MDP-8008 Assembler

User Manual

By
Ronald T. Kneusel

1993

The MDP-8008 assembler was designed to ease the task of creating programs for the MDP-80 simulator. This user manual describes the assembler and how to use it to create your own assembly language programs. Basic knowledge of programming at the machine or assembler level is assumed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the beginning there was machine code, and it was not good. Machine code is difficult to write, hard to modify, and nearly impossible to debug. Life was miserable and the power of the computer was locked in the hands of the elite few who could manipulate the machine at its own level. Enter assembly language. Assembly language is basically a substitution of (relatively) easy to remember mnemonics (alphanumeric codes that give some indication of the purpose of an instruction) for the difficult to remember numerical instruction codes. Unlike high level computer languages, there is a one-to-one correspondence between an assembly language instruction and an actual machine instruction. Not only are individual machine instructions difficult to remember but the numerical value for a particular instruction varies from one address mode to another. An assembler allows the programmer to specify both an instruction and its address mode in an easy to read form.

The assembler described below will be of primary use to you as you work through the tutorials and on to creating your own programs. This manual assumes a basic familiarity with assembly language programming. If this is your initial attempt at assembly language it will be helpful to work through the tutorial first.

1.1 Features

The MDP-8008 assembler is a fully functional, though simple, two pass assembler. Some of its features are listed below:

1. Free-form source code format
2. Unlimited source code size
3. Unlimited label length
4. Optional assembly listing and symbol table
5. Inclusion of useful pseudo instructions, ASC and HEX
6. Special address operators $>$ and $<$
7. Ability to include header files with standard equates

1.2 Restrictions

In order to preserve the simple nature of the assembler some restrictions apply. These are as follows:

1. No operand arithmetic
2. All numbers assumed to be hexadecimal
3. Only a single #ORG permitted per file
4. The special address operators $>$ and $<$ are restricted to labels only
5. All branches are assembled as absolute, not relative
6. All labels assemble as two bytes
7. Zero page instructions assembled only if address is a number

1.3 A Sample Session

The assembler does not contain a built in text editor. In order to create programs you will need some form of external text editor. If all else fails you can use *TeachText* which was included with your Macintosh. A standard word processor can be used provided the file is saved as a TEXT file. If you are using System 7 or Multifinder, you can run both the assembler and your text editor and quickly switch between the two. This creates a rapid development environment in which changes to the source program can be quickly assembled and errors corrected.

1.3.1 Creating 'Hello.ASM'

Using a text editor, enter the following sample assembly language program and save it as 'Hello.ASM'. This is the famous "Hello, world!" example.

```
;  Hello.world  -  Displays "Hello, world!"

#include std.equ    ; Standard monitor equates

#org  1000           ; Program begins at address 1000 (hexadecimal)

; Begin program

         lda >message              ; put high byte of MESSAGE in A
         ldx <message              ; put low byte of MESSAGE in X
         jsr _print                ; monitor routine to print string
         rts                       ; pointed to by A and X
.message asc 'Hello, world!'       ; here is the string
         hex 0d00                  ; 0d is return, 00 ends the string

; end
```

1.3.2 Assembling 'Hello.ASM'

Open the assembler application and select *Assemble File* from the *File* menu. A standard Macintosh open file dialog will appear. Use it to select the file *Hello.ASM*. The assembler will respond with another dialog box asking for a name to give to the assembled output file. Notice that the assembler has changed the .ASM suffix into .OBJ, this is the standard suffix for an assembled file. Other possible suffixes include .LST for an assembly listing and .SYM for a symbol table. Clicking the *Okay* button will cause the file to be assembled.

During the first pass the assembler locates the labels used by the program. In the second pass it generates the output file. A '.' is displayed for each line of the source program that is read. If there are no assembly errors it will report that assembly is complete and indicate the number of lines in the source file, as well as the starting and ending memory addresses. Errors are indicated by the corresponding line number in the source program. Assuming no errors, the program is assembled and the file Hello.OBJ is ready to be loaded into the simulator and run.

1.3.3 Modifying 'Hello.ASM'

Suppose we want to modify our program to print the text in the middle of a clear screen. To accomplish this we will need to use two new subroutines, one in the system monitor and the other of our own design. After the last line in the file include the following code:

```
.down    lda #0d                  ; the return character
.again   jsr _cout                ; monitor print character routine
         dey                      ; decrease count
         bne again                ; if not zero print again
         rts
```

This will print Y blank lines where Y is the current value of the Y register. We will call this routine with Y set to 10 after clearing the screen. After the begin program comment include the following:

```
        lda #0c                    ; 0C=12, clears the screen
        jsr _cout                  ; monitor routine
        ldy #0a                    ; load 10 into Y
        jsr down                   ; and call our routine
```

Lastly, we want to print 35 blank spaces before displaying the message. Fortunately, there exists a monitor routine to do this for us. After the call to the subroutine *down* enter the following:

```
        ldx #23                    ; X holds count
        jsr _blanks                ; print the blanks
```

The final modified program should appear as below:

```
;  Hello.world  -  Displays "Hello, world!"

#include std.equ    ; Standard monitor equates

#org  1000          ; Program begins at address 1000 (hexadecimal)

; Begin program

        lda #0c                    ; 0C=12, clears the screen
        jsr _cout                  ; monitor routine
        ldy #0a                    ; load 10 into Y
        jsr down                   ; and call our routine
        ldx #23                    ; X holds count
        jsr _blanks                ; print the blanks
        lda >message               ; put high byte of MESSAGE in A
        ldx <message               ; put low byte of MESSAGE in X
        jsr _print                 ; monitor routine to print string
        rts                        ; pointed to by A and X
.message asc 'Hello, world!'       ; here is the string
        hex 0d00                   ; 0d is return, 00 ends the string
.down   lda #0d                    ; the return character
.again  jsr _cout                  ; monitor print character routine
        dey                        ; decrease count
        bne again                  ; if not zero print again
```

```
        rts
; end
```

Using the same procedure as before, assemble the modified file replacing the earlier version of 'Hello.OBJ'. Load this version into the simulator and run it.

Chapter 2

The Assembler Program

The assembler is completely menu driven and is designed to be as simple as possible. There are three menus: File, Edit, and Options. The Edit menu is for desk accessories and is disabled.

2.1 The File Menu

The File menu is used to choose a file to assemble or to quit the application. The first menu item 'Assemble File...' will bring up a standard Macintosh dialog box allowing the user to select a source file to assemble. If the file selected ends with '.ASM' the assembler will strip it before appending any other extensions.

2.2 The Options Menu

Figure 2.1 shows the Options menu. The currently active options are indicated by a checkmark. The menu is divided into two sections, the first pertains to the assembly listing and the second to the symbol table. Initially, the assembler is set for no listing and no symbol table. Selecting any of the options activates it and places a checkmark next to it. Only one option can be active at a time.

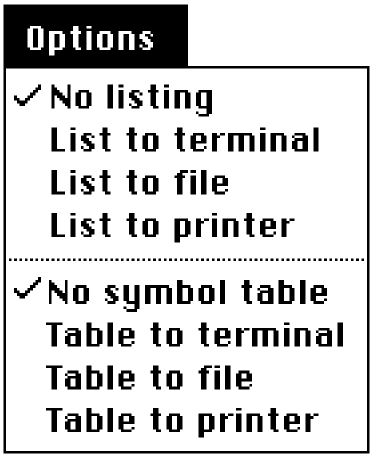

Figure 2.1: The Options menu.

2.2.1 The Assembly Listing

The assembly listing contains both the original source code and the machine code. An assembly listing for the file 'Hello.ASM' appears as,

```
 1-           ;  HELLO.WORLD  -  DISPLAYS "HELLO, WORLD!"
 2-
 3-           #INCLUDE STD.EQU    ; STANDARD MONITOR EQUATES
 4-
 5-           #ORG  1000          ; PROGRAM BEGINS AT ADDRESS 1000 (HEXADECIMAL)
 6-
 7-           ; BEGIN PROGRAM
 8-
 9-  1000: A9 0C       LDA #0C                  ; 0C=12, CLEARS THE SCREEN
10-  1002: 20 F8 76    JSR _COUT                ; MONITOR ROUTINE
11-  1005: A6 0A       LDY #0A                  ; LOAD 10 INTO Y
12-  1007: 20 10 26    JSR DOWN                 ; AND CALL OUR ROUTINE
13-  100A: A8 23       LDX #23                  ; X HOLDS COUNT
14-  100C: 20 FF D8    JSR _BLANKS              ; PRINT THE BLANKS
15-  100F: A9 10       LDA >MESSAGE             ; PUT HIGH BYTE OF MESSAGE IN A
16-  1011: A8 17       LDX <MESSAGE             ; PUT LOW BYTE OF MESSAGE IN X
17-  1013: 20 F8 D2    JSR _PRINT               ; MONITOR ROUTINE TO PRINT STRING
18-  1016: 60          RTS                      ; POINTED TO BY A AND X
19-  1017: 48656C6C6F2C20776F726C6421   .MESSAGE ASC 'Hello, world!'      ; HERE IS THE STRING
20-  1024: 0D00   HEX 0D00                 ; 0D IS RETURN, 00 ENDS THE STRING
21-  1026: A9 0D       .DOWN    LDA #0D                  ; THE RETURN CHARACTER
22-  1028: 20 F8 76    .AGAIN   JSR _COUT                ; MONITOR PRINT CHARACTER ROUTINE
23-  102B: C5          DEY                      ; DECREASE COUNT
24-  102C: D1 10 28    BNE AGAIN                ; IF NOT ZERO PRINT AGAIN
25-  102F: 60          RTS
26-
27-           ; END
28-
```

The line number is followed by the actual address and machine code that correspond to the instruction listed. Note that all assembler input is mapped to uppercase characters.

The options concerning the assembly listing are straightforward. The listing can be sent to the terminal (*List to terminal*), to a file on disk (*List to file*), or directly to the printer (*List to printer*). Listing directly to the printer is rather slow, so be patient.

If *List to file* is selected the assembler will create a file with the same name as the source file plus a '.LST' extension. This file is a standard TEXT file and has the format given above. You can read this file into any text editor, though the file will have the creator 'ALFA' and will look for the Shareware editor *Alpha* by Peter Keleher when double-clicked. *Alpha* is an excellent editor and is highly recommended as a companion to the assembler and simulator.

2.2.2 The Symbol Table

The symbol table is a list of the labels defined in an assembly language program. Labels are created in one of three ways: (1) by an #EQU statement, (2) by prefixing an instruction line with '.label' where 'label' is the label that will reference that line, or (3) by using an #INCLUDE to include a file of equates (such as the file STD.EQU which contains the equates for the system monitor). During the first pass of the assembler all labels are identified and the location they reference is determined. The second pass then uses this information to fill in the proper addresses for creating the output file.

