A MICROPROCESSOR DEVELOPMENT SYSTEM
FOR THE INTEL 8748 MICROCOMPUTER

by

Theodore Clark Seward, Jr.

December 1979

Thesis Advisor: R. Panholzer

Approved for public release; distribution unlimited.
A microprocessor development system (MDS) for the Intel 8748 microcomputer was designed and built around the Naval Postgraduate School's TRS-80 computer system. This MDS provides the capability to use the TRS-80 as an editor to write and edit 8748 mnemonic programs and store them on a magnetic disk. Also developed was an assembler to convert the user generated source program into object code. As a final step,
20. (continued)

A software driven hardware programmer has been constructed to enable the object code to be loaded into the Erasable Programmable Read Only Memory (EPROM) on the 8748 microcomputer chip.
A Microprocessor Development System
for the Intel 8748 Microcomputer

by

Theodore Clark Seward, Jr.
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, June 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1979

Approved by:

R. Paulose
Thesis Advisor

Mitchell K. Cotton
Second Reader

Chairman, Department of Electrical Engineering

William M. Tolle
Dean of Science and Engineering
ABSTRACT

A microprocessor development system (MDS) for the Intel 8748 microcomputer was designed and built around the Naval Postgraduate School's TRS-80 computer system. This MDS provides the capability to use the TRS-80 as an editor to write and edit 8748 mnemonic programs and store them on a magnetic disk. Also developed was an assembler to convert the user generated source program into object code. As a final step, a software driven hardware programmer has been constructed to enable the object code to be loaded into the Erasable Programmable Read Only Memory (EPROM) on the 8748 microcomputer chip.
# TABLE OF CONTENTS

I. INTRODUCTION----------------------------------------- 7

II. INTEL 8748 MICROCOMPUTER----------------------------- 9

III. TRS-80 MICROCOMPUTER---------------------------- 15

IV. MICROPROCESSOR DEVELOPMENT SYSTEM SOFTWARE-------- 23
    A. EDITOR PROGRAM----------------------------------- 25
    B. ASSEMBLER PROGRAM------------------------------- 30
    C. PROGRAMMER PROGRAM------------------------------ 35

V. PROGRAMMER HARDWARE AND OPERATION-------------------- 41

VI. PROGRAMMER POWER SUPPLIES--------------------------- 53

VII. CONCLUSIONS------------------------------------------ 57

APPENDIX A: Program Listing for Printer Subroutine---- 60

APPENDIX B: Flowchart for Editor-------------------------- 62

APPENDIX C: Flowchart for Assembler----------------------- 66

APPENDIX D: Flowchart for Programmer---------------------- 70

APPENDIX E: 8748 Editor/Assembler Operating Instructions---------------------------------- 74

APPENDIX F: EDTASM Program------------------------------- 80

APPENDIX G: Master Program------------------------------- 81

APPENDIX H: Assemble Program---------------------------- 84

APPENDIX I: "Program" Program---------------------------- 94

APPENDIX J: Sample Assembled Program Printout---------- 98

LIST OF REFERENCES----------------------------------------- 100

INITIAL DISTRIBUTION LIST----------------------------------- 101
LIST OF FIGURES

1. Intel 8748 architecture------------------------ 11
2. Data transfer instructions---------------------- 14
3. TRS-80 memory map---------------------------- 16
4. TRS-80 system block diagram-------------------- 17
5. Inter mode map-------------------------------- 21
6. Editor program block diagram------------------- 27
7. Programmer schematic page 1---------------------- 42
8. Programmer schematic page 2---------------------- 43
9. Programmer circuit board------------------------- 44
10. Pinout of expansion interface------------------- 46
11. Intel 8212 I/O port schematic------------------ 48
12. Programmer timing chart------------------------- 52
13. 25 volt power supply--------------------------- 54
I. INTRODUCTION

The Electrical Engineering (EE) Department at the Naval Postgraduate School operates a small but growing microprocessor/microcomputer laboratory. This laboratory supports several EE courses in microprocessor applications as well as student thesis efforts. Since the primary thrust of these efforts is the implementation of microprocessors in operating circuits, the most useful tool to have available is the microprocessor development system (MDS). Such a system will typically allow the user to write microprocessor programs in assembly level language using a keyboard and cathode ray tube (CRT) to provide for clarity and ease of editing. The MDS will then assemble the assembly level language into machine language and provide the capability to program an Electrically Programmable Read Only Memory (EPROM) with the newly generated program. Additional features normally available on an MDS are a debugging facility which allows for dynamic operation of the program structure and an In-Circuit-Emulator (ICE) to enable the MDS to plug into the circuit under trial and act as the microprocessor with memory. The system presently available in the EE laboratory for this purpose is built around the Tektronix 8002 development system which has modules for the Intel 8080 and the Motorola 6800 microprocessors. Because of the availability of this development system, most projects have been designed around
the 8080 and 6800 microprocessor which until recently have been representative of industry standards in an 8 bit word size machine.

As an additional applications tool the school has obtained a number of Intel 8748 microcomputers. This single "chip" computer provides a complete system in one package and is useful particularly for controller oriented applications. A more detailed description of this chip is provided in the next section.

Because of the usefulness of the 8748 and the lack of a software controlled development system it was decided to design and build an MDS for the 8748 using the TRS-80 microcomputer. The TRS-80 had also been recently acquired by the EE department and expanded to include 64K bytes of memory, mini disk drive, and a printer interface.
II. INTEL 8748 MICROCOMPUTER

A majority of the Central Processing Units (CPU) on the market today are microprocessors only, requiring many support chips such as bus controllers, clocks, RAM, ROM, and input/output (I/O) ports to allow for a functioning computer system.

The Intel 8748 is technically more than a microprocessor. It is in reality a microcomputer and is advertised as such. The 8748 microcomputer is therefore a significant deviation from the support chip philosophy, trending instead toward a complete and self-contained system on one chip of silicon. While such a trend will provide for an increased miniaturization of many components and systems, such closed-end units have only limited capability to be expanded for larger applications. For this reason a market will always exist for the support chip philosophy.

The 8748 microcomputer is actually a development tool intended for use in engineering design for systems which will eventually be equipped with one of several microcomputers in the Intel MCS-48 product line. The other members of the MCS-48 family contain varying capabilities, all of which require off-chip memory or factory programmed ROM. The 8748 contains all of the capabilities of the other chips but utilizes a 1K on-chip EPROM. This EPROM allows the 8748 microcomputer to be used to perfect the program for a given system, or it may be reused for many different applica-
tions by erasing and reprogramming the EPROM. Thus, the 
8748 is an excellent tool for use by students in project 
work.

The architecture of the 8748 microcomputer can be seen 
in Figure 1. This design utilizes NMOS technology to achieve 
the following capabilities:

1. CPU with 8 bit word handling capability.
2. 1K x 8 EPROM.
3. 64 x 8 RAM for data registers (in two banks).
4. 27 I/O lines normally used as 3-8 bit I/O ports.
5. 8 bit event counter.
6. Single +5 volt power supply requirement.
7. 8 level working stack.
8. RC, LC, crystal, or external frequency source 
   for clock.
10. 40 pin dual inline pin (DIP) package.

By selecting a 6 MHz clock crystal, the 8748 operates 
with an instruction cycle of 2.5 microseconds. This speed 
is on a par with the fastest of the CPU chips presently on 
the market, particularly when it is considered that 70% 
of the 8748 instructions are single byte and the remainder 
are only 2 byte. This compares with many of the major CPU 
chips which have 2 and 3 byte instructions.

The following pin-out from Reference 1 describes pin 
utilization:
Figure 1 - Intel 8748 Architecture
The instruction set for the Intel-8748 consists of 96 total instructions. These instructions all execute in either 2.5 or 5 microseconds when using the 6 MHz crystal. Over half of them execute in a single cycle. Many of the instructions are designed to handle BCD and single bit operations for controller oriented applications. Figure 2
is a good picture of how the data transfer instructions interact. Reference 1 contains a complete description of both the hardware and software for the 8748.
Figure 2 - Data Transfer Instructions
III. TRS-80 MICROCOMPUTER

The TRS-80 microcomputer is a small hobby and business computer sold nationwide by Radio Shack franchises. At the time of this writing the company is credited with having sold over 135,000 units, making it the leader in total computers sold. The TRS-80 is built around the Zilog Z-80 microprocessor which is an evolution from the Intel 8080. The TRS-80 has several different levels of capability based on a building block approach. The lowest block available is the keyboard/computer with only 4K of RAM and a very simple BASIC language capability. The highest block is that which is currently available in the EE lab's model. This includes a total 64K of memory and disk BASIC which is loaded into the RAM from a mini-disk (5¼" diskette) drive.

The TRS-80 is a memory mapped system providing fixed addresses for various preprogrammed functions such as video display. Figure 3 shows the 64K system memory map and reveals which portions of memory are available for general user use, normally the highest 48K of RAM. The preprogrammed functions are, of course, in ROM. The user accessible RAM is dynamic, with a 450 millisecond access time and a refresh period of 2 milliseconds. Figure 4 shows a macro block diagram of the TRS-80 system.

The TRS-80 in this configuration uses an operating system (OS) called TRSDOS (TRS Disk Operating System).
Figure 3 - TRS-80 Memory Map
This OS is analogous to the operating systems used on large computers. It enables the user to communicate with the computer using only high level languages and relieves him of the need to manage such computer housekeeping functions as where to store programs and interfacing the CPU with the I/O and storage devices. Thus, the machine level language operations are normally transparent to the user. Programs written in Z-80 machine language can, however, be loaded and run in the TRS-80 using two different methods. One is to load the machine language from a "system" tape which has the program already loaded on it. The other method is to use the BASIC command POKE (address, value) to load one word at a time. It is interesting to note that the variables in the POKE command must be in decimal, which means converting all addresses and program instructions from hexadecimal to decimal.

The Postgraduate School's TRS-80 system also includes an "expansion interface." This unit interfaces directly with the computer and contains 32K words of the total 64K possible memory. It also has a number of bus outputs which provide for parallel printer output, disk drive operations, RS-232 serial I/O and a full system bus for user access.

Since the computer lab already had a very good line printer in the Teletype model 40, it was decided to utilize this printer for the TRS-80 output. The model 40 is an RS-232 serial data unit, however, and the original TRS-80 provided no serial output for listing of programs. Instead,
Radio Shack sells several printers which are driven by the TRS-80 parallel output. The first attempt at interfacing was conducted as part of a student thesis. This project involved construction of a parallel to serial conversion device using standard Universal Asynchronous Receiver Transmitter (UART) techniques. Unfortunately, the TRS-80 does not output a carriage return on its printer bus. Both the carriage return and line feed are driven by the line feed output in the Radio Shack printers. Since both signals are required individually to drive the TTY printer, it was necessary to use an Intel 8748 microcomputer chip, in conjunction with the UART, to provide a carriage return each time a line feed was recognized on the bus. While this device did allow for output to the printer, it had a persistent problem of also inserting random line feeds which resulted in undesirable appearing printouts.

It was decided, shortly after the parallel-serial conversion device failed completely, to purchase Radio Shack's RS-232-C interface unit. This saved many manhours of additional engineering effort and provided for trouble-free printer output. One drawback to using this serial output method is that the unit is driven via special software which must be loaded into user RAM each time the system is powered up. The small machine language program to accomplish this is initially stored in the highest portion of memory and then write protected to prevent the system from putting other data in those memory locations. This memory
protection feature is part of the operating system and is implemented on power up by answering the question MEMORY SIZE? with a decimal address. All RAM above that address is then locked out of being utilized for BASIC programs. The assembly level language program required for operation of the serial printer output is provided in Appendix A. Several switch selections on the RS-232-C board allow the TRS-80 to also be used as a terminal for another computer at several different baud rates. Reference 2 provides further details on operation of the RS-232 unit.

The disk BASIC language used in this machine is a very capable high level language. The commands are simple and straightforward while providing maximum capabilities. It is especially strong in the number of commands available for manipulating "strings" of alphanumeric characters. This ability made BASIC an excellent choice in this MDS application. References 3, 4 and 5 contain the BASIC commands available along with descriptions and sample applications.

While the user of the MDS system does not normally require knowledge of DOS or system commands, there are several which might be of use. To give a better picture of how the different modes interact with one another, the map in Figure 5 is provided. To get from one operating mode to another, code words provided in this map are typed and "ENTER" is depressed. In the case of going from the 8748 editor/assembler, the words "press BREAK key" are not typed. Rather there is a key in the upper right hand corner labeled
Figure 5 - Inter Mode Map
"BREAK." Holding this key down for several seconds will cause program execution to halt. The word READY will then be printed, indicating the system is in BASIC mode.

The primary commands useful to the MDS user while in TRSDOS are listed below:

- **DIR**: Lists all user files stored on the disk.
- **DIR (A)**: Lists all user files along with the space taken up by those files.
- **LIST (filename)**: Prints the contents of the file on the CRT.
- **KILL (filename)**: Erases the chosen file (filename).
- **FREE**: Lists total disk space remaining. (Each diskette holds .48 user files and 44 maximum granules. For a further discussion of files and granules see Ref. 4.)
- **BASIC**: Transfers to the BASIC mode.

While in the BASIC mode the following commands are usable:

- **CMD "S"**: Transfers to TRSDOS mode.
- **KILL (filename)**: Erases file (filename).
- **LIST**: Prints present program on the CRT.
- **RUN**: Runs the program currently in TRS-80 memory.
- **RUN "EDTASM"**: Loads and runs the 8748 editor/assembler.

Commands to be used in the 8748 editor/assembler mode are covered in the next several chapters.
Once the topic for this thesis had been selected, most of the bounding parameters were automatically defined. The choice of the Intel 8748 defined the assembly level language to be used. Selecting the TRS-80 as the computer system in which to implement the microprocessor development system defined the majority of the hardware as well as the programming languages to use. The major decisions remaining to be made involved what capabilities to include in the MDS. The components normally present in a typical MDS are listed below:

1. Editor
2. Assembler
3. Debugger
4. EPROM Programmer
5. In Circuit Emulator (ICE)

Because of the finite time available to carry out this project, it was decided to concentrate on the components which were an absolute requirement to provide an ability to implement a programmed 8748 microcomputer. For this reason an editor, assembler, and EPROM programmer were included as the most essential tools. Additional hardware and software room has been left in the project to allow future student projects to concern the debugger and ICE as additions to this MDS. As a possible adjunct to the MDS,
groundwork was also laid for a software driven EPROM programmer for the Intel 2708 and 2716 chips. These EPROMs would not normally be used with the 8748 microcomputer, but their prevalence at the school for other applications, along with a paucity of easy to use programmers, made such an addition to the MDS desirable. Unfortunately time did not allow for completion of that effort.

