNF BNNNNNNNNF BNNNNNPPNF BNNNNNNNPF
     56 BNPNPNPNPF BNNNNNPPPF BNNNPPPPNF BNPPPNNPPF
        BNNNPPNNNF BNPNNPNNNF BNNPPPPNNF BNNNNNNNNF
     64 BNNNNNPPPF BNNNPPPPNF BPNPPPNPNF BNNNPPNNNF
        BNPNNPNNNF BNPNNNNPPF BNNNNNNNNF BNNNNNPPPF
     72 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
     80 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
     88 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
     96 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    104 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    112 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    120 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    128 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    136 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    144 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    152 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    160 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    168 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    176 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    184 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    192 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    200 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    208 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    216 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    224 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    232 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    240 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
    248 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
        BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
.END

 ALL DONE
```

Figure 5

```
TRIX AC / .5 .1
.O(MCSOUT)
66 LINES.
.NN T
=====================
     SYMBOL   VALUE
=====================
   1 BEGIN        0
   2 TAPE        11
   3 TTY         17
   4 TTYIN       30
   5 TTYD1       58
   6 ST          60
   7 TTYD2       65
   8 ST2         67


==================================================
LOC    OBJECT CODE     SOURCE STATEMENTS
==================================================
    0 *30             * TTY TAPE READER CONTROL
    0 *30             * INTEL PROGRAM A0800-00 5/22/72 (DR. PHIL TAI)
    0 *30                    ORG 0
    0   6   1         BEGIN  LAI 1       SUPRESS TTY
    2  85                    OUT 12B     OUTPUT 2
    3 168                    XRA         CLEAR AC
    4  87                    OUT 13B     OUTPUT 3 - TAPE RDR. CONTROL
    5  70  11   0            CAL TAPE    REO. TAPE RDR. CONT. RT.
    8  68   0   0            JMP BEGIN
   11   6   1         TAPE   LAI 1       TAPE READER ENABLE CODE
   13  87                    OUT 13B     OUTPUT 3 - ENABLE TAPE RDR.
   14  70  58   0            CAL TTYD1   TAPE RDR. CONTROL DELAY
   17   0             TTY    HLT         WAIT FOR TTY START PULSE
   18  70  65   0            CAL TTYD2   TTY DELAY - 4.468 MSEC.
   21 168                    XRA         TAPE RDR. DISABLE CODE
   22  87                    OUT 13B     OUTPUT 3 - DISABLE TP. RDR.
   23  65                    INP 0B      INPUT 0 - READ START PULSE
   24  22 255                LCI 255     COMPLEMENT TTY START PULSE
   26 170                    XRC         EXCLUSIVE-OR REG. C
   27  85                    OUT 12B     OUTPUT 2 - START PULSE OUT
   28  38 248                LEI 248     TTY DATA SAMPLING COUNTER
   30  70  58   0     TTYIN  CAL TTYD1   TTY DELAY - 9.012 MSEC.
   33  65                    INP 0B      READ TTY DATA INPUT
   34  22 255                LCI 255     COMPLEMENT TTY DATA
   36 170                    XRC
   37  85                    OUT 12B     OUTPUT 2 - TTY DATA OUT
   38  26                    RAR         STORE TTY DATA OUT
   39 193                    LAB         LOAD TTY DATA TO
   40  26                    RAR
   41 200                    LBA         LOAD AC TO REG. B
   42  32                    INE         E = E + 1
   43  72  30   0            JFZ TTYIN   JUMP IF ZERO F/F IS NOT SET
   46 193                    LAB         LOAD REG. B TO AC
   47  83                    OUT 11B     OUTPUT 1, TTY CHAR.
   48  20 128                SUI 128     REMOVE PARITY BIT
   50 200                    LBA         STORE TTY INPUT DATA
   51  70  58   0            CAL TTYD1
   54   6   1                LAI 1
   56  85                    OUT 12B     SUPPRESS TTY
   57   7                    RET
   58  30 115         TTYD1  LDI 115     9.012 MSEC. DELAY
   60  24             ST     IND         D = D + 1
   61  72  60   0            JFZ ST
   64   7                    RET
   65  30 186         TTYD2  LDI 186     4.468 MSEC. DELAY
   67  24             ST2    IND
   68  72  67   0            JFZ ST2
   71   7                    RET
   72 *30                    END
.END