The symbol table for the program 'Hello.ASM' appears as,

```
Symbol Table

[0200] KOUT         [0201] CSTB         [0202] KBDS         [0203] KEYB
[0100] STAK         [FFFD] COLD         [FFFA] BREAK        [0300] TEXT
```

```
[F800] _MON        [F830] VERSION    [F832] _MON1       [F851] _BRK
[F876] _COUT       [F87D] _KEYIN     [F884] _INPUT      [F889] _INPUT1
[F8D2] _PRINT      [F8D6] PRINT1     [F8E9] _PRINTHEX   [F920] _PRINTWORD
[F928] _UPPER      [F92C] _UPPER1    [F952] _KILLB      [F956] _KILLB1
[F989] _LENGTH     [F99B] _MAKEHEX   [F9C3] _MAKEBYTE   [F9DD] _NUMBER
[F9E1] _NUMBER1    [FA50] _ADD16     [FF7F] _REG        [FFD8] _BLANKS
[1017] MESSAGE     [1026] DOWN       [1028] AGAIN
```

Note that there are far more symbols present than the few defined in the program itself. This is due to the #INCLUDE STD.EQU statement which includes the symbols for the more important and useful system monitor routines. The assembler does not differentiate between #INCLUDEd symbols and symbols created in the program. The value in square brackets is the address associated with the symbol whose name follows. By convention, all system monitor subroutines have the '_' character as the first character in the symbol name. The individual subroutines are discussed in *The System Monitor via Assembly Language* chapter.

Symbol table options include sending the table to the terminal (*Table to terminal*), to a file (*Table to file*), or to the printer (*Table to printer*). As with the assembly listing, sending the table to the printer is rather slow.

If *Table to file* is selected and there is no assembly listing or the listing is not being sent to a file the assembler will prompt you for a file name. By default it will choose the name of the source file plus a '.SYM' extension. If both *List to file* and *Table to file* are selected the symbol table will be automatically appended to the end of the listing file and no '.SYM' file will be created.

Chapter 3

Assembler Instruction Format

An assembly language source file will contain several elements, some mandatory, others optional. These elements include:

1. Assembler directives
2. Labels (or symbols)
3. Mnemonics (instruction codes)
4. Operands
5. Comments and white space
6. Special forms

3.1 Assembler directives

The MDP-8008 assembler understands only three types of directives: #EQU, #ORG, and #INCLUDE. These are not instructions but are a way of providing the assembler with information about the source file. They typically appear near the beginning of a program before any actual instructions. This is not a requirement, however, since the assembler uses two passes the directives may appear anywhere without restriction.

3.1.1 #EQU

The #EQU (equate) directive may appear anywhere and is used to bind an address to a symbol. The format is:

```
#EQU  ‘label’  ‘address’
```

Where ‘label’ is the symbol being defined and ‘address’ is the address bound to ‘label’. The symbol must begin with a letter of the alphabet or the ‘_’ character, which, by convention, is reserved for system monitor routines. If a symbol is defined more than once an assembler warning error is generated and the initial binding is used. This usually results in garbage when the program is run. The address bound to the symbol must be a valid hexadecimal number and in the range 0000 to FFFF.

3.1.2 #ORG

The #ORG (origin) directive determines the address at which to begin assembling the program. The format is:

```
#ORG  ‘address’
```

Where ‘address’ must be a valid hexadecimal number in the range 0000 to FFFF. If there is more than one #ORG statement in a file the last one read will be used as the starting address. If no #ORG is specified the assembler defaults to the address 0400.

3.1.3 #INCLUDE

The #INCLUDE directive specifies the name of a file that may only contain comments, white space, and valid #EQU statements. The format is:

```
#INCLUDE  ‘file’
```

Where ‘file’ is the name of the file to be included. If no pathname is specified the file is assumed to be in the same folder as the assembler application. #INCLUDE is useful when a series of programs will be using the same equates. The assembler comes with the file STD.EQU which contains equates for the more useful system monitor routines. Any number of #INCLUDE statements can be in a file. The assembler will read all the files before proceeding further.

3.2 Labels

Much of the power of an assembler comes from its ability to use labels (symbols) in place of actual addresses. This frees the programmer to make alterations without concerning himself or herself with the effect on the rest of the program. For the MDP-8008 assembler a label is defined as: any string of characters whose first character is a letter of the alphabet (or '_') and does not contain any white space (spaces or tabs). There is no practical limit to the length of a label. In use, a label may be substituted anywhere an address would belong. When binding a label to an instruction precede it with a '.' (period):

```
            lda >string
.string     asc 'This is a string'
```

If the assembler encounters a multiply defined label it will use the value of the first binding and ignoring subsequent definitions.

3.3 Mnemonics

Mnemonics are the three letter labels used by the assembler to represent the actual microprocessor intructions. It is far easier to remember that LDA means to 'load the accumulator' than to remember the hexadecimal number A9. A complete outline of the instruction set for the MDP-8008 microprocessor is given in Appendix B. Many instructions permit multiple addressing modes which the assembler indicates in the operand field. All MDP-8008 mnemonics are three letters in length. In addition to the actual instructions the assembler recognizes two pseudo-mnemonics, ASC and HEX. These are used to assemble ASCII strings and hexadecimal data into a file. Their format is described in the section on *Special forms*.

3.4 Operands

The mnemonic determines the instruction while the operand (what the instruction operates on) determines the address mode. There are seven possible addressing modes for the MDP-8008 and two assembler pseudo modes, each with a different operand format. The modes are:

1. immediate
2. implied
3. absolute
4. absolute indexed
5. indirect absolute
6. zero page
7. relative
8. > and < (pseudo)

3.4.1 Immediate mode

Immediate mode instructions are two bytes in length and are indicated by a # as the first character of the operand field. For example,

```
.nothing    lda #A4     ; put the value A4 into the accumulator
```

Note that there is no space between the # and the data.

3.4.2 Implied mode

Implied mode is for single byte instructions that require no data. Indicate this to the assembler by using no operand field:

```
.label      tax         ; transfer the accumulator to X
```

3.4.3 Absolute mode

Absolute mode instructions are three bytes in length (two bytes for an address) and are indicated by a '\$' preceding the address value. If a symbol is being used do not use a '\$'. For example,

```
lda $1e39   ; load the value in $1e39 into A
sta data    ; and put it in the address bound to 'data'
```

3.4.4 Absolute Indexed mode

Functionally the same as the absolute mode with the addition of the characters ',x' at the end of the operand field. Example,

```
lda $1e39,x  ; load the value in $1e39+X into A
sta data,x   ; and put it in the address bound to 'data'+X
```

3.4.5 Indirect Absolute mode

This mode is indicated by enclosing the address in parentheses and appending ',x' at the end of the operand field. Again, numerical addresses must be prefixed by the '\$' character. Example,

```
lda ($1e39),x   ; indirect load of A
sta (data),x    ; store indirectly
```

3.4.6 Zero Page mode

The microprocessor can take advantage of the implied 00 in addresses below 0100. This is called the zero page and instructions using it require only two bytes instead of three. The assembler forces all labels to be two bytes so the only way to use zero page instructions is by giving a numerical address. Zero page is indicated by a '\$' as absolute mode is, but the address given is only two digits long:

```
lda $1e     ; load the value in $1e into A
```

3.4.7 Relative mode

The assembler does not directly support relative mode branching instructions. All branches are assembled as absolute references.

3.4.8 > and < pseudo modes

The special pseudo modes > and < must be used with a label, not a numerical address. The > before the label is translated by the assembler into an immediate mode instruction whose data byte is the most significant byte of the address of the label. The < translates into the least significant byte of the address. Examples,

```
            lda >string   ; load MSB of ‘string’ into A
            ldx <string   ; load LSB of ‘string’ into X
.string     asc ’a string’
```

3.5 Comments and White Space

Comments start with a semicolon and continue to the end of the line. Copious use of comments is strongly encouraged. White space is defined as blank spaces or tab characters. Blank lines are also permited. There must be at least one space or tab between the parts of a line, but leading or trailing white space is ignored.

3.6 Special Forms

The special forms used pertain to the ASC and HEX pseudo instructions. The ASC instruction expects a string of characters delimited by single quotes. The HEX instruction expects a series of hexadecimal bytes, each of two digits (include leading zero) and with no spaces between them. Examples for each are,

```
            asc ’this is a string’  ; use single quotes
            hex 000102030405060708  ; no spaces allowed
```

Both ASC and HEX are very useful for placing character data into a program.

Chapter 4

The System Monitor via Assembly Language

The simulator's built in system monitor program occupies the 2K of memory from F800 to FFFF and contains the routines necessary to make it possible to enter, execute, and modify machine language programs. It also contains a number of useful routines for printing strings and numbers, entering data, and converting strings to numbers. The file STD.EQU which is included with the assembler contains equates to many of these subroutines. #INCLUDEing this file at the beginning of your programs will make these routines available to you. This chapter will discuss many of these routines, giving examples and noting side-effects. Table 4.1 shows the subroutines that will be covered.

_COUT	F876	Output character in A

Description: Displays the character whose ASCII code is stored in the accumulator.
Registers altered: None.
Example:

```
lda #41     ; ASCII code for 'A'
jsr _cout   ; print the letter
```

Table 4.1: Some useful system monitor subroutines.

Name	Location	Input	Output	Use
_COUT	F876	character in A	none	Prints a character
_KEYIN	F87D	none	character in A	Read keyboard
_INPUT	F884	none	text starts 0300, X length	Input line, prompt
_INPUT1	F889	none	text starts 0300, X length	Input line, no prompt
_PRINT	F8D2	AX holds string start	none	Print a string
_PRINTHEX	F8E9	number in A	none	Print A as a number
_PRINTWORD	F920	number in AX	none	Print AX as 16-bit number
_UPPER	F928	AX holds start	none	Convert string to uppercase
_KILLB	F952	AX holds start	none	Remove blanks from a line
_LENGTH	F989	06 & 07 hold start	X length	Length of string
_NUMBER	F9DD	AX holds start	01 & 02	String to number (16-bit)
_REG	FF7F	none	none	Display current register values
_BLANKS	FFD8	number in X	none	Output X blanks

_KEYIN	F87D	Input character from keyboard

Description: Waits for a key to be pressed and places the ASCII value in the accumulator.
Registers altered: A.
Example:

```
jsr _keyin    ; get a character
jsr _cout     ; and echo it back
```

_INPUT	F87D	Input a line of text, with prompt
_INPUT1	F889	Input a line of text, no prompt

Description: Inputs a line of text. Characters are stored starting at location 0300. Backspace and delete keys work as normal. X holds the length on exit.
Registers altered: A, X.
Example:

```
jsr _input      ; get a line of text
lda 0300        ; check first character
cmp #41         ; compare to 'A'
beq Acommand    ; if so, then branch
```

_PRINT	F8D2	Print a null terminated string

Description: Displays the null terminated string the address of which is in A (MSB) and X (LSB).
Registers altered: A, X.
Example:

```
        lda >string     ; get MSB of address of string
        ldx <string     ; get LSB of address of string
        jsr _print      ; print the string
.string asc 'a string'  ; a string
        hex 00          ; null terminates string
```

_PRINTHEX	F8E9	Print A as a hexadecimal number

Description: Display the accumulator as a two digit hexadecimal number.
Registers altered: A.
Example:

```
        lda #2D         ; load a number
        jsr _printHex   ; displays '2D' on the screen
```

_PRINTWORD	F920	Print AX as a four digit hexadecimal number

Description: Displays AX as a four digit number. A holds the high part, X holds the low.
Registers altered: A, X.
Example:

```
        lda #33         ; load the high part
        ldx #44         ; load the low part
        jsr _printWord  ; outputs '3344'
```

_UPPER	F928	Convert a string to uppercase

Description: Converts the null terminated string starting at the address in AX to uppercase.