For the software portion of this project, it was decided to use the BASIC language capability of the TRS-80 rather than the Z-80 machine language. While the machine language would have been more efficient and would have executed faster, the use of BASIC was selected primarily due to the considerable and time consuming effort required to write the software programs in assembly level language.

The BASIC software is broken down into four different programs rather than loaded as one large program for several reasons. First, it was desired to minimize the amount of system memory taken up by the operating program to allow for maximum room for 8748 program lines and comments. Secondly, calling another program into memory is made especially simple with the disk system because the storing of data and loading of programs is so straightforward and rapid. Third, the function of the editor, assembler, and EPROM programmer are different and independent of each other. Fourth, the writing of each program is simpler if it is an entity independent of the other programs. Thus, the software for the MDS is broken up into the following programs:
**EDTASM**  Loads the printer serial output machine language program into memory and loads and runs the editor program.

**MASTER**  Editor program. Provides for inputting and editing of 8748 assembly language programs. Transfers to assembler upon command.

**ASSEMBLE**  Assembles 8748 mnemonics to machine language represented in hexadecimal format. Provides CRT printout of errors. Also provides hard copy printout of assembled program.

**PROGRAM**  Converts assembled hex code to decimal and outputs to the programmer. Verifies EPROM is correctly programmed. Reads EPROM upon command.

A. EDITOR PROGRAM  
The first portion of this thesis was development of the editor for use in entering the Intel mnemonic code for the 8748 into the MDS. Use of mnemonics is an integral part of an MDS because writing the programs for the 8748 or any other computer would be a difficult and time consuming task if machine language were used. The major advantage of mnemonics is that they have an English language meaning while machine language is simply a string of numbers in one of several possible bases.

In addition to providing a neat format for entering mnemonic instructions, the editor provides many operator aids. Among these useful aids are an "edit" mode to allow for changes, additions, and deletions to the program text; a comments column to allow the operator to describe what various program steps do; and a capability to store the
program under development on magnetic disk for further editing at a later time.

The operating format for the editor program is apparent by examining the flowchart in Appendix B. The program is written to provide for a number of modules, each of which operates independently of the other. Figure 6 shows the relationship of these modules.

To begin editing a program, the operator first loads the system as described in Appendix E. When the statement "enter mode selection" appears on the CRT the operator first types in INPUT to enter that mode. INPUT is repeated on the CRT to confirm to the operator that he is in that mode. Using the mnemonics listed in either Ref. 1 or Ref. 6, the desired assembly level program for the 8748 is entered. Correct format for these line entries is accomplished by using the right arrow key (→) on the TRS-80 which provides a tab to columns at 8, 16, 24, 32, 40 and 48 spaces across. The first column contains only labels consisting of one to six letters and followed by a colon (:). If no label is used, this column is left blank. The second column contains the opcode of the desired instruction. The third column contains the operand applicable to the opcode selected. The operand must be in decimal rather than hex code for quantities. In the case of addresses, the operand must be a one to six letter label which will be used to point to the correct address during assembly. The fourth column is available
Figure 6 - Editor Program Block Diagram
for any comments the user may desire to include for the purpose of describing program operations. These comments must be preceded by a semicolon (;) to prevent the assembler from confusing comments with operands. If the operand is longer than the 8 spaces available in the third column, the comments are started after several spaces instead of pressing the tab key to the fifth column.

Because of the large amount of memory required to store comments it is desirable to limit both the length and number used to a minimum. Also, the TRS-80 allows for single strings of a maximum 256 bytes in length. Since each program line (including comments) is stored as one string, the 256 byte limit will be exceeded if too long a comment is included and the error "string too long" will appear. This will result in the system dropping out of RUN and back to BASIC mode. If this occurs, all data previously entered in memory will be lost when the program is reinitiated. If this or any other error results in the program "bombing," a READY will appear on the CRT indicating the system is in BASIC mode. To get back into the editor/assembler again simply type RUN "EDTASM".

Once the complete program has been entered using the input mode, it is desirable to check for errors in the edit mode by typing EDIT. If the user were confident of his input he could simply enter FILE (filename) which transfers his 8748 program, complete with comments, to disk storage. The filename used with this command can be any group of
letters from one to eight in length. One space must be allowed between FILE and the filename when typing it in.

Another option for leaving the input mode is to enter QUIT which puts the program back in the command mode after resetting the pointer to zero. This command causes all previous lines written to be lost.

Assuming the user went directly from the input to the edit mode, the next logical choice of action would be to type PRINT and check the program listing on the CRT for errors. In the event the program is too long to fit on the screen the rapid scroll can be halted by pressing shift and @ simultaneously. The scroll is started again by striking the space bar (or any other key).

When entering the edit mode the editor pointer will be pointing to the first line in the program and that line will be displayed on the CRT. The pointer can be moved by using the following commands:

- UP     Moves the pointer up one line.
- DN     Moves the pointer down one line.
- EOF    Moves the pointer to the end of the file.
- TOF    Moves the pointer to the top line of the file.

The command "L /substring/" is used to move the pointer to the location of the first line in the text which contains the exact substring located between the slash (/) lines. One space must be provided between the "L" and the first "/". Note that no quotes (") are actually used in this command. The length of the substring is not critical but
it should be long enough to ensure the program does not locate another line with that same short substring.

Once an error is located and the pointer is at that line the following commands are used to make corrections:

- **C /substring 1/substring 2/** Replaces all of substring 1 with all of substring 2. Again, one space must be inserted between the "C" and the first "/".
- **DEL** Deletes the entire line.
- **INS** Provides for insertion of a new line above the current pointer position.

When all editing has been completed, the operator would use the command FILE (filename) again to place the program on the disk. If assembly of this program is then desired, the command ASM (filename) is entered. A single space must be inserted between the "ASM" and the filename. This results in the program "filename" being stored on the disk under the name "STORE" to enable the assembler program to know which file to assemble. The assemble program is then loaded into memory from disk storage and executed.

B. ASSEMBLER PROGRAM

This program has the responsibility for converting the assembly level mnemonics into hex code. The TRS-80 automatically converts the hex code to binary for loading into the EPROM on the computer chip. This is an extremely lengthy program which, without a great deal of sophistication, examines each mnemonic in turn and assigns the correct hexadecimal
code for further action. The basic flowchart for ASSEMBLE can be seen in Appendix C and the program listing is in Appendix H. The following variable usage is assigned for this program:

- **X(L)** Full line from editor.
- **T** Full line but with comments deleted.
- **BK** Number of bytes in a given opcode.
- **D(L)** Byte number in decimal.
- **HX$(L)$** Byte number in hex.
- **V(I)** Label (if any).
- **Y(L)** Hex code for opcode with 2 byte instructions.
- **Z(L)** Hex code for opcode with single byte instruction or data for 2 byte instructions.
- **U(L)** Error for line L.

The first task carried out by this program is to load the desired 8748 mnemonic program into memory from the disk. The next step is to complete the first pass of the assembler. Each line of input is looked at in sequence. The first 8 spaces of the line are examined first to determine if that line has a label. If it does, the label is stored in memory for use by the second pass assembler in determining intra-program directives. The next step is to examine the first line for the opcode ORIG. If this code is present, the operand, which is the user's desired start address, is stored for use in beginning the byte count at that address. The program then checks each opcode in sequence to see if
it is a one or two byte instruction. The appropriate number is added to the present instruction address to determine the next instruction address. When the opcode END is recognized the first pass is completed and no address is assigned to that line. If the opcode END is not present, the program merely exits to the second pass assembler after the line count number reaches that number which was passed from the editor.

As each line is looked at and an address is assigned in the first pass, that line, with numbering, is printed on the CRT to keep the operator aware of assembly progress. The format for this printout is as follows:

<table>
<thead>
<tr>
<th>LINE NUMBER</th>
<th>HEX ADDRESS</th>
<th>MNEMONIC CODE</th>
<th>COMMENTS</th>
</tr>
</thead>
</table>

Upon completion of this phase, FIRST PASS COMPLETED is printed on the CRT and the second pass of the assembler begins automatically. The task of the second pass is two-fold. First, the mnemonic opcode and operand are converted to the appropriate machine language in hex code. Second, each time an operand is located which requires an address, the label representing that desired address is searched for in the list of labels formerly made up in the first pass. When the label is located, the corresponding address is used as the second byte of the calling two byte instruction. As an example, consider the following lines of program:

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Code</th>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>03</td>
<td>0407</td>
<td></td>
<td>JMP</td>
<td>BACK</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>8909</td>
<td></td>
<td>MOV</td>
<td>R1,#9</td>
</tr>
<tr>
<td>05</td>
<td>07</td>
<td>59</td>
<td>BACK</td>
<td>ANL</td>
<td>A,R1</td>
</tr>
</tbody>
</table>
In this example the programmer desires to jump to line 5 upon executing line 3. When the second pass reaches line 3 it first recognizes the opcode JMP and assigns the appropriate hex code of 04. The assembler then looks for what address to JMP (jump) to and looks at the operand BACK. The program then searches through the labels tabulated during the first pass and locates BACK. It then brings the address associated with that label, 07; back to add onto the JMP code to form the two byte instruction 0407 as seen above. The code column in this example is not added until the second pass is actually completed.

The actual search process of the second pass is done in two steps to increase speed of execution. Except for a few singular instructions, the first look is at 38 groups of instructions by type. Once the opcode group heading is recognized, the assembler jumps to a subroutine which assigns the specific hex code for that opcode and operand. Upon completion of that step the assembler returns to the beginning of the opcode list to begin again. An attempt has been made to arrange the opcode groups so that the more frequently used will be at the top of the list to provide faster average locating speed. During this process the assembler also identifies errors which are filed for display when assembly is completed. The recognized errors and meanings are listed below.

SYNTAX ERROR

Opcode or operand are not recognized. Probably an incorrect format or misspelled.
DATA EXCEEDS BYTE SIZE  A number greater than 255 is being used.
REGISTER SIZE EXCEEDS 7  Use of a non-allowed register.
R EXCEEDS 1  Register should be 0 or 1 only.
INCORRECT PORT #  Use of port not allowed in that instruction.

During execution of the second pass, as the code for each line is generated, it is presented on the CRT. Again, this presentation is provided to the operator so that he can follow the progress of assembly. After assembly is completed, that fact will be noted on the screen along with the statement 0 ERRORS, or the number of errors followed by the line number of each error and the error found in that line. If the line printer is connected and turned on prior to the end of assembly, a printout will be provided which will list the following data for the entire program:

<table>
<thead>
<tr>
<th>Line No.</th>
<th>Hex Address</th>
<th>Hex Code</th>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
</table>

See Appendix J for a sample assembled program printout.

Errors that were detected will be printed below the line affected. At the end of this printout a tabulation is provided for reference listing the labels and the address they are located at. If no errors were detected, the program then loads the object code (machine language hex code) onto the disk and calls the program PROGRAM for the purpose of programming the assembled code into an 8748 EPROM. This code is filed under the name assigned by the user but with
an "0" appended to the name after the last letter. This 0, of course, represents object code.

If upon completion of the assembly and printout, errors had been detected, the editor program is called and run to enable the operator to correct his mistakes.

Under the circumstances where the operator had either accidentally or deliberately failed to connect the TRS-80 to the line printer and to turn the printer on, the system will "freeze up" after assembly is completed. The recovery procedure is to press the "BREAK" key until READY appears on the screen. The system is now in BASIC mode. If errors have been detected and the user wants to examine the line numbers in which errors existed, he can enter the command RUN and use the shift key and @ key simultaneously to stop the scroll of assembled lines to check errors. To return to the edit mode again, simply allow the assemble program to continue to run until it calls the edit program, or press the break key to return to BASIC mode and type RUN "EDTASM".

It should be noted that if ASSEMBLE runs to completion and does not locate any errors it will automatically load the object file onto the disk before proceeding. If any errors are located, however, the object code will not be saved since it is not correct.

C. PROGRAMMER PROGRAM

The purpose of this program is to enable the system user to load his assembled program into the EPROM of an 8748
This is accomplished through associated hardware which is discussed in the next chapter. This program and hardware also enables the user to read an 8748 EPROM in order to verify its contents. The flowchart for the program named "program" is located in Appendix D and the program listing is Appendix I.

This program is normally executed upon successful completion of the assembly program, but may also be entered directly from BASIC mode by typing RUN "PROGRAM". Before running this program, however, the programmer assembly must be connected to the TRS-80 I/O bus. This connection should not be attempted while a program is running because electrical transients may be generated which could halt program execution.

When the program is initiated it will first present the statement ENTER PROGRAM MODE. The user may then select one of the following commands:

- **STOP**: Ends program execution and returns to BASIC mode.
- **EDIT**: Loads and runs the editor program.
- **RPROM**: Used to read an EPROM.
- **WPROM (filename0)**: Used to write to an EPROM. The "0" must be added to the filename to designate the object file.

Any other command will result in the statement ILLEGAL COMMAND--TRY AGAIN being presented on the CRT.

Whether the WPROM or RPROM mode is selected, the same setup routine is used. This routine first asks the question

36
START ADDRESS IN DECIMAL? to which the operator answers with a decimal number indicating the EPROM address in the 8748 he desires the reading or writing to start at. The next question will be END ADDRESS IN DECIMAL? which asks for the last EPROM address to be read or written to. If the end address entered is greater than the start address, the statement ILLEGAL ADDRESS will appear on the screen followed by the start address question again. Likewise, if either the start or end address are greater than the 1024 byte capability of the EPROM, the statement ILLEGAL ADDRESS will again appear. If the first two questions are answered satisfactorily the next question EPROM SOCKET EMPTY? (YES OR NO) will appear. This is to ensure that the EPROM is not inserted in the socket before power is applied and initial setup is completed. Other actions could result in damage to the 8748 chip. If the answer to the socket empty question is YES, the next question will be IS POWER SWITCH ON? (YES OR NO). If power is not yet on this is the time to turn it on. When the answer to this question is YES the program sets the hardware to the required initial conditions and prints the hexadecimal code of the object program about to be loaded on the CRT. The statement INSERT 8748 CHIP AND TYPE-GO: is then presented. It is especially important here that the 8748 chip not be inserted incorrectly in the socket as considerable damage to the chip would result. Once the command GO is typed and entered, the program will
return to either the RPROM or WPROM routine originally selected.