 ALL DONE
```

Figure 4

REFERENCES

1. INTEL Corp., "MCS8 8008 8-bit Parallel Central Processor Unit," 55 pages, June 1972. This report describes the 8008 processor, processor timing, instruction set, controls signals, electrical specifications, etc. It is the basic reference if you are going to use the 8008.

2. INTEL Corp., "MCS8 8008 Assembler," 20 pages, June 1972. Section I, user's manual, is not applicable to the way the MCS8 assembler program is run at LLL. Section II describes the symbolic assembly language appears as Appendix A of this LER.

3. INTEL Corp., "MCS8 Bootstrap Loader Control Program," 19 pages, June 1972.

4. A. Cecil, H. Mill, and J. Rinde, "File Editing with TRIX," UCID-30040, 36 pages, March 1972. Copies are available from TID. Use page 35-selected commands from TRIX AC - as a guide when using TRIX.

5. "Introduction to OCTOPUS," CIC Manual I-002, October 1967. Pages 76-78 give a brief description of NAB. NAB is also to be described in Utility routine UR-204 when it is published.

Appendix A

1.0 GENERAL DESCRIPTION

The 8008 Assembler generates object programs from symbolic assembly language instructions. Programs are written in the assembly language using mnemonic symbols both for 8008 instruction and for special assembler operations. Symbolic addresses can be used in the source program; however, the assembled program will use absolute addresses.

1.1 Assembler Use and Operation

Source programs are written in assembly language and edited prior to assembling, using an editor program. Edited programs can then be assembled. The Assembler processes the source program in two passes or cycles.

The Assembler generates a symbol table from the source statement names in the first pass and checks for errors.

In the second pass the Assembler uses the symbol table and the source program to generate both a program listing and an absolute binary program. Error conditions are indicated in the program listing.

1.2 Symbol Usage

Symbols can represent specific addresses in memory for data and program words, or can be defined as constants. Symbols are used as labels for locations in the program or as data storage area labels or as constants.

Expressions can be formed from a symbol combined by plus or minus operators with other symbols or numbers to indicate a location other than that named by the symbol. Every symbol appearing as part of an operand must also appear as a statement label or else it is not defined and will be treated as an error. Symbols that are used as labels for two or more statements are also in error.

1.3 Absolute Addressing

Object programs use all absolute addresses. The starting address is specified by a pseudo instruction at the beginning of the source program. All subroutines referenced by symbol in the main program must be assembled as part of the main program. Subroutines not assembled with the main program must be referenced by their starting addresses.

1.4 Program Addresses

Consecutive memory addresses are generated by the Assembler program counter and assigned to each source statement. Two byte source statements are assigned two consecutive addresses and three byte source statements are assigned three consecutive addresses.

The starting address is set by an ORG pseudo instruction at the beginning of the source program.

1.5 Output Options

The Assembler output is stored in files and can be read out in several forms. Some of the options available are:

a. binary paper tape at the terminal (if your teletype is so equipped);
b. card output at computer center;
c. program listing at the terminal;
d. program listing at the computer center;
e. symbol table listing at the terminal;
f. symbol table listing at the computer center.

The printout of the program listing will have the following format:

Columns	
1- 5	Location (octal) of first byte of object code
6- 7	Blank
8- 10	First byte object code word in octal
11	Blank
12- 14	Second byte object code word in octal
15	Blank
16- 18	Third byte object work in octal
19	Blank
20- 22	Fourth byte object code word in octal
23- 24	Blank
25- 27	First 48 characters of source statement

2.0 INSTRUCTION FORMAT

The Intel Assembly program consists of a sequence of symbolic statements. Each source language statement contains a maximum of four fields in the following order:

location field;
operation field;
operand field;
comment field.

The format is essentially free field. Fields are delimited by one or more blanks. Blanks are interpreted as field separators in all cases, except in the comments field or in a literal character string.

The maximum length of any statement is 80 characters. The instruction must end prior to character 48 but the comments may extend to column 80.

2.1 Symbols

Symbols are used in the location field and in the operand field. A symbol is a sequence of one to six characters representing a value. The first character of any symbol must be an alphabetic. Symbols are comprised of the characters A through Z, and zero through nine.

The value of a symbol is determined by its use. In the location field of a machine instruction or a data definition, the value assigned to the symbol is the current value of the program counter. In the location field of an EQU pseudo instruction, the value of the operand field is assigned to the symbol.

An asterisk is a special purpose symbol. It represents the location of the first byte of the current instruction. Thus if an operand contains *-1, then the value calculated by the Assembler is one less than the location of the first byte of the current instruction.

Examples of legal symbols:

```
MAT   START2
MIKE  Z148
TED24 RONA3Z
*
```

2.2 Numeric Constants

Two types of numeric constants are recognized by the Assembler: decimal and octal. A decimal number is represented by one to five digits (0-9) within the range of 0 to 16383. An octal number contains from one to five digits (0-7) followed by the letter B. The range of octal numbers is 0 to 37777B.

Numeric constants can be positive or negative. Positive constants are preceded by a plus sign or no sign. Negative constants are preceded by a minus sign. There can be no blanks between the sign and the digits. If a minus sign precedes the number, then the complement of the binary equivalent is used.

2.3 Expressions

Expressions may occur in the operand field. The Assembler evaluates the expression from left to right and produces an absolute value for the object code. There can be symbols and numbers in expressions separated by arithmetic operator + and - Octal and decimal numbers are acceptable. No embedded blanks are allowed within expressions.

Parenthese are not permitted in an expression. Thus terms cannot be grouped as in the expression Z-(4+T). That expression must be written as Z-4-T to be acceptable to the Assembler.

2.4 Location Field

The location field of a statement contains a symbol when needed as a reference by other statements. If a statement is not referenced explicitly, then the location field may be blank.

The symbol must start in column 1 of the statement. That is, if a symbol is required it must be punched immediately following the end of statement mark of the preceding statement. The Assembler therefore assumes that if column 1 is blank, the location field of that statement does not contain a symbol.

Column 1 of the location field can also indicate that the entire line is a comment. If an asterisk occurs in column 1, then positions 2 through 80 contain remarks about the program. These remarks have no effect on the assembled program but do appear in the output listing.

2.5 Operation Field

The operation field must be present and is represented by a mnemonic code. The code describes a machine operation or an Assembler operation.

The operation code follows the location field and is seperated by one or more blanks from the location field. The operation field is terminated by a blank or an end of statement mark when there is no operand field and no comment field.

Examples of machine operations:

LAB	Load Register A with the contents of Register B
CPM	Compare contents of A register with contents of memory location m.

Example of Assembler operation:

ORG	Set program counter to specified origin.

2.6 Operand Field

The contents and significance of the operand field are dictated by the operation code. The operand field can contain the following:

blank
symbol
numeric
expression
data list.

The operand field follows the operation code and is separated from that code by one or more blanks. The operand is terminated by a blank or an end of statement mark if no comments follow the operand.

Examples of operands:

```
DANI      MIKE2-MIKE4+1
143B      773B+X2
1869      *-1
RON+33B   AA44-22B
(blank)
```

2.7 Comment Field

The comment field is optional. It follows the operand field and is seperated from that field by at least one blank. If there is no operand field for a given operation code, then the comment field follows the operation field. Once again at least one blank separates the operation code and the comments. Comments must terminate on or before the 80th character position. If the comment extends beyond that position, it will be truncated on the output listing. Comments up to the 48th character position are printed along with the source code. If comments are in positions 49 through 80, then they are printed on the next line.

3.0 MACHINE OPERATION

Each instruction in the 8008 repertoire can be represented by a three letter mnemonic in the 8008 assembly language. For each source statement in the assembly language (except for some pseudo instructions), the Assembler will generate one or more bytes of object code. Source language statements use the following notation:

Label - optional statement label;
Operand - one of the following:

data - a number, symbol or expression used to generate the second byte of an immediate instruction.

address - a number, symbol or expression used to generate the second and third bytes of a call or jump instruction.

device - a number, symbol or expression used to define input/output instructions to select specific devices.

Comment - optional comment.

(   ) - information enclosed in brackets is optional.

3.1 Move Statements-- 1 byte, or 2 bytes when operand is used.

Move instructions replace the contents of memory or of the A, B, C, D, E, H and L Registers with the contents of one of the Registers A, B, C, D, E, H or L or with the contents of the memory location specified by H and L or with an operand from the second byte of the instruction. In what follows, r_1 can represent A, B, C, D, E, H, L or M. r_2 can represent A, B, C, D, E, H, L, M or I. If $r_1 = M$, the contents of memory are replaced by the contents of r_2. If $r_2 = M$, the contents of r_1 are replaced by the contents of memory. If $r_2 = I$, the contents of r_1 are replaced by the operand from the second byte of the instruction.

(Label)	Lr_1r_2	data	(Comment)

Move r_2 to r_1.

Examples:

Label	LEH		Comment

Move H to E.

Label	LAM		Comment

Move A from memory.

Label	LMB		Comment

Move B to memory.

Label	LCI	062B	Comment

Load octal 062 into C.

Label	LMI	135B	Comment

Load octal 135 into memory.

The contents of the sending location are unchanged after each move. An operand is required if and only if $r_2 = I$.

3.2 Arithmetic and Logical Operation Statements-- 1 byte, or 2 bytes when operand is used.

These instructions perform arithmetic or logical operations between the contents of the A Register and the contents of one of the Registers B, C, D, E, H or L or the contents of a memory location specified by H, and L or an operand. The result is placed in the A Register. In what follows, r may be B, C, D, E, H or L, M or I. If $r = M$, memory location is specified. If $r = I$, the operand from the second byte of the instruction is specified.

3.2.1

(Label)	ADr	data	(Comment)

Add r to A.

3.2.2

(Label)	ACr	data	(Comment)

Add r to A with carry.

3.2.3

(Label)	SUr	data	(Comment)

Subract r from A.

3.2.4

(Label)	SBr	data	(Comment)

Subtract r from A with borrow.

3.2.5

(Label)	NDr	data	(Comment)

Logical AND r with A.

3.2.6

(Label)	XRr	data	(Comment)

Exclusive OR r with A.

3.2.7

(Label)	ORr	data	(Comment)

Inclusive OR r with A.

3.2.8

(Label)	CPr	data	(Comment)

Compare r with A.

Examples:

Label	ADB		Comment

Add B to A.

Label	SUM		Comment

Subtract the contents of the memory location specified by H and L from A.

Label	CPI	024B	Comment

Compare octal 024 with A.

An operand is required if and only if r = I.

3.3 Rotate Statements -- 1 byte

3.3.1

(Label)	RLC		(Comment)

Rotate A one bit left.

3.3.2

(Label)	RRC		(Comment)

Rotate A one bit right.

3.3.3

(Label)	RAL		(Comment)

Rotate A through the carry one bit left.

3.3.4

(Label)	RAR		(Comment)

Rotate A through the carry one bit right.

3.4 Call Statements -- 3 bytes

Call instructions are used to enter subroutines. The second and third bytes of call instructions are generated from source programs operands and are used to address the starting locations for the called subroutines. An operand is always required.

3.4.1 (Label) | CAL | address | (Comment)

Call subroutine unconditionally.

3.4.2 (Label) | CTC | address | (Comment)

Call subroutine if carry = 1.

3.4.3 (Label) | CFC | address | (Comment)

Call subroutine if carry = 0.

3.4.4 (Label) | CTZ | address | (Comment)

Call subroutine if accumulator = 0.

3.4.5 (Label) | CFZ | address | (Comment)

Call subroutine if accumulator ≠ 0.

3.4.6 (Label) | CTP | address | (Comment)