Registers altered: A, X.
Example:

```
        lda >string     ; get MSB of address of string
        ldx <string     ; get LSB of address of string
        jsr _upper      ; make it uppercase
        lda >string
        ldx <string
        jsr _print      ; print the string giving ‘A STRING’
.string asc ’a string’  ; a string
        hex 00          ; null terminates string
```

_KILLB	F952	Remove the blanks from a string

Description: Removes the blanks from the null terminated string that starts at the address in AX.
Registers altered: A, X.
Example:

```
        lda >string     ; get MSB of address of string
        ldx <string     ; get LSB of address of string
        jsr _killB      ; remove blanks
        lda >string
        ldx <string
        jsr _print      ; outputs ’astring’
.string asc ’a string’  ; a string
        hex 00          ; null terminates string
```

_LENGTH	F989	Find the length of a string

Description: Returns in X the length of the null terminated string pointed to by 06 and 07.
Registers altered: A, X.
Example:

```
        lda >string     ; get MSB of address of string
        ldx <string     ; get LSB of address of string
        jsr _length     ; get length
        txa             ; put in A
        jsr _printHex   ; outputs ‘08’
.string asc ’a string’  ; a string
        hex 00          ; null terminates string
```

_NUMBER	F9DD	Convert a string to a number

Description: Converts the null terminated string in AX into a 16-bit hexadecimal number stored in 01 and 02.
Registers altered: A, X, and Y.
Example:

```
#equ text 0300   ; text input buffer
 jsr _input      ; get a number
 lda >text
 ldx <text
 jsr _number     ; convert it
 lda $01
 ldx $02
 jsr _printWord  ; display it
```

_REG	FF7F	Display current register values

Description: Displays the current register values.
Registers altered: None.
Example:

```
 jsr _reg        ; output registers
```

_BLANKS	FFD8	Output blank spaces

Description: Displays blank spaces, count in X.
Registers altered: A, X.
Example:

```
 ldx #0D         ; set up for 13 blanks
 jsr _blanks     ; and print them
```

Appendix A

Glossary

Accumulator is the register that the microprocessor uses for math and logic functions.

Address mode refers to the way in which a particular microprocessor instruction uses the computer's memory. The MDP-8008 has seven possible address modes though not all modes are available to each instruction.

ASCII stands for American Standard Code for Information Interchange. It is a method for coding the alphabet and other symbols in a machine usable form. The letter 'A' has an ASCII value of 65.

Assembler translates programs written using mnemonics and labels into pure machine code for use by the microprocessor.

Bit is the smallest unit of information a computer can use. It is a single binary digit, 0 or 1, *on* or *off*.

Byte is a series of 8 bits and is the smallest unit of memory for most computers. A single ASCII character occupies one byte of memory.

Equate binds a symbol to an address. It is similar to defining a constant in other programming languages. For example, *#equ start 0400* will bind the symbol 'start' to the address 0400.

Hexadecimal numbers are base 16. Digits range from 0 through 9 and A through F with $A = 10$ and $F = 15$. Most machine code programming is done in hexadecimal. The MDP-8008 assembler assumes all numbers to be hexadecimal.

Include file is a file containing equates that may be included in an assembly language program. The file STD.EQU is included with the assembler and contains equates for useful monitor subroutines.

Label is a character string associated with a machine address. It is identical in meaning to *symbol*.

List file holds the assembly listing. The listing gives the assembled code along with the original source and optionally includes the symbol table.

Machine code is the native tongue of a microprocessor. It is purely numerical and is generally represented as hexadecimal numbers.

MDP-80 is the name of the microcomputer simulator that runs programs written in MDP-8008 machine code.

MDP-8008 is the microprocessor for which the assembler generates machine code. It is the heart of the MDP-80 microcomputer simulator.

Monitor is the program burned into the ROM of a microcomputer. It lets the user modify memory and run machine language programs.

Object file is the output from the assembler. It contains machine code in a format that is usable by the simulator.

Pseudo instructions are instructions that while not part of the actual microprocessor instruction set are simulated by the assembler. The ASC and HEX instructions are pseudo instructions.

Register is a single byte storage space on the microprocessor. The MDP-8008 has three user registers, the accumulator (A), the X, and Y.

Simulator is the program that simulates the operations of a small microcomputer.

Source file is the file that contains the assembly language program. It is used by the assembler to generate the object file.

Symbol see **label** above.

Symbol table is a list of all of the symbols used in a particular program and the address bound to that symbol.

Word is the standard unit used by a particular microprocessor. An 8-bit microprocessor uses one byte words. A 32-bit microprocessor uses 4 byte words. In the assembler and simulator, a word generally refers to a 16-bit, or 2 byte, value.

Appendix B

MDP-8008 Reference

The MDP-8008 microprocessor has 52 instructions and 7 addressing modes. Table B.1 gives a summary of the MDP-8008 instruction set. The instruction names are given in Table'B.2. The second column of Table B.1 indicates with an '*' which of the processor status flags are affected by each instruction. The flags are (N) negative (bit seven is set), (C) carry, (Z) zero, (L) less than, and (G) greater than.

B.1 MDP-8008 addressing modes

An instruction's addressing mode determines the way in which the instruction will access and use memory. The seven modes supported by the MDP-8008 are implied, immediate, absolute, absolute indexed, zero page, relative, and indirect indexed.

Implied addressing

The implied address in implied addressing is the accumulator. Implied mode is used by single byte instructions that only act on the accumulator and do not reference memory.

Immediate addressing

The value used in immediate mode addressing is the next byte of the program, or the first operand. The target of the instruction is one of the microprocessor's registers. For example, `LDA #FF` loads the accumulator with the immediate value `FF`.

Absolute addressing

An 8-bit microprocessor like the MDP-8008 can access a total of 65536 memory locations. It therefore takes two bytes to represent the address of any memory location. Absolute addressing uses two operand bytes (the first two bytes of the program following the byte containing the opcode) to represent the address of a memory location. For example, `STA $1E37` will store the contents of the accumulator in memory location `1E37`.

Absolute indexed addressing

Similar to absolute addressing, absolute indexed addressing uses as a target address the value obtained when the address represented by the operand bytes is added to the current value of the X register. If the X register contains `14` and the instruction is `STA $1E38,X` the contents of the accumulator will be stored in location `1E38` + `14` = `1E4C`. Absolute indexed addressing is particularly useful when dealing with lists.

Indirect indexed addressing

The target address in indirect index addressing is found by taking the absolute address in the operands, reading the two byte address that starts at the given address and adding the contents of the X register to get the final address. For example, suppose the instruction being executed (in assembly form) is `LDA ($2E33),X`. If the values stored in memory locations `2E33` and `2E34` are `14` and `1C` and the current value of the X register is `A4` then the accumulator will be loaded with the contents of location `141C` + `A4` = `14C0`.

Zero page addressing

Zero page addressing takes advantage of the fact that the first 256 memory locations can be addressed in a single byte (`00` to `FF`). Therefore, all instructions using the zero page are only two bytes long, which can add up to a significant savings if frequently used locations are in the zero page.

Relative addressing

Only the branch instructions use relative addressing. Instead of specifying an absolute location in memory, a relative address is a positive or negative offset from the current address. The use of relative addressing allows one to write code that will run at any location in memory. Relative addressing uses one operand and as a result can reference locations that are -128 to +127 bytes from the current location. Tables B.3 and B.4 can be used to determine the appropriate operand for a specific branch. The MDP-8008 assembler does not support relative addressing.

Table B.1: Summary of MDP-8008 instruction set.

Mnemonic	N-C-ZLG-	Implied	Immediate	Absolute	Abs,X	Zero	Relative	Indirect
ADC	*-*-*---		A5	B5	BE			
AND	----*---		A0	99				
BCC	--------			D5			D4	
BCS	--------			DB			DA	
BEQ	--------			D3			D2	
BGT	--------			D9			D8	
BLT	--------			D7			D6	
BNE	--------			D1			D0	
BNG	--------			DD			DC	
BPL	--------			DF			DE	
BRA	--------						E0	
BRK	--------	00						
CLC	--*-----	54						
CMP	----***-		88	89	8A	CB		
CPX	----***-		86	85		CC		
CPY	----***-		8B	8C		CD		
DCA	----*---	AF						
DEC	----*---			45	34			
DEX	----*---	C7						
DEY	----*---	C5						
EOR	----*---		A2	97				
INA	----*---	AE						
INC	----*---			46	35			
INX	----*---	E8						
INY	----*---	EE						
JMP	--------			4C				
JSR	--------			20				
LDA	*---*---		A9	AA	AD	C0		61
LDX	*---*---		A8	AB		C2		
LDY	*---*---		A6	AC		C4		
LSH	--*-----	26						
NOP	--------	EA						
ORA	----*---		A1	98				
PHA	--------	3C						
PHP	--------	33						
PLA	--------	3D						
PLP	--------	36						
ROL	--*-----	28						
ROR	--*-----	27						
RSH	--*-----	29						
RTS	--------	60						
SBC	*-*-*---		A4	BB	BA			
SEC	--*-----	55						
STA	--------			B9	BD	C6		63
STX	--------			B8		C8		
STY	--------			B6		CA		
TAX	--------	14						
TAY	--------	13						
TSX	--------	07						
TXA	--------	12						
TXS	--------	10						
TYA	--------	11						

Table B.2: MDP-8008 instruction names.