If RPROM had been selected, the desired address of the EPROM would be queried and the data at those addresses input to program memory. Since this data is input to the BASIC mode in decimal format, it must be converted to hexadecimal before presenting to the operator on the CRT. This presentation is made in the following format:

Decimal Address XX XX XX XX XX XX XX XX XX XX XX XX XX

where the decimal address is the address of the first instruction in that row and XX is a hexadecimal representation of the machine language in that address. When all desired addresses have been printed, any spaces left in that row will be filled with 00 and the program will reinitialize the electronics in the programmer and provide the message REMOVE EPROM NOW--THEN TURN POWER OFF. This is to ensure power is not turned off before the EPROM is removed. The program then returns to start with the statement ENTER PROGRAM MODE.

If the user had originally selected WPROM and given the correct name of the object code file on disk, the first step would have been the loading of the object file into TRS-80 memory. The initialization routine discussed above would then have been completed and the programming process commenced. The program is loaded into the EPROM one byte at a time. Each address requires approximately 100 milli-
seconds to program so the user should expect about one second of programming time for each 10 instructions. During the 100 msec cycle the program first inserts data into the given address and then reads that same address. The decimal number read is compared by the program with the number which should have been programmed. If the two numbers are not identical the EPROM programming ceases and the statement PROGRAMMING ERROR--ERASE EPROM AND TRY AGAIN will appear, followed by REMOVE EPROM NOW--THEN TURN POWER OFF and a return to program start. If this does occur, the most likely cause is that the EPROM was not thoroughly erased before programming. It is also possible, however, that the EPROM is defective or that the programmer is operating incorrectly. Check also to see that programming power is turned on.

If no errors are detected in the verification routine, the statement PROGRAMMING COMPLETED SATISFACTORILY will appear followed shortly by REMOVE EPROM NOW--THEN TURN POWER OFF and a return to program start.

If the user desires to run several RPROMs in succession or an RPROM followed by a WPROM or vice versa, it is not necessary to remove and reinsert the 8748 chip each time. Simply ignore the command to remove the EPROM and proceed with the steps in order. The questions EPROM SOCKET EMPTY? and POWER SWITCH ON? may both be answered YES with no ill effects. The important point to remember is to not remove
the EPROM until the second to last statement on the CRT is
REMOVE EPROM NOW--THEN TURN POWER OFF.
V. PROGRAMMER HARDWARE AND OPERATION

The design for the hardware portion of the 8748 EPROM programmer was based on the requirements set forth in Ref. 1. Reference 7 provided additional assistance in switching circuit techniques and methodology. Figures 7 and 8 show the schematic for the final programmer design. Figure 9 shows component arrangement on the circuit board.

In first examining the possibilities for interfacing the TRS-80 with an EPROM programmer, the question of output procedure arose. The TRS-80 is able to provide an output via either memory mapped or port selection modes. Since the memory mapped system requires memory addresses to be used for output and input, this method had to be rejected. With all possible 64K of addresses already in use for either RAM or ROM, much confusion could result. The port based system allows only 255 possible ports, but this is more than sufficient if much of the work is done by the TRS-80 software rather than programmer hardware. In fact, for this 8748 programmer only 4 ports are needed. In the port system the commands used are OUT (port), (value) and INP (port) in the BASIC language. The OUT command sets the OUT line low, simultaneously putting the port number on the lower eight address lines and the desired value on the eight data lines (data bus). Likewise, the INP command sets the IN line low while outputting the port number on the lower eight
Figure 8 - Programmer Schematic
Figure 9 - Programmer Circuit Board
address lines. The data present on the data bus is then read into the TRS-80.

The first major obstacle to overcome in preparing the design of the programmer was to verify the correct pinout from the TRS-80 I/O port.

Since Ref. 8 has no data on the TRS-80 expansion interface it was largely up to the author to verify the pinout. In connecting up to the I/O port a flat cable 40 pin connector or left over from another TRS-80 application was used. After some time was spent searching for signals out of this cable it was discovered that the cable and connectors are wired to reverse the signal from top to bottom. That is, the top row of signals in one end of the flat cable comes out on the bottom row at the other end and vice versa. The similar cable coming from the computer/keyboard to the expansion interface of the TRS-80 also reverses the signals top to bottom. The interface board is wired to again reverse the signals so they are upright coming out of the interface I/O port. Figure 10 shows the pinout of the expansion interface port.

Once the proper pinout had been verified, the design and construction of the programmer board could begin. Because of their ready availability, it was decided to build the system on a 4 x 6 Vector plugboard with a 44 pin connector and which was predrilled for wirewrap sockets. This board provided for a compact unit easy to interface via 44
<table>
<thead>
<tr>
<th>P/N</th>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RAS*</td>
<td>Row Address Strobe Output for 16-Pin Dynamic Rams</td>
</tr>
<tr>
<td>2</td>
<td>SYSRES*</td>
<td>System Reset Output, Low During Power Up Initialize or Reset Depressed</td>
</tr>
<tr>
<td>3</td>
<td>CAS*</td>
<td>Column Address Strobe Output for 16-Pin Dynamic Rams</td>
</tr>
<tr>
<td>4</td>
<td>A10</td>
<td>Address Output</td>
</tr>
<tr>
<td>5</td>
<td>A12</td>
<td>Address Output</td>
</tr>
<tr>
<td>6</td>
<td>A13</td>
<td>Address Output</td>
</tr>
<tr>
<td>7</td>
<td>A15</td>
<td>Address Output</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>9</td>
<td>A11</td>
<td>Address Output</td>
</tr>
<tr>
<td>10</td>
<td>A14</td>
<td>Address Output</td>
</tr>
<tr>
<td>11</td>
<td>A8</td>
<td>Address Output</td>
</tr>
<tr>
<td>12</td>
<td>OUT*</td>
<td>Peripheral Write Strobe Output</td>
</tr>
<tr>
<td>13</td>
<td>WR*</td>
<td>Memory Write Strobe Output</td>
</tr>
<tr>
<td>14</td>
<td>INTAK*</td>
<td>Interrupt Acknowledge Output</td>
</tr>
<tr>
<td>15</td>
<td>RD*</td>
<td>Memory Read Strobe Output</td>
</tr>
<tr>
<td>16</td>
<td>MUX</td>
<td>Multiplexer Control Output for 16-Pin Dynamic Rams</td>
</tr>
<tr>
<td>17</td>
<td>A9</td>
<td>Address Output</td>
</tr>
<tr>
<td>18</td>
<td>D4</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>19</td>
<td>IN*</td>
<td>Peripheral Read Strobe Output</td>
</tr>
<tr>
<td>20</td>
<td>D7</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>21</td>
<td>INT*</td>
<td>Interrupt Input (Maskable)</td>
</tr>
<tr>
<td>22</td>
<td>D1</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>23</td>
<td>TEST*</td>
<td>A Logic &quot;Y&quot; on TEST* Input Tri-States A0-A15, D0-D7, WR*, RD*, IN*, OUT*, RAS*, CAS*, MUX*</td>
</tr>
<tr>
<td>24</td>
<td>D6</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>25</td>
<td>A9</td>
<td>Address Output</td>
</tr>
<tr>
<td>26</td>
<td>D3</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>27</td>
<td>A1</td>
<td>Address Output</td>
</tr>
<tr>
<td>28</td>
<td>D5</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>29</td>
<td>GND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>30</td>
<td>D0</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>31</td>
<td>A4</td>
<td>Address Bus</td>
</tr>
<tr>
<td>32</td>
<td>D2</td>
<td>Bidirectional Data Bus</td>
</tr>
<tr>
<td>33</td>
<td>WAIT*</td>
<td>Processor Wait Input, to Allow for Slow Memory</td>
</tr>
<tr>
<td>34</td>
<td>A3</td>
<td>Address Output</td>
</tr>
<tr>
<td>35</td>
<td>A6</td>
<td>Address Output</td>
</tr>
<tr>
<td>36</td>
<td>A7</td>
<td>Address Output</td>
</tr>
<tr>
<td>37</td>
<td>GND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>38</td>
<td>A8</td>
<td>Address Output</td>
</tr>
<tr>
<td>39</td>
<td>GND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>40</td>
<td>A2</td>
<td>Address Output</td>
</tr>
</tbody>
</table>

NOTE: *means Negative (Logical "Y") True Input or Output

Figure 10 - Pinout of Expansion Interface
pin sockets. The use of wirewrap techniques on the board enabled the system to be put together rapidly and yet to provide a high degree of reliability.

Intel 8212 I/O port chips were selected for the programmer unit because of their versatility and compactness. These 8 bit ports can be wired to operate in a number of different ways since they include both tri-state buffers on the output lines and latches on the input lines. An Intel schematic for the 8212 chip is provided in Figure 11.

The next major design hurdle for the programmer hardware was the higher voltage switching circuits. This circuitry can be seen in programmer schematic diagram, Figure 7. While several of the drive signals to the 8748 chip such as TESTO and RESET require 0 and +5 volts for off and on, the program functions of EA, $V_{DD}$ and PROG require 23, 25, and 23 volts respectively as the high input. In fact, EA and $V_{DD}$ also require a low of +5 volts while PROG must have a low of 0 volts as well as a "float" condition. The output of the 8212 chip is easily able to provide a direct 0 to 25 volt transition with the assistance, in some cases, of a pull-up resistor. Since no digital chips provide the range of 0 to 25 volts required for the program function, it was necessary to construct separate circuitry using switching transistors. The design of all 3 of the 25 volt switching circuits was basically the same. A pair of transistors, one PNP 2N3906 and one NPN 2N3904, were tied together at their collectors and driven by a common voltage into their bases. In this
Figure 11 - Intel 8212 I/O Port Schematic
configuration, when the 2N3906 PNP transistor has a ground level at its base, current flows to ground and the transistor is switched on, allowing +25.4 volts at its collector output. At the same time a low on the base of the 2N3904 shuts it off, thereby directly all current into the collector output connection. For the reverse condition, when the 3906 is shut off and the 3904 is turned on, no current is provided by the transistor circuitry. Instead, in the case of $V_{DD}$ and EA, 5 volts is provided at the appropriate input pin from the +5 volt supply. The emitter of the 3904 is also tied to +5 volts to ensure rapid switching from +25.4 to +5 volts. While the $V_{DD}$ high operating voltage is set at 25.4 volts (allowable range is 24 to 26 volts) the EA high operating voltage is 23 volts (allowable range is 21.5 to 24.5 volts). Rather than providing two different power supplies, the 23 volts is reached, for both EA and PROG, by dropping the 25.4 volts across 3 IN 753A diodes in series. With a $V_o$ of .8 volts, the resultant 2.4 volt drop enables the desired voltage to be achieved.

The switching circuitry for the PROG input is necessarily somewhat different from the other two because of the requirement for 3 states, namely +23 volts, ground, and floating. This is achieved by utilizing two inputs from the TRS-80 rather than one. One input controls the switching of the 2N3904 transistor and another controls the 2N3906. When the 3906 is on and the 3904 is off, 23 volts will be present at the output. When the 3906 is off and the 3904 is also off,
no current supply or drain will exist and the input may seek its own level (about 4 volts under operating conditions). When the 3906 is off and the 3904 is on, the PROG input will be tied to ground.

The operating sequence of one complete cycle of the system as a whole is illustrated below for one program pulse. To assist the reader in following this discussion, the 8 bit output of port 4 is listed below. Underneath each bit of port 4 is listed the item controlled by that bit.

<table>
<thead>
<tr>
<th>bit 8</th>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_DD</td>
<td>RESET</td>
<td>TESTO</td>
<td>EA</td>
<td>PROG1</td>
<td>PROG2</td>
<td>N.C.</td>
<td>PORT 1 MD</td>
</tr>
</tbody>
</table>

Before the 8748 is allowed in the socket of the programmer, the hex code 36, binary 00100100, is output to port 4. This provides a +5 volts to PROG2 and +5 volts to TESTO with all other port outputs equal to 0. The result is proper initial conditions for the 8748 to be inserted in the socket, i.e. PROG is floating and TESTO is high. The next action is to output hex 04. This switches TESTO to 0 volts to begin the programming process. The next step is to output 20 to port 4 which switches EA to +23 volts. The high 2 bits of the desired 8748 address are then output to port 2. Port 1 is switched to the latch-on condition next by outputting a 21 to port 4. The lower 8 bits of the desired address are now passed to port 1. An 81 is next output to port 4 which turns RESET off with a +5 volts and turns PROG to the 0 volt (ground) condition. This latches the address onto the 8748 bus. The data is then passed out to port 1 followed by a
209 to port 4 to raise $V_{DD}$ to 25.4 volts, a 221 to port 4 to raise PROG to +23 volts, and a 50 msec delay to allow the programming to occur. PROG is then taken low by outputting 209 to port 4. $V_{DD}$ is then lowered to +5 volts by sending an 81 out to port 4. $V_{DD}$ is then lowered to +5 volts by sending an 81 to port 4. An 84 is next put out to place PROG back in a floating condition and set up for the verify. The port 1 tristate is also turned off at this time by the same step. A 116 is then placed on the bus to port 4 to raise TESTO to +5 volts. This action places the data at the current 8748 address onto the 8748 output bus where it is read into the TRS-80 using the BASIC command INP(3) to bring it in via port 3. The software then checks for correct programming of that address and outputs a 20 to port 4 to lower RESET and TESTO to 0 volts. The cycle then begins again with the output of the high address and continues until the programmer's last desired address is reached.

The RPROM routine is basically similar to the above except that no $V_{DD}$ or PROG pulses are used and no data is output to the 8748. Instead, the addresses are put out to the chip and the verify procedure follows immediately. A picture of the actual timing diagram for this process is available as Figure 12. These waveforms correspond favorably with the required traces as seen on page 6-8 of Ref. 1.
Figure 12 - Programmer Timing Diagram
VI. PROGRAMMER POWER SUPPLIES

The power for the 8748 programmer is drawn from two separate power supplies contained within the programmer enclosure. These supplies are not shared with the TRS-80 power supplies. The right hand circuit board within the enclosure was a pre-built supply available from another student's project. It is powered by a 26 volt transformer and produces 3 separate DC voltages using on-board rectifiers and LM 723 voltage regulators. The DC levels available from this supply are +5, +12, and -12 volts. Variable resistors are available for each supply to allow adjusting of output voltages by approximately ±1 volt. In this application only the +5 volts is required and it is hard wired to the motherboard. The +12 and -12 supplies are available for future student projects. In fact, if a 2708 programmer is constructed, the +12 volt and a -5 volt supply drawn from the -12 volts will be required.

The other power supply board in the programmer enclosure provides a regulated 25.4 volts DC for the EPROM programming pulses. This supply was constructed by the author. Figure 13 is a schematic diagram of the unit which uses an LM-317 for voltage regulation.