Call subroutine if accumulator parity is even.

3.4.7 (Label) | CFP | address | (Comment)

Call subroutine if accumulator parity is odd.

3.4.8 (Label) | CTS | address | (Comment)

Call subroutine if accumulator sign is minus.

3.4.9 (Label) | CFS | address | (Comment)

Call subroutine if accumulator sign in plus.

At the conclusion of each subroutine, control returns to the address "Label+3".

3.5 Jump Statements -- 3 bytes

Jump instructions are used to alter the normal program sequence. The second and third bytes of jump instructions are generated from source program operands and are used as the address of the next instruction. An operand is always required.

3.5.1

(Label)	JMP	address	(Comment)

Jump to address unconditionally.

3.5.2

(Label)	JTC	address	(Comment)

Jump to address if carry = 1.

3.5.3

(Label)	JFC	address	(Comment)

Jump to address if carry = 0.

3.5.4

(Label)	JTZ	address	(Comment)

Jump to address if accumulator = 0.

3.5.5

(Label)	JFZ	address	(Comment)

Jump to address if accumulator ≠ 0.

3.5.6

(Label)	JTP	address	(Comment)

Jump to address if accumulator parity is even.

3.5.7

(Label)	JFP	address	(Comment)

Jump to address if accumulator parity is odd.

3.5.8

(Label)	JTS	address	(Comment)

Jump to address if accumulator sign is minus.

3.5.9

(Label)	JFS	address	(Comment)

Jump to address if accumulator sign is plus.

3.6 Return Statements -- 1 byte

Return instructions are used at the end of subroutines to return control to the address following the call instruction that entered the subroutine. In what follows, assume a subroutine was called as shown:

MAIN	CAL	SUBRTN	Comment

3.6.1

(Label)	RET		(Comment)

Return unconditionally to "MAIN+3".

3.6.2

(Label)	RTC		(Comment)

Return to "MAIN+3" if carry = 1.

3.6.3

(Label)	RFC		(Comment)

Return to "MAIN+3" if carry = 0.

3.6.4

(Label)	RTZ		(Comment)

Return to "MAIN+3" if accumulator = 0.

3.6.5

(Label)	RFZ		(Comment)

Return to "MAIN+3" if accumulator ≠ 0.

3.6.6

(Label)	RTP		(Comment)

Return to "MAIN+3" if accumulator parity is even.

3.6.7

(Label)	RFP		(Comment)

Return to "MAIN+3" if accumulator parity is odd.

3.6.8

(Label)	RTS		(Comment)

Return to "MAIN+3" if accumulator sign is minus.

3.6.9

(Label)	RFS		(Comment)

Return to "MAIN+3" if accumulator sign is plus.

3.7 Input/Output Statements -- 1 byte

These instructions are used to input or output data, one byte at a time, between the A Register and the external device selected by the operand. An operand is always required.

3.7.1

(Label)	INP	device	(Comment)

Inputs one byte of data from device to the A Register.

3.7.2

(Label)	OUT	device	(Comment)

Outputs one byte of data from the A Register to device.

The device operand must have a value between 0 and 7 for input instructions and between 10 and 37 octal for output instructions.

3.8 Increment/Decrement Statements -- 1 byte

These instructions are used to increment by one or decrement by one of the registers r. In what follows, r can represent B, C, D, E, H or L. Increment and decrement operations affect the accumulator conditions zero, parity and sign, but not carry.

3.8.1

(Label)	INr		(Comment)

Add 1 to r.

3.8.2

(Label)	DCr		(Comment)

Subtract 1 from r.

Example:

(Label)	INB		(Comment)

Add 1 to B.

3.9 Halt Statement -- 1 byte

The halt instruction is used to stop the 8008 processor.

(Label)	HLT		(Comment)

3.10 Restart Statement -- 1 byte

The restart instruction is used in conjunction with an interrupt signal to start the 1201 after a halt. The program counter is set to a starting address equal to the operand multiplied by octal 10. A start operand is required which may have a value from 0 to 7.

(Label)	RST	start	(Comment)

3.11 Load Address Statement -- 4 bytes

This instruction is used to load H and L with a memory address and is simply an assembly language convention equivalent to the two separate instructions LHI and LLI. An operand is required.

(Label)	SHL	address	(Comment)

4.0 PSEUDO INSTRUCTIONS

The purpose of pseudo instructions is to direct the Assembler, to define constants used by the object code, and define values required by the Assembler. The following is a list of pseudo operations.

ASB	Define paper tape output.
ORG	Define origin of program.
EQU	Define symbol value for Assembler.
DEF	Define constants for object code.
DAD	Define two byte address.
END	Define End of source code.

4.1 Program Origin

The program origin can be defined by the user by an ORG pseudo operation. If no ORG statement is defined, the origin is assumed to be zero. The origin can be redefined whenever necessary by including an ORG statement prior to the section of code which starts at a specific program location.

The format of the ORG statement is:

	ORG	n	(Comments)

The operand n can be a number symbol, or an expression. If a symbol is used it must be predefined in the code.
Example of the ORG statement:

```
        LAB                                     Instruction starts in
                                                LOC 0000.

        LCD
         .
         .
         .
        ORG     1000B
SAM     LCD                                     Instruction stored in
                                                LOC 1000.
         .
         .
         .
        ORG     5000B
SALLY   DEF     1,4,77B,7000B                   Data starts in LOC 5000.
        END
```

4.2 Equate Symbol

A symbol can be given a value other than the one normally assigned by the program location counter by using the EQU pseudo operation. The symbol contained in the location field is given the value defined by the operand field.

The EQU statement does not produce a machine instruction or data word in the object code. It merely assigns a value to a symbol used in the source code.

Format of the EQU statement:

Symbol	EQU	operand	(Comment)

The operand may contain a numberic, a symbol, or an expression. Symbols which appear in the operand must be previously defined in the source code.

All fields are required except for the comment field, which is always optional.

Example of EQU statements:

```
        TELET   EQU     4
        MAGT2   EQU     2
        MAGT6   EQU     6
        SAM     EQU     1000B
                INP     TELET
                LAB
                CALL    SAM
                OUT     MAGT2
```

4.3 Define Constant

Constant data values can be defined using the DEF pseudo statement. The data values are placed in sequential words in the object code. If a symbol appears in the location field, it is associated with the first data word. That symbol can be then used to reference the defined data.

Format of the DEF statement:

(Symbol)	DEF	data list	(Comment)

The data list consists of one or more terms separated by commas. There can be no embedded blanks in the data list (except in a literal character string). The terms can be octal or decimal numerics, literal character strings, symbols or expressions.

A literal character string is enclosed in single quote marks ('). It can contain any ASCII characters, including blanks. The internal BCD 8 bit codes correspoinding to the given characters are stored in sequential bytes, one character per byte.

Octal and decimal numbers are stored one per byte in binary.
Octal numbers must be in the range 0 to 377B.
Decimal numbers must be in the range 0 to 255.
Two's complements are stored for minus numbers.

The program counter is incremented by one for each numberic term in the data string and by n for each literal string of n characters.

Examples of data strings:

```
MESS1  DEF      'SYMBOL TABLE OVERFLOWED', Y-2, SUB2
MESS2  DEF      'LITERAL STRING 1', 'LITERAL STRING2'
MASKS  DEF      77B, 177B, 130B, LABELS3, X+3 Required masks
       DEF      24,133,37B,99,232, 'ERROR' Required constants
```

4.4 Define Address

Program addresses, defined by alphabetic symbols, are stored as data by the DAD pseudo operation. The 16 bit address is stored in sequential bytes; the first byte contains the 8 least significant bits and the second byte contains the 8 most significant bits of the address.

Format of the DAD statement:

(Symbol)	DAD	data list	(Comment)

The data list consists of one or more symbols seperated by commas. There can be no embedded blanks in the data list.

The program counter is incremented by two for each symbol in the data list.

Examples of DAD statements:

```
LINK   DAD     SUB1,SUB2,SUB3
ERRSUB DAD     ERRORX     Print Errors
       DAD     SOCTAL,SPECM,SYMBOL,SEXPR,SLIT
```

4.5 End of Source

The end of the source code statements is defined with the END pseudo statement. The END operation code generates no object code; it merely signals to the Assembler that there is no more source code.

Format of the END statement:

```
______________________________________
               | END |         | (Comment)
```

Note that no symbol is allowed in the location field of the END statement.

4.6 Assembler paper tape output

The format of the paper tape output is defined by the ASB pseudo output. The operand specifies the format with the following mnemonic codes.

F1601 - 1601 format described in Intel Manual
SILI CON GATE MOS LSI ROM 1601, 1301

F8008 - F8008 Format
(this logic is not included in the Assembler but the position of the code is described in the PAPER Subroutine)

The entire 80 character statement is written on the paper tape file as the first record. It is used to describe the contents of the paper tape. If no ASB pseudo operation appears, then format F1601 is assumed and a string of asterisks appear on the paper tape file as the first record.

Examples of ASB statements:

```
ASB     F1601     Keyboard Code
ASB     F1601     Data Transmission Code
```

5.0 ERRORS

5.1 Various types of errors can be detected by the Assembler. Message is emitted following the statement which contains the error. The error messages and their meanings follow.

$ERROR$ ILLEGAL CHARACTER X
The special character X(such as $, / .,) appears in the statement (not in the comment) or perhaps a required operand field is missing.

$ERRORS$ MULTIPLY DEFINED SYMBOL XXXXXX
The symbol XXXXXX has been defined more than one time.

$ERROR$ UNDEFINED SYMBOL XXXXXX
The symbol XXXXXX has been used but never defined.

$ERROR$ ILLEGAL NUMBERIC CONTAINS CHARACTER X
An octal number includes an illegal digit (such as an 8 or 9) or the numberic contains non numeric characters

$ERROR$ ILLEGAL OPCODE XXX
The operation code XXX is not one of the acceptable mnemonics.

$ERROR$ MISSING OPERAND FIELD
No operand found for an operation code which requires one.

$ERROR$ ILLEGAL VALUE=YYYYYY,MAXIMUM=XXXXXX
The numberic value of an octal or decimal number of an expression has overflowed its limit.

XXXXXX = 377B	for 1 byte operands or data word
XXXXXX = 37777B	for 2 byte operands
XXXXXX = 37B	for output device numbers
XXXXXX = 7	for input device numbers
YYYYYY = given operand value	

$ERROR$ ILLEGAL SYMBOL
A location field contains a symbol that has more than six characters or that does not start with an alphabetic.

$ERROR$ MISSING LABEL
The label, which is required by the EQU pseudo operation, is missing.

$ERROR$ SYMBOL TABLE OVERFLOW, MAXIMUM=XXXXXX
Too many symbols in source program to fit into allocated symbol table.

$ERROR LINE OVERFLOW, MAXIMUM=XXXX
Input line exceeds 48 characters; or missing carriage return.

$ERROR$ ERRONEOUS LABEL
Opcodes END and ORG may not have a label

$ERROR$ ILLEGAL ORIGIN XXXXXX is less than XXXXXX
Value of new origin is less than current program count.

$ERROR$ ILLEGAL OPERAND
DAD opcode requires symbolic operand