Mnemonic	Function
ADC	Add to accumulator with carry
AND	AND accumulator with memory
BCC	Branch on carry clear
BCS	Branch on carry set
BEQ	Branch on equal or zero
BGT	Branch on greater than
BLT	Branch on less than
BNE	Branch on not equal or not zero
BNG	Branch on negative set
BPL	Branch on positive (not negative)
BRA	Unconditional relative branch
BRK	Break
CLC	Clear carry flag
CMP	Compare accumulator to memory
CPX	Compare X register to memory
CPY	Compare Y register to memory
DCA	Decrement accumulator by one
DEC	Decrement memory by one
DEX	Decrement X register by one
DEY	Decrement Y register by one
EOR	Exlusive-OR accumulator with memory
INA	Increment accumulator by one
INC	Increment memory by one
INX	Increment X register by one
INY	Increment Y register by one
JMP	Jump to address
JSR	Jump to subroutine at address
LDA	Load the accumulator with memory
LDX	Load the X register with memory
LDY	Load the Y register with memory
LSH	Bit shift accumulator left, no carry
NOP	No operation
ORA	OR accumulator with memory
PHA	Push accumulator on stack
PHP	Push processor status on stack
PLA	Pull accumulator from stack
PLP	Pull processor status from stack
ROL	Rotate accumulator to the left with carry
ROR	Rotate accumulator to the right with carry
RSH	Bit shift accumulator right, no carry
RTS	Return from subroutine
SBC	Subtract memory from accumulator with carry
SEC	Set carry flag
STA	Store accumulator in memory at address
STX	Store X register in memory at address
STY	Store Y register in memory at address
TAX	Transfer accumulator to X register
TAY	Transfer accumulator to Y register
TSX	Transfer stack pointer to X register
TXA	Transfer X register to accumulator
TXS	Transfer X register to stack pointer
TYA	Transfer Y register to accumulator

Table B.3: Forward relative branch table. Find the number of bytes forward by locating the value in the body and reading the most significant and least significant hexadecimal digits from the row and column headings respectively. Example, to branch forward 73 bytes, locate 73 in the body of the table and choose 4 from the row and 9 from the column to make 49.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Table B.4: Backward relative branch table. Find the number of bytes backwards by locating the value in the body and reading the most significant and least significant hexadecimal digits from the row and column headings respectively. Example, to branch backwards 73 bytes, locate 73 in the body of the table and choose B from the row and 7 from the column to make B7.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table B.5: 7-bit ASCII codes.

Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
ctl-@ (NUL)	00	00	space	20	32	@	40	64	sp	60	96
ctl-A (SOH)	01	01	!	21	33	A	41	65	a	61	97
ctl-B (STX)	02	02	"	22	34	B	42	66	b	62	98
ctl-C (ETX)	03	03	#	23	35	C	43	67	c	63	99
ctl-D (EOT)	04	04	$	24	36	D	44	68	d	64	100
ctl-E (ENQ)	05	05	%	25	37	E	45	69	e	65	101
ctl-F (ACK)	06	06	&	26	38	F	46	70	f	66	102
ctl-G (BEL)	07	07	’	27	39	G	47	71	g	67	103
ctl-H (BS)	08	08	(	28	40	H	48	72	h	68	104
ctl-I (HT)	09	09	)	29	41	I	49	73	i	69	105
ctl-J (LF)	0A	10	*	2A	42	J	4A	74	j	6A	106
ctl-K (VT)	0B	11	+	2B	43	K	4B	75	k	6B	107
ctl-L (FF)	0C	12	,	2C	44	L	4C	76	l	6C	108
ctl-M (CR)	0D	13	-	2D	45	M	4D	77	m	6D	109
ctl-N (SO)	0E	14	.	2E	46	N	4E	78	n	6E	110
ctl-O (SI)	0F	15	/	2F	47	O	4F	79	o	6F	111
ctl-P (DLE)	10	16	0	30	48	P	50	80	p	70	112
ctl-Q (DC1)	11	17	1	31	49	Q	51	81	q	71	113
ctl-R (DC2)	12	18	2	32	50	R	52	82	r	72	114
ctl-S (DC3)	13	19	3	33	51	S	53	83	s	73	115
ctl-T (DC4)	14	20	4	34	52	T	54	84	t	74	116
ctl-U (NAK)	15	21	5	35	53	U	55	85	u	75	117
ctl-V (SYN)	16	22	6	36	54	V	56	86	v	76	118
ctl-W (ETB)	17	23	7	37	55	W	57	87	w	77	119
ctl-X (CAN)	18	24	8	38	56	X	58	88	x	78	120
ctl-Y (EM)	19	25	9	39	57	Y	59	89	y	79	121
ctl-Z (SUB)	1A	26	:	3A	58	Z	5A	90	z	7A	122
ctl-[(ESC)	1B	27	;	3B	59	[	5B	91	{	7B	123
ctl-\ (FS)	1C	28	<	3C	60	\	5C	92	|	7C	124
ctl-] (GS)	1D	29	=	3D	61	]	5D	93	}	7D	125
ctl-$\uparrow$ (RS)	1E	30	>	3E	62	$\uparrow$	5E	94	˜	7E	126
ctl-$\leftarrow$ (VS)	1F	31	?	3F	63	$\leftarrow$	5F	95	DEL	7F	127

Appendix C

System Monitor Source Code

```
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;                                            ;
            ; MDP-80 SYSTEM MONITOR , VERSION 1.0, RTK.  ;
            ; ------------------------------------------ ;
            ;                                            ;
            ; LAST MOD:  11.10.93                        ;
            ;                                            ;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

            ;
            ;  TO BE ADDED: LIST TO HANDLE RELATIVE INSTRUCTIONS PROPERLY
            ;


            #ORG F800      ; START OF MONITOR


            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;                                                        ;
            ;                   MONITOR EQUATES                      ;
            ;                                                        ;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


            #EQU  KOUT   0200 ; CHARACTER OUT
            #EQU  CSTB   0201 ; CHARACTER OUT STROBE
            #EQU  KBDS   0202 ; KEYBOARD STROBE
            #EQU  KEYB   0203 ; KEYBOARD CHARACTER
            #EQU  STAK   0100 ; START OF STACK
            #EQU  COLD   FFFD ; CPU COLD START
            #EQU  BREAK  FFFA           ; BRK VECTOR
            #EQU  TEXT   0300 ; TEXT INPUT BUFFER
            #EQU  ZERO   0000 ; ENTRANCE TO EXTERNAL ROUTINES


            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;                                                        ;
            ;                 MAIN MONITOR ROUTINE                   ;
            ;                                                        ;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

            ; MONITOR COLD START - PRINTS STARTUP MESSAGE

F800: A9 0C       ._MON LDA #0C ; CLEAR THE SCREEN
F802: 20 F8 76    JSR _COUT
F805: A9 F8       LDA >MESSAGE ; GET HIGH BYTE
F807: A8 0F       LDX <MESSAGE ; AND LOW BYTE
```

```
F809: 20 F8 D2    JSR _PRINT ; PRINT THE TEXT
F80C: 4C F8 32    JMP _MON1 ; SKIP OVER TEXT
F80F: 4D44502D3830204D6F6E69746F7220524F4D   .MESSAGE ASC 'MDP-80 Monitor ROM' ; STARTUP MESSAGE
F821: 2076657273696F6E20312E30   ASC ' version 1.0'
F82D: 0D0D00   HEX 0D0D00 ; NULL IS END-OF-STRING MARKER
F830: 0100   .VERSION    HEX 0100                    ; MONITOR VERSION - 01.00

            ; MONITOR WARM START - INPUT A LINE AND EXECUTE IT

F832: A9 20       ._MON1 LDA #20 ; SET UP GO VECTOR
F834: C6 00       STA $00
F836: A9 60       LDA #60
F838: C6 03       STA $03
F83A: 20 F8 84    JSR _INPUT ; PRINT PROMPT & GET LINE
F83D: A9 03       LDA >TEXT
F83F: A8 00       LDX <TEXT
F841: 20 F9 28    JSR _UPPER ; MAKE LINE UPPERCASE
F844: A9 03       LDA >TEXT
F846: A8 00       LDX <TEXT
F848: 20 F9 52    JSR _KILLB ; REMOVE BLANKS
F84B: 20 FA 60    JSR _EXECUTE ; EXECUTE LINE
F84E: 4C F8 32    JMP _MON1 ; AND GET THE NEXT COMMAND

            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;                                                         ;
            ;                 SUBROUTINE DEFINITIONS                  ;
            ;                                                         ;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

            ;
            ; _BRK OUTPUT REGISTER VALUES AND JUMP TO _MON1
            ;

F851: 20 FF 7F    ._BRK JSR _REG
F854: A9 F8       LDA >STRNG
F856: A8 5E       LDX <STRNG
F858: 20 F8 D2    JSR _PRINT
F85B: 4C F8 67    JMP NXTBRK
F85E: 427265616B204020   .STRNG ASC 'Break @ '
F866: 00   HEX 00
F867: C0 01       .NXTBRK LDA $01
F869: C2 02       LDX $02
F86B: 20 F9 20    JSR _PRINTWORD
F86E: A9 0D       LDA #0D
F870: 20 F8 76    JSR _COUT
F873: 4C F8 32    JMP _MON1


            ;
            ; _COUT PRINT CHARACTER IN A
            ;

F876: B9 02 00    ._COUT STA KOUT ; STORE A IN KOUT
F879: 46 02 01    INC CSTB ; OUTPUT CHARACTER
F87C: 60          RTS

            ;
            ; _KEYIN GET A CHARACTER FROM THE KEYBOARD
            ;

F87D: 46 02 02    ._KEYIN INC KBDS ; SET KEYBOARD IN STROBE
F880: AA 02 03    LDA KEYB ; PUT KEY INTO A
F883: 60          RTS

            ;
            ; _INPUT INPUT A LINE OF TEXT TO $0300 (TEXT).  X HOLDS LENGTH ON EXIT.
            ;