Current requirements for the 8748 programmer are fairly significant for the +5 volt supply. In excess of 500 milliamps is required to power the board. Much of this current is
Figure 13 - 25.4 Volt Power Supply
taken up by the 8212 I/O port chips which operate at a higher than ambient temperature. The 25.4 volt supply current requirement is less than 40 milliamps and then only during actual programming.

A schematic is not provided of the mother board since it is simply an extension from the TRS-80 expansion interface output port. It should be noted, however, that the printed circuit lines on the top of the mother board and on the right side of the 44 pin sockets represent the bottom row of output pins from the expansion interface. Likewise, the top row outputs from the interface are on the bottom of the printed circuit board and the left side of the sockets. The 4 outside lines on the mother board are not connected to the TRS-80 40 pin connector and are intended for use in supplying power from the programmer power supplies. Additionally, it should be noted that all lines from the TRS-80 are connected to only the first 44 pin socket. The left most 3 sockets have only the data lines, lower 8 address lines, ground, OUT and IN. The reason for this is that these signals are the only ones required to provide port mode input and output. Thus, they will suffice for most applications.

To provide maximum protection for the programmer circuitry, 3 fuses have been installed. A 120 volt, 2 amp fuse is located in the input power line to protect against major transformer failure. A 750 milliamp fuse is inserted in the 5 volt supply line and a 3/8 amp fuse is in the 25 volt programming supply line to protect the programmer board.
against major damage in the event of overload.

The light on the front panel of the programmer assembly indicates that 120 volts have been received past the 120 volt fuse. The fuses for the 5 and 25 volt supplies must be checked visually if the programmer is not operating correctly.
VII. CONCLUSION

The microprocessor development system discussed in this thesis is already in use by several groups of students who are employing the 8748 chip in various applications. These users have been favorably impressed with the effectiveness and simplicity of the system, especially when compared with the Tektronix 8002 MDS. While the 8002 has many more capabilities, its sophistication is at such a high level that the beginning student in microprocessors must spend many hours learning how to use it. The typical requirements for an MDS, to edit, assemble and program EPROMs, are more than met by the author's system.

Perhaps an even more significant difference between the TRS-80 based system and the more sophisticated systems is cost. The TRS-80 in its present configuration is available for about $2000, while the Tektronix system costs over $15,000. In fact, the 8748 assembler module alone for the 8002 is worth $850 with the complete emulator card and probe raising that price to over $4,000.

In constructing the programmer assembly for the 8748, additional sockets were provided in a 4 socket mother board. These sockets accept the standard 44 pin Vector plugboard. It is recommended that future student projects and thesis work be directed toward the construction of software and hardware to expand this MDS. Some additions which might prove useful to 8748 users would be an in-circuit-emulator
and a debugger to allow for real time execution of the user's program in TRS-80 software. Other possible projects could be a programmer for the Intel 2708 and 2716 or other commonly used EPROMs. With little additional effort a parallel to the already installed capabilities for the 8748 could be included to provide for the 8080, Z-80, 6800, and other popular microprocessors. Since the editor program would work for any microprocessor language, the task of program building would be limited to only the assembler program. Thus, a new capability for the TRS-80 MDS would consist of only a $5 magnetic disk to store the programs.

Other capabilities could be added to the digital laboratory by interfacing the TRS-80 to the Tektronix 8002 to allow for exchanging programs and data between the two. Additionally, the TRS-80 might be used as a real time processor for the IBM-360 or other main frame computer, allowing for the transfer of programs and data between several computers.

With the advent of smaller, faster, and more capable microprocessors, the age of truly distributed processing systems is upon us. Additionally, microprocessor systems of the future will be the equivalent in capability of main frame computers of the past. For these reasons the importance of understanding the capabilities and limitations of microprocessors and microcomputers cannot be over emphasized. The microprocessor development system is one necessary and concrete step toward this goal. To be able to rapidly and
effectively program and utilize the microprocessor is the raison d'etre of the microprocessor development system and is, not coincidentally, the path of the future.
### APPENDIX A

**PROGRAM LISTING FOR PRINTER SUBROUTINE**

<table>
<thead>
<tr>
<th>Location</th>
<th>Hex Code</th>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF00</td>
<td>E5</td>
<td>INIT</td>
<td>PUSH</td>
<td>HL</td>
</tr>
<tr>
<td>FF01</td>
<td>C5</td>
<td></td>
<td>PUSH</td>
<td>BC</td>
</tr>
<tr>
<td>FF02</td>
<td>F5</td>
<td></td>
<td>PUSH</td>
<td>AF</td>
</tr>
<tr>
<td>FF03</td>
<td>3A48FF</td>
<td></td>
<td>LD</td>
<td>A,(FLAG)</td>
</tr>
<tr>
<td>FF06</td>
<td>FE01</td>
<td></td>
<td>CP</td>
<td>01H</td>
</tr>
<tr>
<td>FF08</td>
<td>2820</td>
<td></td>
<td>JR</td>
<td>Z,RESTOR</td>
</tr>
<tr>
<td>FF0A</td>
<td>3E01</td>
<td></td>
<td>LD</td>
<td>A,01H</td>
</tr>
<tr>
<td>FF0C</td>
<td>3248FF</td>
<td></td>
<td>LD</td>
<td>(FLAG),A</td>
</tr>
<tr>
<td>FF0F</td>
<td>D3E8</td>
<td></td>
<td>OUT</td>
<td>E8,A</td>
</tr>
<tr>
<td>FF11</td>
<td>DBE9</td>
<td></td>
<td>IN</td>
<td>AE9</td>
</tr>
<tr>
<td>FF13</td>
<td>E6F8</td>
<td></td>
<td>AND</td>
<td>0F8H</td>
</tr>
<tr>
<td>FF15</td>
<td>F604</td>
<td></td>
<td>OR</td>
<td>04H</td>
</tr>
<tr>
<td>FF17</td>
<td>3247FF</td>
<td></td>
<td>LD</td>
<td>(SWTIMG),A</td>
</tr>
<tr>
<td>FF1A</td>
<td>D3EA</td>
<td>BAUDST</td>
<td>IN</td>
<td>A,E9</td>
</tr>
<tr>
<td>FF1C</td>
<td>DBE9</td>
<td></td>
<td>AND</td>
<td>07H</td>
</tr>
<tr>
<td>FF20</td>
<td>213FFF</td>
<td></td>
<td>LD</td>
<td>HL,BDTABL</td>
</tr>
<tr>
<td>FF23</td>
<td>0600</td>
<td></td>
<td>LD</td>
<td>B,00H</td>
</tr>
<tr>
<td>FF25</td>
<td>4F</td>
<td></td>
<td>LD</td>
<td>C,A</td>
</tr>
<tr>
<td>FF26</td>
<td>09</td>
<td></td>
<td>ADD</td>
<td>HL,BC</td>
</tr>
<tr>
<td>FF27</td>
<td>7E</td>
<td></td>
<td>LD</td>
<td>A,(HL)</td>
</tr>
<tr>
<td>FF28</td>
<td>D3E9</td>
<td>RESTOR</td>
<td>OUT</td>
<td>E9,A</td>
</tr>
<tr>
<td>FF2A</td>
<td>F1</td>
<td></td>
<td>POP</td>
<td>AF</td>
</tr>
<tr>
<td>FF2B</td>
<td>C1</td>
<td></td>
<td>POP</td>
<td>BC</td>
</tr>
<tr>
<td>FF2C</td>
<td>E1</td>
<td></td>
<td>POP</td>
<td>HL</td>
</tr>
<tr>
<td>FF2D</td>
<td>DBEA</td>
<td>STATIN</td>
<td>IN</td>
<td>A,EA</td>
</tr>
<tr>
<td>FF2F</td>
<td>CB77</td>
<td></td>
<td>BIT</td>
<td>6,A</td>
</tr>
<tr>
<td>FF31</td>
<td>28FA</td>
<td></td>
<td>JR</td>
<td>Z,STATIN</td>
</tr>
<tr>
<td>FF33</td>
<td>79</td>
<td>LD</td>
<td>A,C</td>
<td></td>
</tr>
<tr>
<td>FF34</td>
<td>D3EB</td>
<td></td>
<td>OUT</td>
<td>EB,A</td>
</tr>
<tr>
<td>FF36</td>
<td>FEOD</td>
<td></td>
<td>CP</td>
<td>0DH</td>
</tr>
<tr>
<td>FF38</td>
<td>2004</td>
<td></td>
<td>JR</td>
<td>NZ,RETRN</td>
</tr>
<tr>
<td>FF3A</td>
<td>C350FF</td>
<td></td>
<td>JMP</td>
<td>FF50</td>
</tr>
<tr>
<td>FF3D</td>
<td>00</td>
<td></td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>FF3E</td>
<td>C9</td>
<td>RETRN</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>FF3F</td>
<td>22</td>
<td>BDTABL</td>
<td>DEFB</td>
<td>22H</td>
</tr>
<tr>
<td>FF40</td>
<td>44</td>
<td></td>
<td>DEFB</td>
<td>44H</td>
</tr>
<tr>
<td>FF41</td>
<td>55</td>
<td></td>
<td>DEFB</td>
<td>55H</td>
</tr>
<tr>
<td>FF42</td>
<td>66</td>
<td></td>
<td>DEFB</td>
<td>66H</td>
</tr>
<tr>
<td>FF43</td>
<td>77</td>
<td></td>
<td>DEFB</td>
<td>77H</td>
</tr>
<tr>
<td>FF44</td>
<td>AA</td>
<td></td>
<td>DEFB</td>
<td>0AAH</td>
</tr>
<tr>
<td>FF45</td>
<td>CC</td>
<td></td>
<td>DEFB</td>
<td>0CCH</td>
</tr>
<tr>
<td>FF46</td>
<td>EE</td>
<td></td>
<td>DEFB</td>
<td>0E6H</td>
</tr>
<tr>
<td>FF47</td>
<td>00</td>
<td>SWTIMG</td>
<td>DEFB</td>
<td>00H</td>
</tr>
<tr>
<td>FF48</td>
<td>00</td>
<td>FLAG</td>
<td>DEFB</td>
<td>00H</td>
</tr>
<tr>
<td>FF49</td>
<td>00</td>
<td></td>
<td>NOP</td>
<td></td>
</tr>
</tbody>
</table>

60
<table>
<thead>
<tr>
<th>Address</th>
<th>Hex</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF50</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF51</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF52</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF53</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF54</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF55</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF56</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>FF57</td>
<td>E5</td>
<td>PUSH HL</td>
</tr>
<tr>
<td>FF58</td>
<td>21FF44</td>
<td>LD HL, 44</td>
</tr>
<tr>
<td>FF59</td>
<td>2B</td>
<td>DEC HL</td>
</tr>
<tr>
<td>FF5A</td>
<td>7C</td>
<td>LD A, H</td>
</tr>
<tr>
<td>FF5B</td>
<td>B5</td>
<td>OR L</td>
</tr>
<tr>
<td>FF5C</td>
<td>C254FF</td>
<td>JP NZ, FF54</td>
</tr>
<tr>
<td>FF5D</td>
<td>E1</td>
<td>POP HL</td>
</tr>
<tr>
<td>FF60</td>
<td>OEOA</td>
<td>LD C, OA H</td>
</tr>
<tr>
<td>FF61</td>
<td>C32DFF</td>
<td>JP FF2D</td>
</tr>
</tbody>
</table>
APPENDIX B

FLOWCHART FOR EDITOR PROGRAM

"EDTASM"

LOAD PRINTER PROGRAM FROM DISK AND EXECUTE

RUN "MASTER"

PRINT "8748 EDITOR/ASSEMBLER ON LINE"

1

INPUT "ENTER MODE SELECTION"

LPT1?

NO

INPUT?

NO

EDIT?

NO

GET?

NO

PRINT?

NO

ASM?

YES

PRINT PROGRAM ON LINE PRINTER

BRING PREVIOUSLY FILED PROGRAM FROM DISK

PRINT ALL LINES IN MEMORY ON CRT

PRINT ILLEGAL ENTRY

STORE FILE NAME ON DISK IN "STORE"

RUN "ASSEMBLE"

PRINT?
PRINT "EDIT"

INPUT COMMAND

UP?
  YES MOVE POINTER UP ONE
  NO

DN?
  YES MOVE POINTER DOWN ONE
  NO

DEL?
  YES DELETE CURRENT LINE
  NO

INS?
  YES INSERT NEW LINE ABOVE CURRENT LINE
  NO

FILE?
  YES SAVE ALL LINES IN MEMORY ON DISK
  NO

INPUT?
  YES A
  NO

EDIT?
  YES
  NO C
PRINT? YES
   PRINT ALL LINES IN MEMORY ON CRT
   NO

   L? YES
   LOCATE LINE WITH SPECIFIED SUBSTRING
   NO

   C? YES
   CHANGE OLD SUBSTRING TO NEW SUBSTRING
   NO

   TOF? YES
   POINTER TO TOF OF FILE
   NO

   EOF? YES
   POINTER TO END OF FILE
   NO

   PRINT "ILLEGAL ENTRY"
A

PRINT "INPUT"

LINE INPUT

QUIT? YES NO

EDIT? YES NO

FILE? YES NO

SAVE ALL LINES IN MEMORY ON DISK

STORE AND ADVANCE TO NEXT LINE NUMBER

NO

1
APPENDIX C
FLOW CHART FOR ASSEMBLY PROGRAM

"ASSEMBLE"

GET FILE NAME FROM "STORE"

LOAD LINE COUNT FROM FILE

DIMENSION VARIABLES

LOAD FIRST LINE FROM DISK

IS A LABEL PRESENT?

YES

SET FIRST ADDRESS TO OPERAND OF "ORIG"

NO

LAST LINE?

YES

FIRST LINE "PROG"?

YES

SET FIRST ADDRESS TO OPERAND OF "ORIG"

NO

A

NO

LOAD NEXT LINE
DECIMALE ADDES
TO
HEXPP
CONTAIN
NUBE
EX~ ADDRSSEABE
OPCODE
CONTAIN
#?
OPCODE
CONTAIN
CALL?
ADD 1 to
PREVIOUS ADDRESS
ADD 2 TO
PREVIOUS ADDRESS
CONVERT
DECIMAL ADDRESS
TO HEX
PRINT LINE NUMBER
HEX ADDRESS, LABEL
OPCODE, OPERAND AND
COMMENTS
LOOK AT NEXT
LINE
LAST LINE?
PRINT "FIRST
PASS COMPLETED"
B
67
GET FIRST LINE

DELETE ALL COMMENTS

ASSIGN HEX CODE TO OPCODE AND OPERAND

ERROR DETECTED?