```

```
F884: A9 2A      ._INPUT LDA #2A ; * PROMPT
F886: 20 F8 76   JSR _COUT
F889: A8 00      ._INPUT1 LDX #00 ; CLEAR LENGTH COUNTER
F88B: 20 F8 7D   .LOOP JSR _KEYIN ; GET A CHARACTER
F88E: 20 F8 76   JSR _COUT ; AND ECHO IT
F891: BD 03 00   STA TEXT,X ; PUT IN BUFFER
F894: 88 0D      CMP #0D ; IS IT RETURN?
F896: D3 F8 BD   BEQ DONE ; YES, EXIT
F899: 88 08      CMP #08 ; IS IT BACKSPACE?
F89B: D3 F8 AA   BEQ DEL ; YES, DELETE CHARACTER
F89E: 88 7F      CMP #7F ; IS IT DELETE?
F8A0: D3 F8 AA   BEQ DEL ; YES, DELETE CHARACTER
F8A3: E8         INX ; READY FOR NEXT CHARACTER
F8A4: D3 F8 BD   BEQ DONE ; IS X>FF? IF SO THEN DONE
F8A7: 4C F8 8B   JMP LOOP ; GET ANOTHER CHARACTER
F8AA: 86 00      .DEL CPX #00 ; ANYTHING TO DELETE?
F8AC: D3 F8 8B   BEQ LOOP ; NO
F8AF: C7         DEX ; DELETE CHAR IN BUFFER
F8B0: A9 20      LDA #20      ; PRINT A BLANK
F8B2: 20 F8 76   JSR _COUT ; TO CLEAR DELETED CHARACTER
F8B5: A9 08      LDA #08 ; AND BACK UP
F8B7: 20 F8 76   JSR _COUT ; AGAIN
F8BA: 4C F8 8B   JMP LOOP ; GET MORE CHARACTERS
F8BD: 86 00      .DONE CPX #00 ; NEED TO ADJUST LENGTH?
F8BF: D3 F8 CA   BEQ NOCHAR ; NO CHARACTERS
F8C2: A9 00      LDA #00 ; PUT A 0 IN A
F8C4: BD 03 00   STA TEXT,X ; AND STORE AFTER STRING
F8C7: 4C F8 D0   JMP FINISH ; AND FINISH
F8CA: B8 03 00   .NOCHAR STX TEXT ; SPECIAL CASE
F8CD: 4C F8 D1   JMP END ; OF NO CHARACTERS
F8D0: C7         .FINISH DEX ; ADJUST LENGTH
F8D1: 60         .END RTS

           ;
           ; _PRINT PRINT A NULL TERMINATED STRING STARTING AT A (HIGH) & X (LOW)
           ;

F8D2: C6 06      ._PRINT STA $06 ; PUT IN STRING ADDRESS IN SCRATCH
F8D4: C8 07      STX $07 ; LOW BYTE TOO
F8D6: A8 00      .PRINT1 LDX #00 ; CLEAR COUNTER AND PRINT
F8D8: 61 00 06   .LOOP2 LDA ($0006),X ; GET A CHARACTER
F8DB: D3 F8 E8   BEQ DONE2 ; NULL SO EXIT
F8DE: 20 F8 76   JSR _COUT ; PRINT CHARACTER
F8E1: E8         INX ; MOVE TO NEXT CHARACTER IN STRING
F8E2: D3 F8 E8   BEQ DONE2 ; PRINTED 255, EXIT
F8E5: 4C F8 D8   JMP LOOP2 ; PRINT ANOTHER CHARACTER
F8E8: 60         .DONE2 RTS

           ;
           ;  _PRINTHEX OUTPUT A AS A HEXADECIMAL NUMBER, A IS UNCHANGED
           ;

F8E9: C6 06      ._PRINTHEX STA $06      ; MONITOR SCRATCH LOCATION
F8EB: A0 F0      AND #F0 ; MASK LSN
F8ED: 29         RSH ; AND MOVE MSN TO LSN
F8EE: 29         RSH
F8EF: 29         RSH
F8F0: 29         RSH
F8F1: 88 09      CMP #09       ; GREATER THAN 9?
F8F3: D9 F8 FF   BGT HEX1 ; YES, OUTPUT LETTER
F8F6: 54         CLC ; NO, ADD $30 TO GET
F8F7: A5 30      ADC #30 ; ASCII FOR DIGIT
F8F9: 20 F8 76   JSR _COUT ; PRINT FIRST DIGIT
F8FC: 4C F9 05   JMP NEXT ; GO TO NEXT DIGIT
F8FF: 54         .HEX1 CLC          ; ADD $37 TO
F900: A5 37      ADC #37      ; GET LETTER
F902: 20 F8 76   JSR _COUT ; AND PRINT DIGIT
```

```
F905: C0 06       .NEXT LDA $06 ; GET ORIGINAL FROM SCRATCH
F907: A0 0F       AND #0F ; KILL UPPER BITS
F909: 88 09       CMP #09      ; GREATER THAN 9?
F90B: D9 F9 17    BGT HEX2 ; YES, OUTPUT LETTER
F90E: 54          CLC ; NO, ADD $30 TO GET
F90F: A5 30       ADC #30 ; ASCII FOR DIGIT
F911: 20 F8 76    JSR _COUT ; PRINT SECOND DIGIT
F914: 4C F9 1D    JMP END2 ; GO TO NEXT DIGIT
F917: 54          .HEX2 CLC         ; ADD $37 TO
F918: A5 37       ADC #37       ; GET LETTER
F91A: 20 F8 76    JSR _COUT ; AND PRINT DIGIT
F91D: C0 06       .END2 LDA $06 ; RECOVER INITIAL A VALUE
F91F: 60          RTS ; EXIT

            ;
            ;  _PRINTWORD  OUTPUTS A & X AS 16-BIT NUMBER - DESTROYS A
            ;

F920: 20 F8 E9    ._PRINTWORD JSR _PRINTHEX ; PRINT A
F923: 12          TXA ; PUT X IN A
F924: 20 F8 E9    JSR _PRINTHEX ; PRINT IT
F927: 60          RTS ; EXIT

            ;
            ;  _UPPER   MAKES THE NULL TERMINATED STRING IN AX UPPERCASE
            ;

F928: C6 06       ._UPPER STA $06 ; PUT IN SCRATCH SPACE
F92A: C8 07       STX $07
F92C: A8 00       ._UPPER1 LDX #00 ; CLEAR INDEX
F92E: 61 00 06    .LOOP3 LDA ($0006),X ; GET A CHARACTER
F931: D3 F9 51    BEQ DONE3 ; ZERO SO EXIT
F934: 88 60       CMP #60 ; IS A > 'a' ?
F936: D9 F9 3C    BGT RANGE1 ; YES
F939: 4C F9 4A    JMP NEXT2 ; NO, GO TO NEXT LETTER
F93C: 88 7B       .RANGE1 CMP #7B ; IS A < 'z' ?
F93E: D7 F9 44    BLT RANGE2 ; YES
F941: 4C F9 4A    JMP NEXT2 ; NO, GO TO NEXT LETTER
F944: 55          .RANGE2 SEC ; IN RANGE
F945: A4 20       SBC #20 ; SUBTRACT DECIMAL 32
F947: 63 00 06    STA ($0006),X ; PUT IN STRING
F94A: E8          .NEXT2 INX ; GO TO NEXT LETTER
F94B: D3 F9 51    BEQ DONE3 ; REACHED MAX. LENGTH SO EXIT
F94E: 4C F9 2E    JMP LOOP3
F951: 60          .DONE3 RTS ; EXIT

            ;
            ; _KILLB REMOVE BLANKS FROM THE NULL TERMINATED STRING IN AX
            ;

F952: C6 06       ._KILLB  STA $06
F954: C8 07       STX $07
F956: A8 00       ._KILLB1 LDX #00
F958: 61 00 06    .LOOPK LDA ($0006),X ; GET CHARACTER
F95B: 88 00       CMP #00 ; NULL, SO END OF STRING
F95D: D3 F9 88    BEQ DONEK
F960: 88 20       CMP #20 ; BLANK, SO KILL IT
F962: D3 F9 6C    BEQ SHIFT
F965: E8          INX ; GO TO NEXT CHARACTER
F966: D3 F9 88    BEQ DONEK ; FINISH IF 255
F969: 4C F9 58    JMP LOOPK
F96C: 12          .SHIFT TXA
F96D: 13          TAY ; PUT X IN Y TO SAVE IT
F96E: E8          INX
F96F: 61 00 06    .LOOPS LDA ($0006),X ; GET NEXT CHARACTER
F972: C7          DEX
F973: 63 00 06    STA ($0006),X ; AND SHIFT OVER ONE
F976: E8          INX
```

```
F977: E8         INX ; GO TO NEXT CHARACTER
F978: D3 F9 83   BEQ DONES ; DONE 255
F97B: 88 00      CMP #00 ; WAS IT THE END OF THE STRING?
F97D: D3 F9 83   BEQ DONES ; YES
F980: 4C F9 6F   JMP LOOPS ; NO
F983: 11         .DONES TYA
F984: 14         TAX ; RESTORE X
F985: 4C F9 58   JMP LOOPK ; CONTINUE
F988: 60         .DONEK RTS


            ;
            ; _LENGTH FINDS THE LENGTH OF THE NULL TERMINATED STRING THAT STARTS
            ;     AT THE ADDRESS IN $06 & $07.  THE LENGTH IS RETURNED IN X
            ;

F989: A8 00      ._LENGTH LDX #00 ; CLEAR X
F98B: 61 00 06   .LOOPL LDA ($0006),X ; GET CHARACTER
F98E: D3 F9 9A   BEQ QUIT ; LAST CHARACTER, EXIT
F991: E8         INX
F992: D3 F9 98   BEQ MAX ; 255 CHARACTERS
F995: 4C F9 8B   JMP LOOPL
F998: A8 FF      .MAX LDX #FF ; INDICATE MAXIMUM LENGTH
F99A: 60         .QUIT RTS

            ;
            ; _MAKEHEX CONVERTS THE ASCII CHARACTER IN A INTO A HEX DIGIT
            ; IT SETS $05 TO ZERO IF NOT A DIGIT.  USES Y.

F99B: 88 3A      ._MAKEHEX CMP #3A ; IS A < '9'+1 ?
F99D: D7 F9 B0   BLT DIGIT1 ; YES, CONVERT IT AS NUMBER
F9A0: 88 41      CMP #41 ; IS A < 'A' ?
F9A2: D7 F9 BE   BLT ERRM
F9A5: 88 46      CMP #46 ; IS A > 'F' ?
F9A7: D9 F9 BE   BGT ERRM
F9AA: 55         SEC
F9AB: A4 37      SBC #37 ; SUBTRACT TO GET VALUE
F9AD: 4C F9 C2   JMP DONEH
F9B0: 88 2F      .DIGIT1     CMP #2F ; IS A > '0'-1 ?
F9B2: D9 F9 B8   BGT OK1
F9B5: 4C F9 BE   JMP ERRM
F9B8: 55         .OK1 SEC
F9B9: A4 30      SBC #30 ; SUBTR. TO GET VALUE
F9BB: 4C F9 C2   JMP DONEH
F9BE: A6 00      .ERRM LDY #00
F9C0: CA 05      STY $05 ; SET FLAG
F9C2: 60         .DONEH RTS

            ;
            ; _MAKEBYTE TAKES TWO DIGITS FROM STRING IN $06 & $07 AND CREATES
            ; A BYTE WHICH IS STORED IN $04.  INCREMENTS X.
            ;

F9C3: 61 00 06   ._MAKEBYTE LDA ($0006),X ; GET FIRST DIGIT
F9C6: 20 F9 9B   JSR _MAKEHEX ; CONVERT TO HEX
F9C9: 26         LSH ; SHIFT TO UPPER NIBBLE
F9CA: 26         LSH
F9CB: 26         LSH
F9CC: 26         LSH
F9CD: C6 04      STA $04
F9CF: E8         INX
F9D0: 61 00 06   LDA ($0006),X
F9D3: 20 F9 9B   JSR _MAKEHEX
F9D6: 54         CLC
F9D7: B5 00 04   ADC $0004 ; ADD
F9DA: C6 04      STA $04 ; BYTE COMPLETED
F9DC: 60         RTS
```

```
             ;
             ; _NUMBER CONVERTS THE NULL TERMINATED STRING IN AX INTO AN ADDRESS
             ;     AND PLACES IT IN THE MONITOR SCRATCH LOCATIONS ($01 & $02)
             ; STORES RESULT OF CONVERSION IN $05 ( 0 = ERROR, 1 = OKAY )
             ;
F9DD: C6 06       ._NUMBER STA $06
F9DF: C8 07       STX $07
F9E1: A8 00       ._NUMBER1 LDX #00
F9E3: C8 05       STX $05
F9E5: 46 00 05    INC $0005 ; SET FOR NO ERROR INITIALLY
F9E8: 20 F9 89    JSR _LENGTH ; GET THE LENGTH
F9EB: 86 00       CPX #00 ; NO LENGTH - ERROR
F9ED: D3 FA 4C    BEQ ERRORN
F9F0: 86 04       CPX #04
F9F2: D9 FA 4C    BGT ERRORN ; TOO MANY DIGITS
F9F5: D3 FA 38    BEQ FOURN
F9F8: 86 03       CPX #03
F9FA: D3 FA 23    BEQ THREEN
F9FD: 86 02       CPX #02
F9FF: D3 FA 13    BEQ TWON
FA02: A8 00       .ONEN LDX #00 ; SINGLE DIGIT
FA04: 61 00 06    LDA ($0006),X
FA07: 20 F9 9B    JSR _MAKEHEX
FA0A: C6 02       STA $02
FA0C: A9 00       LDA #00
FA0E: C6 01       STA $01
FA10: 4C FA 4F    JMP DONEN
FA13: A8 00       .TWON LDX #00 ; TWO DIGITS
FA15: 20 F9 C3    JSR _MAKEBYTE
FA18: C0 04       LDA $04
FA1A: C6 02       STA $02
FA1C: A9 00       LDA #00
FA1E: C6 01       STA $01
FA20: 4C FA 4F    JMP DONEN
FA23: A8 00       .THREEN LDX #00 ; THREE DIGITS
FA25: 61 00 06    LDA ($0006),X
FA28: 20 F9 9B    JSR _MAKEHEX
FA2B: C6 01       STA $01
FA2D: E8          INX
FA2E: 20 F9 C3    JSR _MAKEBYTE
FA31: C0 04       LDA $04
FA33: C6 02       STA $02
FA35: 4C FA 4F    JMP DONEN
FA38: A8 00       .FOURN LDX #00
FA3A: 20 F9 C3    JSR _MAKEBYTE ; MAKE A BYTE
FA3D: C0 04       LDA $04
FA3F: C6 01       STA $01 ; PUT HIGH BYTE IN PLACE
FA41: E8          INX
FA42: 20 F9 C3    JSR _MAKEBYTE
FA45: C0 04       LDA $04
FA47: C6 02       STA $02 ; PUT LOW BYTE IN PLACE
FA49: 4C FA 4F    JMP DONEN ; EXIT
FA4C: 45 00 05    .ERRORN DEC $0005 ; SET ERROR
FA4F: 60          .DONEN RTS

             ;
             ; _ADD16 ADD THE SIXTEEN BIT NUMBER IN $01 & $02 TO THE SIXTEEN BIT
             ;  NUMBER IN $0A & $0B.  RESULT IN $01 & $02.
             ;
FA50: 54          ._ADD16 CLC
FA51: C0 02       LDA $02
FA53: B5 00 0B    ADC $000B
FA56: C6 02       STA $02
FA58: C0 01       LDA $01
FA5A: B5 00 0A    ADC $000A
```

```
FA5D: C6 01       STA $01
FA5F: 60          RTS

            ;
            ; _EXECUTE EXECUTES THE COMMAND IN THE TEXT BUFFER
            ;
            ;  COMMANDS ARE:
            ;
            ;       G <ADDRESS> - GO AT ADDRESS
            ; L <ADDRESS> - LIST 25 LINES STARTING @ ADDRESS (DISASSEMBLED)
            ;      S <ADDRESS> - SUBSTITUTE/VIEW MEMORY @ ADDRESS
            ;      X <ADDRESS> - HEX DUMP A SCREEN IN STARTING @ ADDRESS
            ;        ! -   CLEAR THE SCREEN & HOME CURSOR
            ;        <ADDRESS> - SHOW THE CONTENTS OF <ADDRESS>
            ;

FA60: AA 03 00    ._EXECUTE LDA TEXT ; GET FIRST LETTER OF COMMAND
FA63: 88 00       CMP #00 ; NULL LINE
FA65: D3 FF 7E    BEQ DONE4 ; EXIT
FA68: 88 47       CMP #47 ; G - GO
FA6A: D3 FA 9B    BEQ GO
FA6D: 88 4C       CMP #4C ; L - LIST
FA6F: D3 FA AB    BEQ LIST
FA72: 88 53       CMP #53 ; S - SUBSTITUTE
FA74: D3 FE 99    BEQ CHANGE
FA77: 88 58       CMP #58 ; X - DUMP
FA79: D3 FF 0E    BEQ DUMP
FA7C: 88 21       CMP #21
FA7E: D3 FF 79    BEQ HOME ; ! - HOME
FA81: A9 03       LDA #03
FA83: A8 00       LDX #00
FA85: 20 F9 DD    JSR _NUMBER ; IS IT AN ADDRESS?
FA88: C0 05       LDA $05 ; ERROR CODE
FA8A: D3 FA 90    BEQ GARBAGE ; =0 , GARBAGE
FA8D: 4C FF 5B    JMP ADDR ; SHOW CONTENTS OF ADDRESS


FA90: A9 3F       .GARBAGE    LDA #3F ; UNKNOWN, PRINT ? AND RETURN
FA92: 20 F8 76    JSR _COUT
FA95: A9 0D       LDA #0D
FA97: 20 F8 76    JSR _COUT
FA9A: 60          RTS

FA9B: A9 03       .GO LDA #03 ; EXECUTE A ROUTINE
FA9D: A8 01       LDX #01
FA9F: 20 F9 DD    JSR _NUMBER ; CONVERT ADDRESS
FAA2: C0 05       LDA $05 ; GET CONVERSION FLAG
FAA4: D3 FA 90    BEQ GARBAGE ; IF = 0 THEN ERROR
FAA7: 20 00 00    JSR ZERO ; NO ERROR, EXECUTE
FAAA: 60          RTS ; AND RETURN TO MONITOR

FAAB: A9 00       .LIST LDA #00
FAAD: C6 0A       STA $0A
FAAF: AE          INA
FAB0: C6 0B       STA $0B ; STORE A 1 FOR INCREMENTING ADDRESS
FAB2: A9 03       LDA #03
FAB4: A8 01       LDX #01
FAB6: 20 F9 DD    JSR _NUMBER ; CONVERT ADDRESS
FAB9: C0 05       LDA $05
FABB: D3 FA 90    BEQ GARBAGE
FABE: A6 10       LDY #10 ; LINE COUNT
FAC0: A8 00       .LOOPLIST LDX #00
FAC2: 61 00 01    LDA ($0001),X ; GET A BYTE
FAC5: 14          TAX
FAC6: AD FB FA    LDA TYPE,X ; GET LENGTH/TYPE
FAC9: C6 1A       STA $1A
FACB: AD FC FA    LDA OFFSET,X ; GET OFFSET TO LABEL
FACE: C6 1B       STA $1B
```

```
FAD0: C0 01      LDA $01
FAD2: C2 02      LDX $02
FAD4: 20 F9 20   JSR _PRINTWORD ; PRINT ADDRESS
FAD7: A9 2D      LDA #2D ; '-'
FAD9: 20 F8 76   JSR _COUT
FADC: A9 20      LDA #20
FADE: 20 F8 76   JSR _COUT
FAE1: A8 00      LDX #00
FAE3: 61 00 01   LDA ($0001),X
FAE6: C6 1C      STA $1C
FAE8: 20 F8 E9   JSR _PRINTHEX ; PRINT 1ST BYTE
FAEB: A9 20      LDA #20
FAED: 20 F8 76   JSR _COUT
FAF0: C0 1A      LDA $1A
FAF2: A0 F0      AND #F0
FAF4: 54         CLC
FAF5: 29         RSH
FAF6: 29         RSH
FAF7: 29         RSH
FAF8: 29         RSH
FAF9: 88 01      CMP #01
FAFB: D3 FB 28   BEQ ONEBYTE
FAFE: 88 02      CMP #02
FB00: D3 FB 30   BEQ TWOBYTE
FB03: 20 FA 50   JSR _ADD16
FB06: A8 00      LDX #00
FB08: 61 00 01   LDA ($0001),X
FB0B: C6 1D      STA $1D
FB0D: 20 F8 E9   JSR _PRINTHEX
FB10: A9 20      LDA #20
FB12: 20 F8 76   JSR _COUT
FB15: 20 FA 50   JSR _ADD16
FB18: 61 00 01   LDA ($0001),X
FB1B: C6 1E      STA $1E
FB1D: 20 F8 E9   JSR _PRINTHEX
FB20: A9 20      LDA #20
FB22: 20 F8 76   JSR _COUT
FB25: 4C FB 42   JMP PRINTLABEL
FB28: A8 06      .ONEBYTE LDX #06
FB2A: 20 FF D8   JSR _BLANKS
FB2D: 4C FB 42   JMP PRINTLABEL
FB30: 20 FA 50   .TWOBYTE JSR _ADD16
FB33: A8 00      LDX #00
FB35: 61 00 01   LDA ($0001),X
FB38: C6 1D      STA $1D
FB3A: 20 F8 E9   JSR _PRINTHEX
FB3D: A8 04      LDX #04
FB3F: 20 FF D8   JSR _BLANKS
FB42: C2 1B      .PRINTLABEL LDX $1B
FB44: AD FD FA   LDA LABELS,X
FB47: 20 F8 76   JSR _COUT
FB4A: E8         INX
FB4B: AD FD FA   LDA LABELS,X
FB4E: 20 F8 76   JSR _COUT
FB51: E8         INX
FB52: AD FD FA   LDA LABELS,X
FB55: 20 F8 76   JSR _COUT
FB58: A9 20      LDA #20
FB5A: 20 F8 76   JSR _COUT
FB5D: C0 1A      LDA $1A
FB5F: A0 0F      AND #0F ; GET INSTRUCTION TYPE
FB61: 88 00      CMP #00
FB63: D3 FB 82   BEQ IMMEDIATE
FB66: 88 01      CMP #01
FB68: D3 FB 92   BEQ ABSOLUTE
FB6B: 88 02      CMP #02
FB6D: D3 FB A1   BEQ ABS.X
FB70: 88 03      CMP #03
```

```
FB72: D3 FB BA    BEQ ZEROPAGE
FB75: 88 04       CMP #04
FB77: D3 FB C7    BEQ RELATIVE
FB7A: 88 05       CMP #05
FB7C: D3 FB CA    BEQ INDIRECT
FB7F: 4C FB EA    .IMPLIED JMP NXTINSTR ; IMPLIED OR UNKNOWN - END LINE
FB82: A9 23       .IMMEDIATE LDA #23 ; '#'
FB84: 20 F8 76    JSR _COUT
FB87: A8 00       LDX #00
FB89: 61 00 01    LDA ($0001),X ; DATA
FB8C: 20 F8 E9    JSR _PRINTHEX
FB8F: 4C FB EA    JMP NXTINSTR
FB92: A9 24       .ABSOLUTE LDA #24 ; '$'
FB94: 20 F8 76    JSR _COUT
FB97: C0 1D       LDA $1D
FB99: C2 1E       LDX $1E
FB9B: 20 F9 20    JSR _PRINTWORD
FB9E: 4C FB EA    JMP NXTINSTR
FBA1: A9 24       .ABS.X LDA #24
FBA3: 20 F8 76    JSR _COUT
FBA6: C0 1D       LDA $1D
FBA8: C2 1E       LDX $1E
FBAA: 20 F9 20    JSR _PRINTWORD
FBAD: A9 2C       LDA #2C
FBAF: 20 F8 76    JSR _COUT
FBB2: A9 58       LDA #58
FBB4: 20 F8 76    JSR _COUT
FBB7: 4C FB EA    JMP NXTINSTR
FBBA: A9 24       .ZEROPAGE LDA #24
FBBC: 20 F8 76    JSR _COUT
FBBF: C0 1D       LDA $1D
FBC1: 20 F8 E9    JSR _PRINTHEX
FBC4: 4C FB EA    JMP NXTINSTR
FBC7: 4C FB BA    .RELATIVE JMP ZEROPAGE ; FOR NOW - TREAT AS ZERO PAGE
FBCA: A9 28       .INDIRECT LDA #28 ; '('
FBCC: 20 F8 76    JSR _COUT
FBCF: A9 24       LDA #24
FBD1: 20 F8 76    JSR _COUT
FBD4: C0 1D       LDA $1D
FBD6: C2 1E       LDX $1E
FBD8: 20 F9 20    JSR _PRINTWORD
FBDB: A9 29       LDA #29 ; ')'
FBDD: 20 F8 76    JSR _COUT
FBE0: A9 2C       LDA #2C
FBE2: 20 F8 76    JSR _COUT
FBE5: A9 58       LDA #58
FBE7: 20 F8 76    JSR _COUT
FBEA: A9 0D       .NXTINSTR LDA #0D
FBEC: 20 F8 76    JSR _COUT
FBEF: C5          DEY
FBF0: D3 FB F9    BEQ DONELIST
FBF3: 20 FA 50    JSR _ADD16 ; GO TO NEXT ADDRESS
FBF6: 4C FA C0    JMP LOOPLIST ; AND DO IT AGAIN
FBF9: 60          .DONELIST RTS



FBFA: 16171717   .TYPE HEX 16171717 ; 00    INSTRUCTION LENGTH/TYPE
FBFE: 17171716   HEX 17171716 ; 04
FC02: 17171717   HEX 17171717 ; 08
FC06: 17171717   HEX 17171717 ; 0C
FC0A: 16161616   HEX 16161616 ; 10
FC0E: 16171717   HEX 16171717 ; 14
FC12: 17171717   HEX 17171717 ; 18
FC16: 17171717   HEX 17171717 ; 1C
FC1A: 31171717   HEX 31171717 ; 20
FC1E: 17171616   HEX 17171616 ; 24
FC22: 16161717   HEX 16161717 ; 28
```

```
FC26: 17171717   HEX 17171717 ; 2C
FC2A: 17171716   HEX 17171716 ; 30
FC2E: 32321617   HEX 32321617 ; 34
FC32: 17171717   HEX 17171717 ; 38
FC36: 16161717   HEX 16161717 ; 3C
FC3A: 17171717   HEX 17171717 ; 40
FC3E: 17313117   HEX 17313117 ; 44
FC42: 17171717   HEX 17171717 ; 48
FC46: 31171717   HEX 31171717 ; 4C
FC4A: 17171717   HEX 17171717 ; 50
FC4E: 16161717   HEX 16161717 ; 54
FC52: 17171717   HEX 17171717 ; 58
FC56: 17171717   HEX 17171717 ; 5C
FC5A: 16351735   HEX 16351735 ; 60
FC5E: 17171717   HEX 17171717 ; 64
FC62: 17171717   HEX 17171717 ; 68
FC66: 17171717   HEX 17171717 ; 6C
FC6A: 17171717   HEX 17171717 ; 70
FC6E: 17171717   HEX 17171717 ; 74
FC72: 17171717   HEX 17171717 ; 78
FC76: 17171717   HEX 17171717 ; 7C
FC7A: 17171717   HEX 17171717 ; 80
FC7E: 17312017   HEX 17312017 ; 84