YES

INCREMENT ERROR COUNT

STORE ERROR AT LINE NUMBER

NO

PRINT HEX CODE

GET NEXT LINE

LAST LINE?

PRINT "ASSEMBLY COMPLETED"

PRINT NUMBER OF ERRORS
C

NO

ERRORS=0?

PRINT ASSEMBLED PROGRAM AT LINE PRINTER

YES

NO

ERRORS=0?

PRINT HEADING "LINE ERROR"

RETURN TO LINE 1

ERROR IN THIS LINE?

NO

PRINT LINE NUMBER AND ERROR

LAST LINE?

NO

GET NEXT LINE

YES

RUN "MASTER"

STORE OBJECT CODE ON DISK UNDER FILENAME + 0

RUN "PROGRAM"
A

LOAD OBJECT CODE FILE

CONVERT HEX CODE TO DECIMAL

CALL SETUP SUBROUTINE

OUTPUT FIRST LINE OF CODE TO DESIRED ADDRESS

READ CODE AT SAME ADDRESS IN

CODE OUT = CODE IN?

NO

PRINT "ERROR-ERASE EPROM AND TRY AGAIN"

YES

OUTPUT CODE TO CRT

ADVANCE TO NEXT LINE

LAST LINE IN?

NO

PRINT "PROGRAMMING COMPLETED SATISFACTORILY"

YES

INITIALIZE PROGRAMMER

PRINT "REMOVE EPROM NOW-THEN TURN POWER OFF"

2

1

71
CALL SETUP SUBROUTINE

READ EPROM FIRST DESIRED LINE

LAST DESIRED LINE READ?

YES

RESTORE PROGRAMMER TO INITIAL CONDITIONS

CONVERT DECIMAL NUMBERS TO HEX

PRINT HEX CODE IN BLOCK FORM

PRINT "REMOVE EPROM NOW- THEN TURN POWER OFF"

NO

READ NEXT LINE
SETUP SUBROUTINE

INPUT "START ADDRESS IN DECIMAL?"

>1023? YES

PRINT "ILLEGAL ADDRESS"

NO

INPUT "END ADDRESS IN DECIMAL?"

>1023? YES

NO

START > END? YES

NO

INPUT "EPROM SOCKET EMPTY? (YES OR NO)"

YES

NO?

YES

INPUT "IS SOCKET POWER SWITCH ON? (YES OR NO)"

PRINT "TURN POWER ON" YES

NO?

YES

NO

NO

INITIALIZE PROGRAMMER

INPUT "INSERT 8748 CHIP AND TYPE-GO:"

GO? NO

YES

RETURN

73
APPENDIX E

8748 EDITOR/ASSEMBLER OPERATING INSTRUCTIONS

This development system operates under its own set of
instructions and commands and no knowledge of the TRS-80
operating system or the BASIC language is presumed or neces-
sary.

To load the 8748 program proceed as follows:

1. Turn on the master power switch on the bus strip.
2. Turn the CRT on by pushing in the button in the upper right
   hand corner.
3. Turn the expansion interface on by pressing in on the
   button in the center front face of the unit.
4. Turn the disk drive on by placing the toggle switch on
   the rear of the drive unit up.
5. Insert the 8748 disk in the drive with the notch up and
   the label facing to the right. Close the disk drive
   door.
6. Turn the TRS-80 on by pressing in on the button located
   on the rear of the keyboard just to the left of the 3
   input cables.
7. The system will now load the disk operating system fol-
   lowed by the BASIC system. The screen will display the
   following:

   HOW MANY FILES?

   Answer this by typing a 1 and pressing ENTER.

   The next question on the screen will read

   MEMORY SIZE?

   Answer this with 65000 and ENTER.

   The system will then respond with

   RADIO SHACK DISK BASIC VERSION 2.2
   READY
Now type in RUN "EDTASM" and press ENTER.

This loads the 8748 editor/assembler and the screen will display

**8748 EDITOR ASSEMBLER ON LINE**

**ENTER MODE SELECTION--**

The operator is now ready to begin entering and editing his 8748 assembly level language program. The following commands and modes provide all the assistance necessary to provide a fully assembled version of his program.

**Modes available:**

- **INPUT**
  To enter program lines into system buffer.

- **PRINT**
  Prints contents of buffer on video display.

- **LPT1**
  Prints contents of buffer at the line printer.

- **EDIT**
  Provides for editing of lines in buffer. (See EDIT commands.)

- **GET (filename)**
  Transfers (filename) program from disk storage into buffer.

- **ASM (filename)**
  Assembles program named (filename) and provides complete printout at the line printer. (ensure that printer is connected to TRS-80 and turned on.)

**Edit Commands:**

- **UP**
  Moves pointer up one line in the buffer.

- **DN**
  Moves pointer down one line in the buffer.

- **TOF**
  Moves pointer to the top line in the buffer.
EOF

Moves the pointer to the end of the file in buffer.

DEL

Deletes the current line.

INS

Provides for a new line to be inserted above the current line.

L /xxx/

Locates the first line containing substring xxx and moves the pointer to that line.

C /xxx/yyyy/

Changes substring xxx to yyy in the current line.

FILE (filename)

Transfers the contents of buffer to disk storage under name (filename).

PRINT

Prints contents of the buffer on the video display.

INPUT

Puts the system in the input mode.

Input operations:

Enter new program lines in the following format:

Label

Left justified, 1 to 6 alphanumeric characters ending with a colon. (Leave blank if no label is desired.)

Tab to 8 by pressing + key.

Opcode

3 or 4 letter code from MCS-48 user's manual.

Tab to 16.

Operand

Alphanumeric as given in MCS-48 user's manual. Numbers must be in decimal. Locations must be a 1 to 6 digit label only.

Tab to 24.

Comments

If desired, type a semicolon followed by a short description of instruction operation.
In general

Press ENTER to move to the next line.

After the last program line type END as an opcode.

The opcode ORIG may be used in the first line with a
decimal number as the operand to direct the
assembler to place the beginning of the program
at that address.

Input Commands:

QUIT
Resets the line pointer to zero and
returns to the executive routine.

FILE (filename)
Transfers files from the buffer to
disk storage and returns to the execu-
tive routine.

EDIT
Transfers to the edit mode to allow
for program changes and corrections.

Errors

There are a number of unlikely but possible errors which
can be made which would result in the system's dropping out
of program RUN and back to the BASIC mode. If this occurs,
in every case the word READY will appear as the last word
on the CRT. To return to that portion of the program which
was in operation, simply type RUN and press ENTER. Un-
fortunately all files in the system buffer will be lost and
must be reentered. For this reason it is wise to periodically
save portions of the new program as they are being written.
This is done by the command FILE (filename). To continue
building on this program go to INPUT and continue writing.
Linking

Programs can be written by different authors or the same author with the eventual aim of combining into one large program for later assembly and execution. To combine two programs they must first be written and FILEd on the disk. Press the BREAK key and wait for the READY signal. Then type APPEND (filename 1) TO (filename 2), after which type KILL "(filename 1)". To return to the executive again for assembly type RUN. It will now be necessary to edit the new program by deleting the END and EOF from the end of the first subprogram.

Programming

Once the assembly of the user's 8748 program is completed with no errors detected, the programmer program will be automatically loaded. The following program modes are available:

- **STOP** Exits the program to BASIC mode.
- **EDIT** Returns system to editor/assembler.
- **RPROM** Used to read an 8748 EPROM.
- **WPROM (filename0)** Writes a program entitled (filename +0) to an 8748 EPROM. (0, for object code, must be appended to the original filename.)

Full questions and commands are provided by the program to prompt the user during the RPROM and WPROM modes.

Ensure that the 8748 chip is not inserted or removed from the socket except when so directed by the program.
Great caution must also be exercised to make certain the chip is not inserted incorrectly as this could result in severe damage to the 8748.

**Turn off sequence.** (Can be followed any time except when disk drive light is on or programmer has 8748 in the socket.)

1. Remove diskette from disk drive.
2. Turn off disk drive.
3. Turn off programmer.
4. Turn off keyboard/computer.
5. Turn off CRT and interface.
6. Turn off master power switch on power bus.

**NOTE:** The power transformers (2) in the interface unit are not affected by any power switches on the TRS-80 components. For this reason the master power to the plugs must be turned off or the transformers will continue to operate.
APPENDIX F

Printer Program - "EDTASM"

1 " THIS PROG PUTS THE SERIAL PRINTER DRIVER
     INTO MEM LOCATIONS FF60 - FF5F
2 " DCB: +0 => DCE TYPE PRINTER STARTS @ 4025
   1 => DRIVER ADDR LSB
   2 => DRIVER ADDR MSB
   3 => LINES/PAGE
   4 => LINE COUNTER
3 POKE 16421,2:POKE 16422,0:POKE 16423,255
4 FOR I = 0 TO 95
5 X=-256 +I
6 READ Y
7 POKE X,Y
8 NEXT I
9 RUN "MASTER"
10 END
11 DATA 229,197,245,58,72,255,254,1,40
12 DATA 32,62,1,50,72,255,211,219,233
13 DATA 230,248,246,4,50,71,255,211,234
14 DATA 219,233,230,7,33,63,255,6,0,79,9
15 DATA 126,211,233,241,193,225,219,234
16 DATA 203,119,40,250,121,211,235,254
17 DATA 13,32,4,195,80,255,80,201,34,68
18 DATA 85,132,119,170,204,238,0,0
19 DATA 0,0,0,0,0,0,229,33,255,66,43
20 DATA 124,181,194,84,255,225,14,10,195,45,255
APPENDIX G

Editor Program - "MASTER"

1 '****"MASTER"****
2 '-----EXECUTIVE ROUTINE-----
3 CLEAR 10000
4 DEFINT H-Q
5 LET STR W,X,Y,Z
6 DIM X(1000),W(1000)
7 CLS
8 PRINT TAB(15) "8748 EDITOR/ASSEMBLER ON LINE"
9 PRINT
10 J=0
11 Y= "G"O
12 LINE INPUT "ENTER MODE SELECTION--"; Y
13 IF Y="LPT" GOTO 67
14 IF Y="INPUT" GOTO 53
15 IF Y="EDIT" GOTO 27
16 IF LEFT$(Y,3)="GET" GOTO 83
17 IF Y>"PRINT" GOTO 15
18 GOSUB 119 : GOTO 11
19 IF LEFT$(Y,3)<"ASM" GOTO 24
20 Z=MI$(Y,5,16)
21 OPEN 0,1,STOR5
22 PRINT#1,Z: CLOSE
23 RUN ASSEMBLE
24 PRINT "ILLEGAL ENTRY-TRY AGAIN"
25 GOTO 11
26 '---EDIT ROUTINE---
27 PRINT "EDIT"
28 M=1
29 PRINT X(M)
30 LINE INPUT ""; Z
31 IF Z<"UP" GOTO 34
32 IF M=0 GOTO 29
33 M=M-1: GOTO 29
34 IF Z<"IN" GOTO 36
35 M=M+1: GOTO 29
36 IF Z="DEL" GOTO 113
37 IF Z="INS" GOTO 104
38 IF LEFT$(Z,4)="FILE" GOTO 72
39 IF Z<"INPUT" GOTO 41
40 J=J+1: GOTO 53
41 IF Z="EDIT" GOTO 30
42 IF Z<"PRINT" GOTO 44
43 GOSUB 119 : GOTO 27
44 IF LEFT$(Z,1)="L" GOTO 91
45 IF LEFT$(Z,1)="C" GOTO 128
46 IF Z<"TOF" GOTO 48
M=0: GOTO 29
47 IF Z<>"EOF" GOTO 50
48 M=J: GOTO 29
49 PRINT "ILLEGAL ENTRY-TRY AGAIN"
50 GOTO 29
51 -----INPUT ROUTINE-----
52 PRINT "INPUT"
53 FOR I=J TO 1000
54 LINE INPUT "";X(I)
55 IF X(I)="QUIT" GOTO 10
56 IF X(I)<"EOF" GOTO 27
57 X(I)="EOF": GOTO 27
58 IF LEFT$(X(I),4)<"FILE" GOTO 62
59 Y=MID$(X(I),6,10): J=I
60 X(I)="EOF": GOTO 76
61 J=I
62 NEXT I
63 PRINT "YOU HAVE USED ALL 8748 MEMORY"
64 ENI
65 -----LPT1 ROUTINE-----
66 FOR O=0 TO J
67 LPRINT X(O)
68 NEXT O
69 GOTO 11
70 -----FILE ROUTINE-----
71 Y=MID$(Z,6,10)
72 IF LEN(Y)<2 GOTO 76
73 PRINT "FILENAME REQUIRED"
74 GOTO 27
75 OPEN"O",1,Y
76 PRINT#1,J
77 FOR K=0 TO J
78 PRINT#1,X(K)
79 NEXT K: CLOSE
80 GOTO 11
81 -----GET ROUTINE-----
82 Z=MID$(Y,5,10)
83 OPEN"I",1,Z
84 INPUT#1,J
85 FOR K=0 TO J
86 PRINT#1,X(K)
87 LINE INPUT#1,X(K)
88 NEXT K: CLOSE
89 GOTO 11
82 -----LOCATE ROUTINE-----
90 FOR K=4 TO 16
91 IF MID$(Z,K,1)<"/" GOTO 100
92 N=K-4
93 XL=MID$(Z,4,N)
94 FOR M=0 TO J
95 L=INSTR(X(M),XL)
96 IF L<>0 GOTO 29
97 NEXT M
98 PRINT "STRING NOT LOCATED": GOTO 29
99 EXIT K
82
101 PRINT "RIGHT HANG DELINEATOR NOT FOUND"
102 GOTO 29
103 ' ----- INSERT ROUTINE ----- 
104 FOR L=M TO J+1
105 W(M)="0"
106 W(L+1)=X(L)
107 X(L)=W(L)
108 NEXT
109 J=J+1
110 LINE INPUT "":X(M)
111 GOTO 29
112 ' ----- DELETE ROUTINE ----- 
113 FOR L=M TO J
114 X(L)=X(L+1)
115 NEXT L
116 J=J-1
117 GOTO 29
118 ' ----- PRINT ROUTINE ----- 
119 FOR K=6 TO J
120 PRINT X(K)
121 NEXT K
122 RETURN
123 END
124 ' ----- TOF ROUTINE ----- 
125 M=0
126 GOTO 29
127 ' ----- CHANGE ROUTINE ----- 
128 FOR K=4 TO 16
129 IF MID$(Z,K,1)<"/" GOTO 148
130 N=K-4
131 XO=MID$(Z,4,N)
132 P=K+1
133 FOR O=P TO 35
134 IF MID$(Z,O,1)<"/" GOTO 145
135 Q=O-P
136 L=INSTR(X(M),XO)
137 IF L=0 GOTO 151
138 H2=L+N
139 L=L-1
140 X1=MID$(X(M),1,L)
141 X2=MID$(Z,P,Q)
142 X3=MID$(X(M),H2,50)
143 X(M)=X1+X2+X3
144 GOTO 29
145 NEXT Q
146 PRINT "LAST DELINEATOR NOT FOUND"
147 GOTO 29
148 NEXT K
149 PRINT "SECOND DELINEATOR NOT FOUND"
150 GOTO 29
151 PRINT "OLD STRING NOT FOUND"
152 GOTO 29
APPENDIX H