FC82: 20313220   HEX 20313220 ; 88
FC86: 31171717   HEX 31171717 ; 8C
FC8A: 17171717   HEX 17171717 ; 90
FC8E: 17171731   HEX 17171731 ; 94
FC92: 31311717   HEX 31311717 ; 98
FC96: 17171717   HEX 17171717 ; 9C
FC9A: 20202017   HEX 20202017 ; A0
FC9E: 20202017   HEX 20202017 ; A4
FCA2: 20203131   HEX 20203131 ; A8
FCA6: 31321616   HEX 31321616 ; AC
FCAA: 17171717   HEX 17171717 ; B0
FCAE: 17313117   HEX 17313117 ; B4
FCB2: 31313231   HEX 31313231 ; B8
FCB6: 17323217   HEX 17323217 ; BC
FCBA: 23172317   HEX 23172317 ; C0
FCBE: 23162316   HEX 23162316 ; C4
FCC2: 23172323   HEX 23172323 ; C8
FCC6: 23231717   HEX 23231717 ; CC
FCCA: 24312431   HEX 24312431 ; D0
FCCE: 24312431   HEX 24312431 ; D4
FCD2: 24312431   HEX 24312431 ; D8
FCD6: 24312431   HEX 24312431 ; DC
FCDA: 24171717   HEX 24171717 ; E0
FCDE: 17171717   HEX 17171717 ; E4
FCE2: 16171617   HEX 16171617 ; E8
FCE6: 17171617   HEX 17171617 ; EC
FCEA: 17171717   HEX 17171717 ; F0
FCEE: 17171717   HEX 17171717 ; F4
FCF2: 17171717   HEX 17171717 ; F8
FCF6: 17171717   HEX 17171717 ; FC

FCFA: 6C9C9C9C   .OFFSET HEX 6C9C9C9C ; 00 OFFSET TO LABEL
FCFE: 9C9C9C90   HEX 9C9C9C90 ; 04
FD02: 9C9C9C9C   HEX 9C9C9C9C ; 08
FD06: 9C9C9C9C   HEX 9C9C9C9C ; 0C
FD0A: 8D8A8784   HEX 8D8A8784 ; 10
FD0E: 819C9C9C   HEX 819C9C9C ; 14
FD12: 9C9C9C9C   HEX 9C9C9C9C ; 18
FD16: 9C9C9C9C   HEX 9C9C9C9C ; 1C
FD1A: 309C9C9C   HEX 309C9C9C ; 20
FD1E: 9C9C9693   HEX 9C9C9693 ; 24
FD22: 63759C9C   HEX 63759C9C ; 28
FD26: 9C9C9C9C   HEX 9C9C9C9C ; 2C
FD2A: 9C9C9C78   HEX 9C9C9C78 ; 30
FD2E: 2D2A7B9C   HEX 2D2A7B9C ; 34
```

```
FD32: 9C9C9C9C   HEX 9C9C9C9C ; 38
FD36: 7E669C9C   HEX 7E669C9C ; 3C
FD3A: 9C9C9C9C   HEX 9C9C9C9C ; 40
FD3E: 9C2D2A9C   HEX 9C2D2A9C ; 44
FD42: 9C9C9C9C   HEX 9C9C9C9C ; 48
FD46: 339C9C9C   HEX 339C9C9C ; 4C
FD4A: 9C9C9C9C   HEX 9C9C9C9C ; 50
FD4E: 69999C9C   HEX 69999C9C ; 54
FD52: 9C9C9C9C   HEX 9C9C9C9C ; 58
FD56: 9C9C9C9C   HEX 9C9C9C9C ; 5C
FD5A: 6F009C21   HEX 6F009C21 ; 60
FD5E: 9C9C9C9C   HEX 9C9C9C9C ; 64
FD62: 9C9C9C9C   HEX 9C9C9C9C ; 68
FD66: 9C9C9C9C   HEX 9C9C9C9C ; 6C
FD6A: 9C9C9C9C   HEX 9C9C9C9C ; 70
FD6E: 9C9C9C9C   HEX 9C9C9C9C ; 74
FD72: 9C9C9C9C   HEX 9C9C9C9C ; 78
FD76: 9C9C9C9C   HEX 9C9C9C9C ; 7C
FD7A: 9C9C9C9C   HEX 9C9C9C9C ; 80
FD7E: 9C1B1B9C   HEX 9C1B1B9C ; 84
FD82: 1818181E   HEX 1818181E ; 88
FD86: 1E9C9C9C   HEX 1E9C9C9C ; 8C
FD8A: 9C9C9C9C   HEX 9C9C9C9C ; 90
FD8E: 9C9C9C15   HEX 9C9C9C15 ; 94
FD92: 120F9C9C   HEX 120F9C9C ; 98
FD96: 9C9C9C9C   HEX 9C9C9C9C ; 9C
FD9A: 0F12159C   HEX 0F12159C ; A0
FD9E: 0C09069C   HEX 0C09069C ; A4
FDA2: 03000003   HEX 03000003 ; A8
FDA6: 06005D60   HEX 06005D60 ; AC
FDAA: 9C9C9C9C   HEX 9C9C9C9C ; B0
FDAE: 9C09279C   HEX 9C09279C ; B4
FDB2: 24210C0C   HEX 24210C0C ; B8
FDB6: 9C21099C   HEX 9C21099C ; BC
FDBA: 009C039C   HEX 009C039C ; C0
FDBE: 065A2157   HEX 065A2157 ; C4
FDC2: 249C2718   HEX 249C2718 ; C8
FDC6: 1B1E9C9C   HEX 1B1E9C9C ; CC
FDCA: 36363939   HEX 36363939 ; D0
FDCE: 3C3C3F3F   HEX 3C3C3F3F ; D4
FDD2: 42424545   HEX 42424545 ; D8
FDD6: 48484B4B   HEX 48484B4B ; DC
FDDA: 4E9C9C9C   HEX 4E9C9C9C ; E0
FDDE: 9C9C9C9C   HEX 9C9C9C9C ; E4
FDE2: 519C729C   HEX 519C729C ; E8
FDE6: 9C9C549C   HEX 9C9C549C ; EC
FDEA: 9C9C9C9C   HEX 9C9C9C9C ; F0
FDEE: 9C9C9C9C   HEX 9C9C9C9C ; F4
FDF2: 9C9C9C9C   HEX 9C9C9C9C ; F8
FDF6: 9C9C9C9C   HEX 9C9C9C9C ; FC

FDFA: 4C44414C44584C4459414443534243414E444F5241   .LABELS ASC 'LDALDXLDYADCSBCANDORA' ; OPCODE LABELS
FE0F: 454F52434D504350584350595354415354585459   ASC 'EORCMPCPXCPYSTASTXSTY'
FE24: 494E434445434A53524A4D50424E45424551424343   ASC 'INCDECJSRJMPBNEBEQBCC'
FE39: 424C544247544243534242E4742504C425241494E58   ASC 'BLTBGTBCSBNGBPLBRAINX'
FE4E: 494E594445584445594 94E41444341524F4C504C41   ASC 'INYDEXDEYINADCAROLPLA'
FE63: 434C4342524B5254534E4F50525348504850504C50   ASC 'CLCBRKRTSNOPRSHPHPPLP'
FE78: 504841544158544159545841545941545853545358   ASC 'PHATAXTAYTXATYATXSTSX'
FE8D: 524F524C53485345433F3F3F                     ASC 'RORLSHSEC???'

FE99: A9 03       .CHANGE LDA #03
FE9B: A8 01       LDX #01
FE9D: 20 F9 DD    JSR _NUMBER ; CONVERT ADDRESS
FEA0: C0 05       LDA $05
FEA2: D3 FA 90    BEQ GARBAGE
FEA5: 20 FF 64    .AGAINC JSR ADDR1 ; DISPLAY PRESENT VALUE
FEA8: A9 20       LDA #20
```

```
FEAA: 20 F8 76    JSR _COUT
FEAD: 20 F8 76    JSR _COUT
FEB0: 20 F8 89    JSR _INPUT1 ; GET A LINE, NO PROMPT
FEB3: A9 03       LDA >TEXT
FEB5: A8 00       LDX <TEXT
FEB7: 20 F9 28    JSR _UPPER ; MAKE IT UPPERCASE
FEBA: AA 03 00    LDA TEXT
FEBD: 88 00       CMP #00
FEBF: D3 FE FB    BEQ NXTBYTE
FEC2: 88 21       CMP #21
FEC4: D3 FF 0D    BEQ DONEC ; EXIT ON '!' CHARACTER
FEC7: A9 03       LDA >TEXT
FEC9: A8 00       LDX <TEXT
FECB: C6 06       STA $06
FECD: C8 07       STX $07
FECF: A8 00       LDX #00
FED1: 20 F9 89    JSR _LENGTH
FED4: 86 01       CPX #01
FED6: D1 FE EF    BNE DOUBLE
FED9: A8 00       LDX #00 ; SINGLE DIGIT
FEDB: 61 00 06    LDA ($0006),X
FEDE: E8          INX
FEDF: 63 00 06    STA ($0006),X
FEE2: A9 00       LDA #00
FEE4: E8          INX
FEE5: 63 00 06    STA ($0006),X
FEE8: A8 00       LDX #00
FEEA: A9 30       LDA #30
FEEC: 63 00 06    STA ($0006),X
FEEF: A8 00       .DOUBLE LDX #00
FEF1: 20 F9 C3    JSR _MAKEBYTE
FEF4: C0 04       LDA $04
FEF6: A8 00       LDX #00
FEF8: 63 00 01    STA ($0001),X ; PUT IN MEMORY
FEFB: 54          .NXTBYTE CLC
FEFC: A9 01       LDA #01
FEFE: B5 00 02    ADC $0002
FF01: C6 02       STA $02
FF03: A9 00       LDA #00
FF05: B5 00 01    ADC $0001
FF08: C6 01       STA $01
FF0A: 4C FE A5    JMP AGAINC
FF0D: 60          .DONEC RTS

FF0E: A9 03       .DUMP LDA #03
FF10: A8 01       LDX #01
FF12: 20 F9 DD    JSR _NUMBER ; CONVERT ADDRESS
FF15: C0 05       LDA $05
FF17: D3 FA 90    BEQ GARBAGE ; CONVERSION ERROR
FF1A: A6 0A       LDY #0A ; DUMP TEN LINES
FF1C: CA 04       STY $04
FF1E: C0 01       .NEXTLINE LDA $01
FF20: C2 02       LDX $02
FF22: 20 F9 20    JSR _PRINTWORD
FF25: A9 2D       LDA #2D
FF27: 20 F8 76    JSR _COUT
FF2A: A6 08       LDY #08
FF2C: A8 00       LDX #00
FF2E: 61 00 01    .NEXTBYTE LDA ($0001),X ; GET VALUE
FF31: 20 F8 E9    JSR _PRINTHEX
FF34: A9 20       LDA #20
FF36: 20 F8 76    JSR _COUT
FF39: E8          INX
FF3A: C5          DEY
FF3B: D1 FF 2E    BNE NEXTBYTE
FF3E: A9 0D       LDA #0D
FF40: 20 F8 76    JSR _COUT
FF43: 45 00 04    DEC $0004
```

```
FF46: C0 04      LDA $04
FF48: 88 00      CMP #00
FF4A: D3 FF 7E   BEQ DONE4
FF4D: A9 00      LDA #00
FF4F: C6 0A      STA $0A
FF51: A9 08      LDA #08
FF53: C6 0B      STA $0B
FF55: 20 FA 50   JSR _ADD16 ; INCREMENT ADDRESS
FF58: 4C FF 1E   JMP NEXTLINE

FF5B: 20 FF 64   .ADDR JSR ADDR1
FF5E: A9 0D      LDA #0D
FF60: 20 F8 76   JSR _COUT
FF63: 60         RTS

FF64: C0 01      .ADDR1 LDA $01 ; SINGLE ADDRESS
FF66: C2 02      LDX $02
FF68: 20 F9 20   JSR _PRINTWORD
FF6B: A9 2D      LDA #2D
FF6D: 20 F8 76   JSR _COUT
FF70: A8 00      LDX #00
FF72: 61 00 01   LDA ($0001),X
FF75: 20 F8 E9   JSR _PRINTHEX
FF78: 60         RTS

FF79: A9 0C      .HOME LDA #0C ; HOME - CLEAR THE SCREEN
FF7B: 20 F8 76   JSR _COUT
FF7E: 60         .DONE4 RTS

           ;
           ; _REG OUTPUTS THE CURRENT REGISTER VALUES.
           ;

FF7F: C6 0A      ._REG STA $0A ; SAVE THE A
FF81: 12         TXA
FF82: C6 0B      STA $0B ; SAVE THE X
FF84: 11         TYA
FF85: C6 0C      STA $0C ; SAVE THE Y
FF87: A9 FF      LDA >STRY ; MESSAGE
FF89: A8 91      LDX <STRY
FF8B: 20 F8 D2   JSR _PRINT
FF8E: 4C FF 95   JMP NXT
FF91: 593D20   .STRY ASC 'Y= '
FF94: 00   HEX 00
FF95: C0 0C      .NXT LDA $0C
FF97: 20 F8 E9   JSR _PRINTHEX
FF9A: A8 02      LDX #02
FF9C: 20 FF D8   JSR _BLANKS
FF9F: A9 FF      LDA >STRX
FFA1: A8 A9      LDX <STRX
FFA3: 20 F8 D2   JSR _PRINT
FFA6: 4C FF AD   JMP NXT2
FFA9: 583D20   .STRX ASC 'X= '
FFAC: 00   HEX 00
FFAD: C0 0B      .NXT2 LDA $0B
FFAF: 20 F8 E9   JSR _PRINTHEX
FFB2: A8 02      LDX #02
FFB4: 20 FF D8   JSR _BLANKS
FFB7: A9 FF      LDA >STRA
FFB9: A8 C1      LDX <STRA
FFBB: 20 F8 D2   JSR _PRINT
FFBE: 4C FF C5   JMP NXT3
FFC1: 413D20   .STRA ASC 'A= '
FFC4: 00   HEX 00
FFC5: C0 0A      .NXT3 LDA $0A
FFC7: 20 F8 E9   JSR _PRINTHEX
FFCA: A9 0D      LDA #0D
```

```
FFCC: 20 F8 76    JSR _COUT
FFCF: C0 0C       LDA $0C ; RESTORE REGISTERS
FFD1: 13          TAY
FFD2: C0 0B       LDA $0B
FFD4: 14          TAX
FFD5: C0 0A       LDA $0A
FFD7: 60          RTS

           ;
           ; _BLANKS OUTPUT BLANKS, NUMBER IN X.  DESTROYS A.
           ;

FFD8: A9 20       ._BLANKS LDA #20
FFDA: 20 F8 76    .LOOPBL JSR _COUT
FFDD: C7          DEX
FFDE: D3 FF E4    BEQ DONEBL
FFE1: 4C FF DA    JMP LOOPBL
FFE4: 60          .DONEBL RTS
```

Symbol Table

[0200] KOUT	[0201] CSTB	[0202] KBDS	[0203] KEYB
[0100] STAK	[FFFD] COLD	[FFFA] BREAK	[0300] TEXT
[0000] ZERO	[F800] _MON	[F80F] MESSAGE	[F830] VERSION
[F832] _MON1	[F851] _BRK	[F85E] STRNG	[F867] NXTBRK
[F876] _COUT	[F87D] _KEYIN	[F884] _INPUT	[F889] _INPUT1
[F88B] LOOP	[F8AA] DEL	[F8BD] DONE	[F8CA] NOCHAR
[F8D0] FINISH	[F8D1] END	[F8D2] _PRINT	[F8D6] PRINT1
[F8D8] LOOP2	[F8E8] DONE2	[F8E9] _PRINTHEX	[F8FF] HEX1
[F905] NEXT	[F917] HEX2	[F91D] END2	[F920] _PRINTWORD
[F928] _UPPER	[F92C] _UPPER1	[F92E] LOOP3	[F93C] RANGE1
[F944] RANGE2	[F94A] NEXT2	[F951] DONE3	[F952] _KILLB
[F956] _KILLB1	[F958] LOOPK	[F96C] SHIFT	[F96F] LOOPS
[F983] DONES	[F988] DONEK	[F989] _LENGTH	[F98B] LOOPL
[F998] MAX	[F99A] QUIT	[F99B] _MAKEHEX	[F9B0] DIGIT1
[F9B8] OK1	[F9BE] ERRM	[F9C2] DONEH	[F9C3] _MAKEBYTE
[F9DD] _NUMBER	[F9E1] _NUMBER1	[FA02] ONEN	[FA13] TWON
[FA23] THREEN	[FA38] FOURN	[FA4C] ERRORN	[FA4F] DONEN
[FA50] _ADD16	[FA60] _EXECUTE	[FA90] GARBAGE	[FA9B] GO
[FAAB] LIST	[FAC0] LOOPLIST	[FB28] ONEBYTE	[FB30] TWOBYTE
[FB42] PRINTLABEL	[FB7F] IMPLIED	[FB82] IMMEDIATE	[FB92] ABSOLUTE
[FBA1] ABS.X	[FBBA] ZEROPAGE	[FBC7] RELATIVE	[FBCA] INDIRECT
[FBEA] NXTINSTR	[FBF9] DONELIST	[FBFA] TYPE	[FCFA] OFFSET
[FDFA] LABELS	[FE99] CHANGE	[FEA5] AGAINC	[FEEF] DOUBLE
[FEFB] NXTBYTE	[FF0D] DONEC	[FF0E] DUMP	[FF1E] NEXTLINE
[FF2E] NEXTBYTE	[FF5B] ADDR	[FF64] ADDR1	[FF79] HOME
[FF7E] DONE4	[FF7F] _REG	[FF91] STRY	[FF95] NXT
[FFA9] STRX	[FFAD] NXT2	[FFC1] STRA	[FFC5] NXT3
[FFD8] _BLANKS	[FFDA] LOOPBL	[FFE4] DONEBL	