Assembler Program - "ASSEMBLE"

1 '****"ASSEMBLE"****
2 '----ASSEMBLY ROUTINE----
3 CLEAR 10000
4 LEFINT H-P
5 LEFSR A,U,V,W,X,Y,Z,T
6 DIM D(100),HX$(100),X(100)
7 DIM U(100),V(100),Y(100),Z(100)
8 OPEN 'I,1,'STORE'
9 INPUT#1,A : CLOSE
10 OPEN'I,A : PRINT'
11 INPUT#1,A
12 FOR I=0 TO J-1
13 LINE INPUT#1,X(I)
14 IF INSTR(X(I),A)<>0 GOTO 17
15 B=INSTR(X(I),' ;
16 V(I)=MILS(X(I),1,B)
17 NEXT I : CLOSE
18 IF INSTR(8,X(0),"ORIG")<>0 GOTO 20
19 W(0)=MILS(X(0),17,4):L(0)=VAL(W(0))
20 FOR K=0 TO J-1
21 T=X(K):IF INSTR(T,";")=0 GOTO 23
22 H=INSTR(T," ;
23 IF INSTR(8,T,"J")<>0 GOTO 91
24 IF INSTR(8,T,"#")<>0 GOTO 91
25 IF INSTR(8,T,"CALL")<>0 GOTO 91
26 BK=1
27 IF INSTR(8,T,"END")<>0 GOTO 31
28 L(K+1)=T+BK : E=L(K) : GOSUB 93
29 HX$(K)=A(3)+A(2)+A(1)
30 PRINT K+1;TAB(6) HX$(K);TAB(12) X(K)
31 NEXT K
32 PRINT "FIRST PASS COMPLETE"
33 FOR L=0 TO J-1
34 T=X(L):IF INSTR(X(L)," ;")<>0 GOTO 36
35 H=INSTR(X(L)," ;
36 IF INSTR(8,T,"CLR")<>0 GOTO 201
37 IF INSTR(8,T,"MOV")<>0 GOTO 316
38 IF INSTR(8,T,"IN")<>0 GOTO 253
39 IF INSTR(8,T,"OUT")<>0 GOTO 378
40 IF INSTR(8,T,"OUT")<>0 GOTO 405
41 IF INSTR(8,T,"ALL")<>0 GOTO 119
42 IF INSTR(8,T,"RET")<>0 GOTO 415
43 IF INSTR(8,T,"RL")<>0 GOTO 420
44 IF INSTR(8,T,"RR")<>0 GOTO 426
45 IF INSTR(8,T,"XCH")<>0 GOTO 446
46 IF INSTR(8,T,"DJNZ")<>0 GOTO 232
47 IF INSTR(6,T, "JC ")<>0 GOTO 279
48 IF INSTR(6,T, "JF ")<>0 GOTO 262
49 IF INSTR(6,T, "JMP ")<>0 GOTO 289
50 IF INSTR(6,T, "JN ")<>0 GOTO 296
51 IF INSTR(6,T, "CPL ")<>0 GOTO 210
52 IF INSTR(6,T, "MOVI ")<>0 GOTO 357
53 IF INSTR(6,T, "MOVP ")<>0 GOTO 365
54 IF INSTR(6,T, "MOVI ")<>0 GOTO 370
55 IF INSTR(6,T, "JB ")<>0 GOTO 271
56 IF INSTR(6,T, "ALDC ")<>0 GOTO 118
57 IF INSTR(6,T, "JT ")<>0 GOTO 307
58 IF INSTR(6,T, "JZ ")<>0 GOTO 314
59 IF INSTR(8,T, "NOP ")=0 GOTO 61
60 Z(L)="0E":GOTO 146
61 IF INSTR(8,T, "DA ")=0 GOTO 63
62 Z(L)=57 : GOTO 140
63 IF INSTR(8,T, "DEC ")<>0 GOTO 219
64 IF INSTR(8,T, "STOP ")=0 OR INSTR(13,T, "TCNT ")=0 GOTO 66
65 Z(L)=65 :GOTO 140
66 IF INSTR(8,T, "SWAP ")=0 OR INSTR(13,T, "A ")=0 GOTO 69
67 Z(L)=47":GOTO 140
68 IF INSTR(8,T, "INC ")<>0 GOTO 259
69 IF INSTR(8,T, "SEL ")<>0 GOTO 432
70 IF INSTR(8,T, "ANL ")<>0 GOTO 149
71 IF INSTR(8,T, "CALL ")<>0 GOTO 183
72 IF INSTR(8,T, "STRT ")<>0 GOTO 441
73 IF INSTR(8,T, "INS ")=0 GOTO 75
74 Z(L)=08 : GOTO 140
75 IF INSTR(8,T, "EN ")<>0 GOTO 244
76 IF INSTR(8,T, "XRL ")<>0 GOTO 458
77 IF INSTR(8,T, "DIS ")<>0 GOTO 227
78 IF INSTR(8,T, "ORIG ")<>0 GOTO 80
79 GOTO 125
80 NEXT L
81 PRINT"ASSEMBLY COMPLETE"
82 PRINT ERS; " ERRORS"
83 IF ERS=0 GOTO 473
84 PRINT"LINE ERROR"
85 FOR I=0 TO J
86 IF LEN(U(I))=0 GOTO 88
87 PRINT (I+1);TAB(9) U(I)
88 NEXT I
89 GOTO 473
90 END
91 IF INSTR(8,T, "JMPP ")<>0 GOTO 26
92 BL=2 : GOTO 27
93 II=FIX(E/16)
94 F(1)=E-(16*I1)
95 I2=FIX(I1/16)
96 F(2)=I1-(16*I2)
97 I3=FIX(I2/16)
98 F(3)=I2-(16*I3)
99 FOR N=1 TO 3
100 IF P(N)=0 THEN A(N)="0"

101 IF F(N)=1 THEN A(N)="1"
102 IF F(N)=2 THEN A(N)="2"
103 IF F(N)=3 THEN A(N)="3"
104 IF F(N)=4 THEN A(N)="4"
105 IF F(N)=5 THEN A(N)="5"
106 IF F(N)=6 THEN A(N)="6"
107 IF F(N)=7 THEN A(N)="7"
108 IF F(N)=8 THEN A(N)="8"
109 IF F(N)=9 THEN A(N)="9"
110 IF F(N)=10 THEN A(N)="A"
111 IF F(N)=11 THEN A(N)="B"
112 IF F(N)=12 THEN A(N)="C"
113 IF F(N)=13 THEN A(N)="D"
114 IF F(N)=14 THEN A(N)="E"
115 IF F(N)=15 THEN A(N)="F"
116 NEXT N : RETURN
117 '----FOR "ADD" & "ANL"----
118 Q=1 : GOTO 120
119 Q=0
120 E=INSTR(S,T,"","")
121 IF B=0 GOTO 125
122 IF INSTR(B,T,"","R")<>0 GOTO 133
123 IF INSTR(B,T,"","G")<>0 GOTO 143
124 IF INSTR(B,T,"","#")<>0 GOTO 126
125 ERS=ERS+1 : U(L)="SYNTAX ERROR" : GOTO 80
126 W=MID$(T,(B+2),3)
127 E=VAL(W)
128 IF E<256 GOTO 131
129 ERS=ERS+1
130 U(L)="DATA EXCEEDS EXT SIZE" :GOTO 80
131 IF Q=0 THEN Y(L)= 03 ELSE Y(L)= 19
132 GOTO 139
133 W=MID$(T,(B+2),1)
134 IF VAL(W)<8 GOTO 137
135 ERS=ERS+1
136 U(L)="REGISTER SIZE EXCEEDS 7" :GOTO 80
137 IF Q=0 THEN E=104 +VAL(W) ELSE E=120+VAL(W)
138 GOSUB 93
139 Z(L)=A(2)+A(1)
140 IF Y(L)<"" THEN PRINT Y(L)+Z(L) ELSE PRINT Z(L)
141 GOTO 80
142 ENL
143 W=MID$(T,(B+3),1)
144 IF VAL(W)<2 GOTO 147
145 ERS=ERS+1
146 U(L)="R EXCEEDS 1" :GOTO 80
147 IF Q=0 THEN E=96+VAL(W) ELSE E=112+VAL(W)
148 GOTO 138 : END
149 IF INSTR(S,T,1,)="E" GOTO 163
150 B=INSTR(S,T,2,)
151 IF INSTR(B,T,",R")="E" GOTO 155
152 W=MID$(T,(B+2),1) : E=VAL(W)
153 IF E>7 GOTO 135
154 E=88+E : GOTO 138
155 IF INSTR(B, T, ",", 0) = 0 GOTO 159
156 W = MID$(T, (B + 3), 1)
157 IF VAL(W) > 1 GOTO 145
158 E = 80 + VAL(W) : GOTO 139
159 IF INSTR(E, T, ",", 0) = 0 GOTO 164
160 W = MID$(T, (B + 2), 3) : E = VAL(W)
161 IF E > 255 GOTO 129
162 Y(L) = ";53": GOTO 138
163 E = INSTR(E, T, ",", 0)
164 IF INSTR(B, T, ",", 0) = 0 GOTO 168
165 W = MID$(T, (B + 2), 3) : E = VAL(W)
166 IF E > 255 GOTO 129
167 Y(L) = ";98": GOTO 138
168 IF INSTR(B, T, ",", 0) = 0 GOTO 125
169 IF INSTR(B, T, ",", 0) = 0 GOTO 178
170 W = MID$(T, (B - 1), 1)
171 IF VAL(W) < 3 AND VAL(W) > 0 GOTO 174
172 ERS = ERS + 1
173 U(L) = ";INCORRECT PORT ": GOTO 80
174 E = 152 + VAL(W) : GOSUB 93
175 Y(L) = A(2) + A(1)
176 W = MID$(T, (B + 2), 3) : E = VAL(W)
177 IF E > 255 GOTO 129 ELSE GOTO 136
178 IF INSTR(B, T, ",", 0) = 0 OR INSTR(B, T, ",", 0) = 0 GOTO 125
179 W = MID$(T, (B - 1), 1) : E = VAL(W) - 4
180 IF E < 0 OR E > 3 GOTO 172
181 E = 156 + E : GOTO 139
182 "-- CALL --"
183 GOSUB 186
184 C = VAL(A(3)) * 2 + 1
185 Y(L) = STR$(C) + 4
186 Y(L) = RIGHT$(Y(L), 2)
187 GOTO 140
188 "-- ADDRESS SUBROUTINE --"
189 IF INSTR(18, T, ";") = 0 GOTO 191
190 B = INSTR(18, T, ";") : W = MID$(T, 17, (B - 17)) : GOTO 192
191 W = MID$(T, (B - 1), 8)
192 FOR M = 0 TO J - 1
193 IF W = V(M) GOTO 197
194 NEXT M
195 ERS = ERS + 1
196 U(L) = ";LABEL-" + W + "; NOT FOUND": GOTO 80
197 E = D(M) : GOSUB 93
198 Z(L) = A(2) + A(1)
199 RETURN
200 "-- CLR --"
201 IF INSTR(13, T, ";") = 0 GOTO 203
202 Z(L) = "27": GOTO 140
203 IF INSTR(13, T, ";") = 0 GOTO 205
204 Z(L) = ";57": GOTO 140
205 IF INSTR(13, T, ";") = 0 GOTO 207
206 Z(L) = A(5) : GOTO 140
207 IF INSTR(13, T, ";") = 0 GOTO 125
208 Z(L) = "85": GOTO 140

87
209 '---"CPL"---
210 IF INSTR(13,T,"A")=0 GOTO 212
211 Z(L)="37":GOTO 140
212 IF INSTR(13,T,"C")=0 GOTO 214
213 Z(L)="A7":GOTO 140
214 IF INSTR(13,T,"P")=0 GOTO 216
215 Z(L)="95":GOTO 140
216 IF INSTR(13,T,"B")=0 GOTO 125
217 Z(L)="B5":GOTO 140
218 '---DEC---
219 IF INSTR(13,T,"A")=0 GOTO 221
220 Z(L)="07":GOTO 140
221 IF INSTR(13,T,"R")=0 GOTO 125
222 B=INSTR(13,T,"R")
223 W=MID$(T,(B+1),1):E=VAL(W)
224 IF E>7 GOTO 135
225 E=E+200:GOTO 138
226 '---DIS---
227 IF INSTR(13,T,"I")=0 GOTO 229
228 Z(L)="15":GOTO 140.
229 IF INSTR(13,T,"TCNTI")=0 GOTO 125
230 Z(L)="35":GOTO 140
231 '---DJNZ---
232 B=INSTR(13,T,"")
233 W=MID$(T,(B-1),1):E=VAL(W)
234 IF E>7 GOTO 135
235 E=E+232:GOSUB 93
236 T(L)=A(2)+A(1)
237 W=MID$(T,(E+1),8)
238 FOR M=0 TO J-1
239 IF W=V(M) GOTO 242
240 NEXT M
241 GOTO 195
242 E=V(M):GOSUB 93:Z(L)=A(2)+A(1):GOTO 140
243 '---EN---
244 IF INSTR(13,T,"TCNTI")=0 GOTO 246
245 Z(L)="25":GOTO 140
246 IF INSTR(13,T,"I")=0 GOTO 248
247 Z(L)="05":GOTO 140
248 IF INSTR(13,T,"CLK")=0 GOTO 250
249 Z(L)="75":GOTO 140
250 IF INSTR(13,T,"EN")=0 GOTO 125
251 GOTO 81
252 '---IN---
253 IF INSTR(13,T,"A")=0 GOTO 125
254 B=INSTR(13,T,"P")
255 W=MID$(T,(B+1),1):E=VAL(W)
256 IF E>2 GOTO 172
257 E=E+8:GOTO 138
258 '---INC---
259 IF INSTR(13,T,"A")=0 GOTO 261
260 Z(L)="17":GOTO 140
261 B=INSTR(13,T,"P")
262 IF INSTR(13,T,"P")=0 GOTO 266
263 \text{W=M}il\$\text{(T,}(B+1),1) : \text{E} = \text{VAL}(\text{W}) \\
264 \text{IF E}>7 \text{ GOTO 135} \\
265 \text{E} = \text{E} + 24 : \text{GOTO 138} \\
266 \text{IF INSTR}(13, \text{T}, "GR") = 0 \text{ GOTO 125} \\
267 \text{W=M}il\$\text{(T,}(B+1),1) \\
268 \text{E} = \text{VAL}(\text{W}) : \text{IF E}>1 \text{ GOTO 145} \\
269 \text{E} = \text{E} + 16 : \text{GOTO 138} \\
270 \text{--JB--} \\
271 \text{E}INSTR(8, \text{T}, "B") \\
272 \text{W=M}il\$\text{(T,}(B+1),1) : \text{E} = \text{VAL}(\text{W}) \\
273 \text{IF E}<8 \text{ GOTO 275} \\
274 \text{ERS} = \text{ERS} + 1 : \text{U}(L)="\text{BIT} > 7" \\
275 \text{E} = 18 + (\text{E} \cdot 32) : \text{GOSUB 93} \\
276 \text{Y}(L) = \text{A}(2) + \text{A}(1) \\
277 \text{GOSUB 189} : \text{GOTO 140} \\
278 \text{--JC--} \\
279 \text{Y}(L)="\text{FE}" \\
280 \text{GOSUB 189} : \text{GOTO 140} \\
281 \text{--JF--} \\
282 \text{IF INSTR}(8, \text{T}, "\text{JF0 }) = 0 \text{ GOTO 285} \\
283 \text{Y}(L)="\text{ES}" \\
284 \text{GOSUB 189} : \text{GOTO 140} \\
285 \text{IF INSTR}(8, \text{T}, "\text{JF1 }) = 0 \text{ GOTO 125} \\
286 \text{Y}(L)="\text{JS}" \\
287 \text{GOSUB 189} : \text{GOTO 140} \\
288 \text{--JMP--} \\
289 \text{IF INSTR}(8, \text{T}, "\text{JMP }) = 0 \text{ GOTO 291} \\
290 \text{Z}(L)="\text{BS}:GOTO 140" \\
291 \text{IF INSTR}(8, \text{T}, "\text{JMP }) = 0 \text{ GOTO 125} \\
292 \text{GOSUB 189} \\
293 \text{E} = \text{VAL}(\text{A}(3)) \cdot 2 : \text{GOSUB 93} \\
294 \text{Y}(L) = \text{A}(1) + '4' : \text{GOTO 140} \\
295 \text{--JN--} \\
296 \text{IF INSTR}(8, \text{T}, "\text{JNC }) = 0 \text{ GOTO 298} \\
297 \text{Y}(L)="\text{ES}:GOSUB 189:GOTO 140" \\
298 \text{IF INSTR}(8, \text{T}, "\text{JN}) = 0 \text{ GOTO 300} \\
299 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
300 \text{IF INSTR}(8, \text{T}, "\text{JNO }) = 0 \text{ GOTO 302} \\
301 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
302 \text{IF INSTR}(8, \text{T}, "\text{JNT }) = 0 \text{ GOTO 304} \\
303 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
304 \text{IF INSTR}(8, \text{T}, "\text{JNZ }) = 0 \text{ GOTO 125} \\
305 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
306 \text{--JT--} \\
307 \text{IF INSTR}(8, \text{T}, "\text{JTF }) = 0 \text{ GOTO 309} \\
308 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
309 \text{IF INSTR}(8, \text{T}, "\text{JTO }) = 0 \text{ GOTO 311} \\
310 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
311 \text{IF INSTR}(8, \text{T}, "\text{JT }) = 0 \text{ GOTO 125} \\
312 \text{Y}(L)="\text{JS}:GOSUB 189:GOTO 140" \\
313 \text{--JZ--} \\
314 \text{Y}(L)="\text{CS}:GOSUB 189:C TO 142" \\
315 \text{--MOV--} \\
316 \text{IF INSTR}(13, \text{T}, "\text{R}) = 0 \text{ GOTO 342}
317 B=INSTR(13,T,"R")
318 W=MID$(T,(B+1),1):E=VAL(W)
319 IF INSTR(13,T,"A",R")=0 GOTO 322
320 IF E>7 GOTO 135
321 E=E+248:GOTO 138
322 IF INSTR(13,T,"A","R")=0 GOTO 325
323 IF E<1 GOTO 146
324 E=E+248:GOTO 138
325 IF INSTR(13,T,"A",R")<>0 AND INSTR(13,T,"OR")<>0 GOTO 329
326 IF INSTR(13,T,"A")=0 GOTO 331
327 IF E>7 GOTO 135
328 E=E+168:GOTO 138
329 IF E<1 GOTO 146
330 E=E+168:GOTO 138
331 IF INSTR(13,T,"A",R")=0 GOTO 125
332 IF INSTR(13,T,"OR")=0 GOTO 340
333 IF E<1 GOTO 146
334 E=E+176
335 GOSUB 93
336 Y(L)=A(2)+A(1)
337 W=MID$(T,(B+4),3):E=VAL(W)
338 IF E>255 GOTO 129
339 GOTO 138
340 IF E>7 GOTO 135
341 E=E+184:GOTO 335
342 IF INSTR(13,T,"A",R")=0 GOTO 348
343 B=INSTR(13,T,"A")
344 Y(L)="23"
345 W=MID$(T,(B+2),3):E=VAL(W)
346 IF E>255 GOTO 129
347 GOTO 138
348 IF INSTR(13,T,"A",PSW")=0 GOTO 350
349 Z(L)="C7":GOTO 140
350 IF INSTR(13,T,"A",T")=0 GOTO 352
351 Z(L)="42":GOTO 140
352 IF INSTR(13,T,"PSW,A")=0 GOTO 354
353 Z(L)="B7"
354 IF INSTR(13,T,"T,A")=0 GOTO 125
355 Z(L)="e2":GOTO 140
356 '('--MOV I--'
357 B=INSTR(13,T,"P")
358 W=MID$(T,(B+1),1):E=VAL(W)-4
359 IF E<0 OR E>3 GOTO 172
360 IF INSTR(13,T,"A",P")=0 GOTO 362
361 E=E+12+E:GOTO 138
362 IF INSTR(13,T,"A")=0 GOTO 125
363 E=E+60:GOTO 138
364 '('--M V P--'
365 IF INSTR(8,T,"MOV P")=0 GOTO 367
366 Z(L)="A3":GOTO 140
367 IF INSTR(8,T,"MOV P")=0 GOTO 125
368 Z(L)="e3":GOTO 140
369 '('--MOV X--'
370 B=INSTR(13,T,"R")
371 \( W = \text{MID}(T,(E+1),1) \): \( E = \text{VAL}(W) \)
372 IF \( E > 1 \) GOTO 145  
373 IF \( \text{INSTR}(13,T,"A,\&") \) = 0 GOTO 375  
374 \( E = E + 128 + E \): GOTO 138  
375 IF \( \text{INSTR}(13,T,"A") \) = 0 GOTO 125  
376 \( E = E + 144 + E \): GOTO 138  
377 "--ORL--"  
378 \( B = \text{INSTR}(13,T,"A") \)  
379 IF \( \text{INSTR}(13,T,"A") \) = 0 GOTO 392  
380 \( W = \text{MID}(T,(B-2),3) \): \( E = \text{VAL}(W) \)  
381 IF \( E > 255 \) GOTO 129  
382 \( \text{GOSUB} 93 \)  
383 \( Z(L) = (A(2)+A(1)) \)  
384 IF \( \text{INSTR}(13,T,"A") \) = 0 GOTO 386  
385 \( Y(L) = 43 \): GOTO 140  
386 IF \( \text{INSTR}(13,T,"BUS,\&") \) = 0 GOTO 388  
387 \( Y(L) = 88 \): GOTO 140  
388 IF \( \text{INSTR}(13,T,"P") \) = 0 GOTO 125  
389 \( W = \text{MID}(T,(B-1),1) \): \( E = \text{VAL}(W) \)  
390 IF \( E < 1 \) OR \( E > 2 \) GOTO 172  
391 \( E = E + 156 + \text{GOSUB} 93 : Y(L) = A(2)+A(1) \): GOTO 140  
392 IF \( \text{INSTR}(8,T,"ORL") \) = 0 GOTO 396  
393 \( W = \text{MID}(T,(B-1),1) \): \( E = \text{VAL}(W) \)  
394 IF \( E < 0 \) OR \( E > 3 \) GOTO 172  
395 \( E = E + 140 + \text{GOSUB} 93 : Y(L) = A(2)+A(1) \): GOTO 140  
396 IF \( \text{INSTR}(13,T,"A,\&") \) = 0 GOTO 400  
397 \( W = \text{MID}(T,(E+2),1) \): \( E = \text{VAL}(W) \)  
398 IF \( E > 7 \) GOTO 135  
399 \( E = E + 72 \): GOTO 138  
400 IF \( \text{INSTR}(13,T,"A,\&") \) = 0 GOTO 125  
401 \( W = \text{MID}(T,(E+3),1) \): \( E = \text{VAL}(W) \)  
402 IF \( E > 1 \) GOTO 145  
403 \( E = E + 64 \): GOTO 138  
404 "--OUL--"  
405 IF \( \text{INSTR}(13,T,"BUS,\&") \) = 0 GOTO 407  
406 \( Z(L) = 02 \): GOTO 140  
407 IF \( \text{INSTR}(13,T,"A") \) = 0 GOTO 125  
408 IF \( \text{INSTR}(13,T,"\&") \) = 0 GOTO 413  
409 \( B = \text{INSTR}(13,T,"A") \)  
410 \( W = \text{MID}(T,(B-1),1) \): \( E = \text{VAL}(W) \)  
411 IF \( E > 2 \) OR \( E < 1 \) GOTO 172  
412 \( E = E + 56 \): GOTO 138  
413 \( Z(L) = 90 \): GOTO 140  
414 "--RET--"  
415 IF \( \text{INSTR}(8,T,"\text{RETR}") \) = 0 GOTO 417  
416 \( Z(L) = 93 \): GOTO 140  
417 IF \( \text{INSTR}(8,T,"\text{RETR}") \) = 0 GOTO 125  
418 \( Z(L) = 83 \): GOTO 140  
419 "--RL--"  
420 IF \( \text{INSTR}(13,T,"A") \) = 0 GOTO 125  
421 IF \( \text{INSTR}(8,T,"\text{RLC}") \) = 0 GOTO 423  
422 \( Z(L) = F7 \): GOTO 140  
423 IF \( \text{INSTR}(8,T,"\text{RL}") \) = 0 GOTO 125  
424 \( Z(L) = E7 \): GOTO 140
425 '---RR---
426 IF INSTR(12,T,"A")=0 GOTO 125
427 IF INSTR(8,T,"RRC")=0 GOTO 429
428 Z(L)="67" :GOTO 140
429 IF INSTR(8,T,"RR")=0 GOTO 125
430 Z(L)="??" :GOTO 140
431 '---SEL---
432 IF INSTR(12,T,"MB")=0 GOTO 434
433 Z(L)="E5" :GOTO 140
434 IF INSTR(12,T,"MB1")=0 GOTO 436
435 Z(L)="F5" :GOTO 140
436 IF INSTR(12,T,"RBO")=0 GOTO 438
437 Z(L)="C5" :GOTO 140
438 IF INSTR(12,T,"RB")=0 GOTO 125
439 Z(L)="L5" :GOTO 140
440 '---STRT---
441 IF INSTR(13,T,"CNT")=0 GOTO 443
442 Z(L)="45" :GOTO 140
443 IF INSTR(13,T,"T")=0 GOTO 125
444 Z(L)="55" :GOTO 140
445 '---XCH---
446 B=INSTR(12,T,"R")
447 W=MID$(T,(B+1),1):E=VAL(W)
448 IF INSTR(9,T,"XCH")=0 OR INSTR(12,T,"A,GR")=0 GOTO 451
449 IF E>1 GOTO 145
450 E=E+48:GOTO 139
451 IF INSTR(12,T,"A,GR")=0 GOTO 454
452 IF E>7 GOTO 135
453 E=E+40:GOTO 139
454 IF INSTR(12,T,"A,GR")=0 GOTO 125
455 IF E>1 GOTO 145
456 E=E+32:GOTO 138
457 '---XRL---
458 B=INSTR(12,T,"R")
459 W=MID$(T,(B+1),1):E=VAL(W)
460 IF INSTR(12,T,"A,GR")=0 GOTO 463
461 IF E>7 GOTO 135
462 E=E+216:GOTO 138
463 IF INSTR(12,T,"A,GR")=0 GOTO 466
464 IF E>1 GOTO 145
465 E=E+208:GOTO 138
466 IF INSTR(12,T,"A,GR")=0 GOTO 125
467 B=INSTR(12,T,"#")
468 W=MID$(T,(B+1),3):E=VAL(W)
469 IF E>255 GOTO 129
470 Y(L)=L3 :GOTO 138
471 END
472 '---LINE PRINT LISTING---
473 LPRINT "LPRINT "ASSEMBLY OF "+";Y;"+" COMPLETED"
474 LPRINT "+"+"HEA"+" +"CODE"+" +"LABEL"+" +"OPCODE +" +"OPERAND +" +"COMMENTS"
475 FOR N=0 TO J-1
476 LPRINT N+1;TAB(6) HX$(N);TAB(12) Y(N)+Z(N);TAB(19) X(N)

92
478 IF LEN(U(N))<>0 LPRINT U(N)
479 NEXT N
480 LPRINT
481 LPRINT "SYMBOL TABLE:"
482 LPRINT ""
483 FOR M=0 TO J
484 IF LEN(V(M))<>0 LPRINT V(M);"---"+H$(M)
485 NEXT M
486 IF ERS<0 GOTO 496
487 LPRINT
488 LPRINT TAB(36) ">>> NO ASSEMBLY ERRORS DETECTED <<<"
489 GOTO 496
490 LPRINT "" LPRINT ERS;" ERRORS DETECTED"
491 RUN "MASTER"
492 Y=Y+"0"
493 OPEN 0,1,Y
494 FOR M=0 TO J
495 IF LEN(Y(M))=0 GOTO 497
496 PRINT #1,Y(M)
497 PRINT#1,Z(M)
498 NEXT M
499 CLOSE
500 RUN "PROGRAM"
APPENDIX I
Programming Program - "PROGRAM"

1 '****PROGRAM****
2 '-----EPROM ROUTINE-----
3 CLEAR 10000
4 DEFINT A,I,J,K,M,N,O,Q,P
5 DEFSTR W,X,Y,Z
6 DIM L(1024),T(1024),X(1024)
7 C=0
8 LINE INPUT "ENTER PROGRAM MOIE--";Y
9 IF Y="STOP" ENL
10 IF Y="ELIT" RUN "ELTASM"
11 IF Y="RPROM" GOTO 9c
12 IF LEFT$(Y,5)="PROM" GOTO 16
13 PRINT "ILLEGAL COMMAND-TRY AGAIN"
14 GOTO 8
15 '-----WPROM ROUTINE-----
16 Z=MID$(Y,7,11)
17 IF RIGHT$(Z,1)="O" GOTO 2e.
18 PRINT "OBJECT CODE ONLY! AED")
19 GOTO 8
20 OPEN "I",1,Z
21 FOR I=0 TO 1024
22 INPUT #1,X(I)
23 IF EOF(1) THEN 25
24 NEXT I
25 CLOSE : J=I
26 GOSUB 65
27 GOSUB 148
28 OUT 4,4
29 OUT 4,20
30 FOR M=1 TO Q
31 OUT 2,N1
32 OUT 4,21
33 OUT 1,11
34 OUT 4,81
35 OUT 1,D(M)
36 OUT 4,209
37 OUT 4,221
38 GOSUB 59
39 OUT 4,209
40 OUT 4,81
41 OUT 4,84
42 OUT 4,116
43 T(M)=INP(3)
44 OUT 4,20
45 I1=I1+1
46 IF I1<>256 OR I1<>512 OR I1<>768 GOTO 48
47 N1 = N1 + 1
48 IF I(M) = T(M) GOTO 51
49 PRINT "PROGRAMMING ERROR—ERASE EPROM AND TRY AGAIN"
50 GOTO 53
51 NEXT M
52 PRINT "PROGRAMMING COMPLETED SATISFACTORILY"
53 OUT 4, 36
54 PRINT "REMOVE EPROM NOW—THEN TURN POWER OFF"
55 GOTO 8
56 END
57 '---50 MSEC DELAY---
58 A = A + 1
59 'DELAY
60 IF A < 2 GOTO 58
61 A = 0
62 RETURN
63 END
64 '---HEX TO DEC CONVERSION---
65 FOR K = 0 TO J
66 W(0) = LEFT$(X(K), 1) : W(1) = MID$(X(K), 2, 1)
67 FOR N = 0 TO 1
68 IF W(N) = "0" A(N) = 0
69 IF W(N) = "1" A(N) = 1
70 IF W(N) = "2" A(N) = 2
71 IF W(N) = "3" A(N) = 3
72 IF W(N) = "4" A(N) = 4
73 IF W(N) = "5" A(N) = 5
74 IF W(N) = "6" A(N) = 6
75 IF W(N) = "7" A(N) = 7
76 IF W(N) = "8" A(N) = 8
77 IF W(N) = "9" A(N) = 9
78 IF W(N) = "A" A(N) = 10
79 IF W(N) = "B" A(N) = 11
80 IF W(N) = "C" A(N) = 12
81 IF W(N) = "D" A(N) = 13
82 IF W(N) = "E" A(N) = 14
83 IF W(N) = "F" A(N) = 15
84 IF LEN(W(N)) = 0 A(N) = 0
85 NEXT N
86 L(K) = 16 * A(0) + A(1)
87 NEXT K
88 RETURN
89 '---READ PROM ROUTINE---
90 GOSUB 148
91 OUT 4, 4
92 OUT 4, 20
93 FOR M = 1 TO (Q + 1)
94 OUT 2, N1
95 OUT 4, 21
96 OUT 1, 11
97 OUT 4, 85
98 OUT 4, 84
99 OUT 4, 116
100 T(M) = INP(3)
101 OUT 4,84
102 OUT 4,20
103 I1=I1+1
104 IF I1<256 GOTO 107
105 I1=I1-256
106 N1=N1+1
107 NEXT M
108 OUT 4,36
109 FOR N=1 TO Q+1
110 E=T(N)
111 N1=FIX(E/16)
112 F(1)=E-(16*N1)
113 N2=FIX(N1/16)
114 F(2)=N1-(16*N2)
115 N3=FIX(N2/16)
116 F(3)=N2-(16*N3)
117 FOR P=1 TO 2
118 IF F(P)=0 THEN W(P)="0"
119 IF F(P)=1 THEN W(P)="1"
120 IF F(P)=2 THEN W(P)="2"
121 IF F(P)=3 THEN W(P)="3"
122 IF F(P)=4 THEN W(P)="4"
123 IF F(P)=5 THEN W(P)="5"
124 IF F(P)=6 THEN W(P)="6"
125 IF F(P)=7 THEN W(P)="7"
126 IF F(P)=8 THEN W(P)="8"
127 IF F(P)=9 THEN W(P)="9"
128 IF F(P)=10 THEN W(P)="A"
129 IF F(P)=11 THEN W(P)="B"
130 IF F(P)=12 THEN W(P)="C"
131 IF F(P)=13 THEN W(P)="D"
132 IF F(P)=14 THEN W(P)="E"
133 IF F(P)=15 THEN W(P)="F"
134 NEXT P
135 X(N)=W(2)+W(1)
136 NEXT N
137 GOSUB 139
138 GOTO 53
139 0=1
140 PRINT I;TAB(6) T(Q)+""+X(Q+1)+""+X(Q+2)+""+X(Q+3)+""
+X(0+4)+""+X(0+5)+""+X(0+6)+""+X(0+7)+""+X(0+8)+
""+X(0+9)+""+X(0+10)+""+X(0+11)+""+X(0+12)+""+X(0+13)+""
+X(0+14)+""+X(0+15)
141 O=O+16
142 IF O>Q+1 GOTO 144
143 IX=IX+16 : GOTO 140
144 PRINT
145 RETURN
146 END
147 '---SETUP ROUTINE---
148 N1=0
149 PRINT "START ADDRESS IN DECIMAL?"
150 INPUT I1
151 IF I1<1024 GOTO 153
PRINT "ILLEGAL ADDRESS" : GOTO 146
PRINT "ENTER ADDRESS IN DECIMAL?"
INPUT I2 : Q=I2-I1
II=I1
IF II>255 GOSUB 172
IF Q<0 GOTO 152
IF I2<1024 GOTO 160
PRINT "ILLEGAL ADDRESS" : GOTO 153
LINE INPUT "EPROM SOCKET EMPTY? (YES OR NO): " ; X3
IF X3="NC" GOTO 160
LINE INPUT "IS SOCKET POWER SWITCH ON? (YES OR NO): " ; X4
IF X4="NO" GOTO 164 ELSE GOTO 165
PRINT "TURN POWER ON" : GOTO 162
OUT 4,36
IF LEFT$(II,5)<"WPROM" GOTO 169
PRINT: PRINT "FOLLOWING IS HEX CODE TO BE PROGRAMMED:"
GOSUB 139
LINE INPUT "INSERT 8748 CHIP AND TYPE GO: " ; X5
IF X5<"GO" GOTO 165
RETURN
IF II>511 GOTO 174
N1=1: II=II-256: RETURN
IF II>767 GOTO 176
N1=2: II=II-512: RETURN
N1=3: II=II-768
RETURN
### APPENDIX J

#### Sample Assembler Printout

**ASSEMBLY OF 'DOUG'S' COMPLETED**

<table>
<thead>
<tr>
<th>LINE</th>
<th>HEX</th>
<th>CODE</th>
<th>LABEL</th>
<th>CPCODE</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>000</td>
<td></td>
<td>ORIG</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>00</td>
<td>002</td>
<td></td>
<td>JMP</td>
<td>START</td>
<td>POWER UP</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>003</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>005</td>
<td></td>
<td>JMP</td>
<td>INT</td>
<td>INTERRUPT</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>007</td>
<td></td>
<td>MOV</td>
<td>R2 #66</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>00</td>
<td>009</td>
<td></td>
<td>MCV</td>
<td>R1 #127</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>00</td>
<td>00B</td>
<td></td>
<td>MOV</td>
<td>R2 #13</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>00</td>
<td>00D</td>
<td></td>
<td>MOV</td>
<td>R3 #16</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00</td>
<td>00F</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>00F</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>011</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>00</td>
<td>013</td>
<td></td>
<td>MOV</td>
<td>A #81</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>00</td>
<td>015</td>
<td></td>
<td>OUTL</td>
<td>P2.A</td>
<td>HI-Z ON</td>
</tr>
<tr>
<td>14</td>
<td>00</td>
<td>017</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>00</td>
<td>019</td>
<td></td>
<td>MCV</td>
<td>A #255</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>00</td>
<td>01B</td>
<td></td>
<td>OUTL</td>
<td>P1.A</td>
<td>ENABLES P1</td>
</tr>
<tr>
<td>17</td>
<td>00</td>
<td>01D</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>00</td>
<td>01F</td>
<td></td>
<td>MCVX</td>
<td>A GR1</td>
<td>BUS TO HI-Z</td>
</tr>
<tr>
<td>19</td>
<td>00</td>
<td>021</td>
<td></td>
<td>ENTC</td>
<td>CLK</td>
<td>MAKES TO A CLOCK</td>
</tr>
<tr>
<td>20</td>
<td>00</td>
<td>023</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>00</td>
<td>025</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>00</td>
<td>027</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>00</td>
<td>029</td>
<td></td>
<td>LOOP</td>
<td>A A0</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>00</td>
<td>02B</td>
<td></td>
<td>OUTL</td>
<td>P1.A</td>
<td>ENABLE 8212</td>
</tr>
<tr>
<td>25</td>
<td>00</td>
<td>02D</td>
<td></td>
<td>JMP</td>
<td>LOOP</td>
<td>LOOP</td>
</tr>
<tr>
<td>26</td>
<td>00</td>
<td>02F</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>00</td>
<td>031</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>00</td>
<td>033</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>00</td>
<td>035</td>
<td></td>
<td>OUTL</td>
<td>P1.A</td>
<td>STOP TRS-G0</td>
</tr>
<tr>
<td>30</td>
<td>00</td>
<td>037</td>
<td></td>
<td>JMP</td>
<td>LOOP</td>
<td>LOOP</td>
</tr>
<tr>
<td>31</td>
<td>00</td>
<td>039</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>00</td>
<td>03B</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>00</td>
<td>03D</td>
<td></td>
<td>INT</td>
<td>A #102</td>
<td>INTERRUPT</td>
</tr>
<tr>
<td>34</td>
<td>00</td>
<td>03F</td>
<td></td>
<td>OUTL</td>
<td>P1.A</td>
<td>STOP TRS-G0</td>
</tr>
<tr>
<td>35</td>
<td>00</td>
<td>041</td>
<td></td>
<td>INS</td>
<td>A BUS</td>
<td>INPUT</td>
</tr>
<tr>
<td>36</td>
<td>00</td>
<td>043</td>
<td></td>
<td>ANL</td>
<td>A #21</td>
<td>7 BITS?</td>
</tr>
<tr>
<td>37</td>
<td>00</td>
<td>045</td>
<td></td>
<td>XRL</td>
<td>A #2</td>
<td>CHECK</td>
</tr>
<tr>
<td>38</td>
<td>00</td>
<td>047</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>00</td>
<td>049</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>00</td>
<td>04B</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>00</td>
<td>04D</td>
<td></td>
<td>JNZ</td>
<td>STOP</td>
<td>END IF NO CR</td>
</tr>
<tr>
<td>42</td>
<td>00</td>
<td>04F</td>
<td></td>
<td>JNP</td>
<td>UNTIL</td>
<td>LOOP TIL EOC-1</td>
</tr>
<tr>
<td>43</td>
<td>00</td>
<td>051</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>00</td>
<td>053</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>00</td>
<td>055</td>
<td></td>
<td>MOV</td>
<td>A #51</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>039</td>
<td>35</td>
<td>OUTL</td>
<td>P1 A</td>
<td>;DISABLE 8212</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>03A</td>
<td>F2</td>
<td>MOVL</td>
<td>A R3</td>
<td>;OA ON BUS</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>03B</td>
<td>02</td>
<td>OUTL</td>
<td>BUS A</td>
<td>;PC LOW</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>03C</td>
<td>23L2</td>
<td>MOVL</td>
<td>A #60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>03D</td>
<td>3A</td>
<td>OUTL</td>
<td>P2 A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>03E</td>
<td>2311</td>
<td>MOVL</td>
<td>A #17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>041</td>
<td>3A</td>
<td>OUTL</td>
<td>P2 A</td>
<td>;END STROBE</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>042</td>
<td>61</td>
<td>STOP:</td>
<td>MOVX</td>
<td>A 0H1</td>
<td>;BJS TO 31-Z</td>
</tr>
<tr>
<td>54</td>
<td>043</td>
<td>00</td>
<td>BACK:</td>
<td>INP</td>
<td>BACK</td>
<td>;LOOP TIL EOC=1</td>
</tr>
<tr>
<td>55</td>
<td>044</td>
<td>8643</td>
<td>JSR</td>
<td>BACK</td>
<td>;JUMP TO 04A</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>045</td>
<td>00</td>
<td>WAIT:</td>
<td>NOP</td>
<td>WAIT</td>
<td>;LOOP</td>
</tr>
<tr>
<td>57</td>
<td>046</td>
<td>564B</td>
<td>JR</td>
<td>WAIT</td>
<td>;RETURN</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>047</td>
<td>0446</td>
<td>JMP</td>
<td>_</td>
<td>END</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>048</td>
<td>33</td>
<td>READY:</td>
<td>RETR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SYMBOL TABLE:

START--005
LOOP--020
INT--028
UNTIL--032
STOP--042
BACK--043
WAIT--046
READY--04B

>>> NO ASSEMBLY ERRORS DETECTED <<<
LIST OF REFERENCES


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Distribution List</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Documentation Center  
                        Cameron Station  
                        Alexandria, Virginia  22314 |
| 2.  | 2      | Library, Code 0142  
                        Naval Postgraduate School  
                        Monterey, California 93940 |
| 3.  | 2      | Department Chairman, Code 62  
                        Department of Electrical Engineering  
                        Naval Postgraduate School  
                        Monterey, California 93940 |
| 4.  | 2      | Professor R. Panholzer, Code 62Pz  
                        Department of Electrical Engineering  
                        Naval Postgraduate School  
                        Monterey, California 93940 |
| 5.  | 1      | Associate Professor M. L. Cotton, Code 62Co  
                        Department of Electrical Engineering  
                        Naval Postgraduate School  
                        Monterey, California 93940 |
| 6.  | 1      | LCDR Theodore C. Seward, Jr., USN  
                        203 Ridgewood Street  
                        Mankato, Minnesota 56001 |