A MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE INTEL 8748 MICROCOMPUTER

Theodore Clark Seward

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE INTEL 8748 MICROCOMPUTER

by

Theodore Clark Seward, Jr.

December 1979

Thesis Advisor: R. Panholzer

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE INTEL 8748 MICROCOMPUTER		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1979
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Theodore Clark Seward, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1979
		13. NUMBER OF PAGES 101
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessor development system (MDS)
Intel 8748 Microcomputer
TRS-80

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

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

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

LIST OF FIGURES

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.
9. Single step mode for use in debugging.
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:

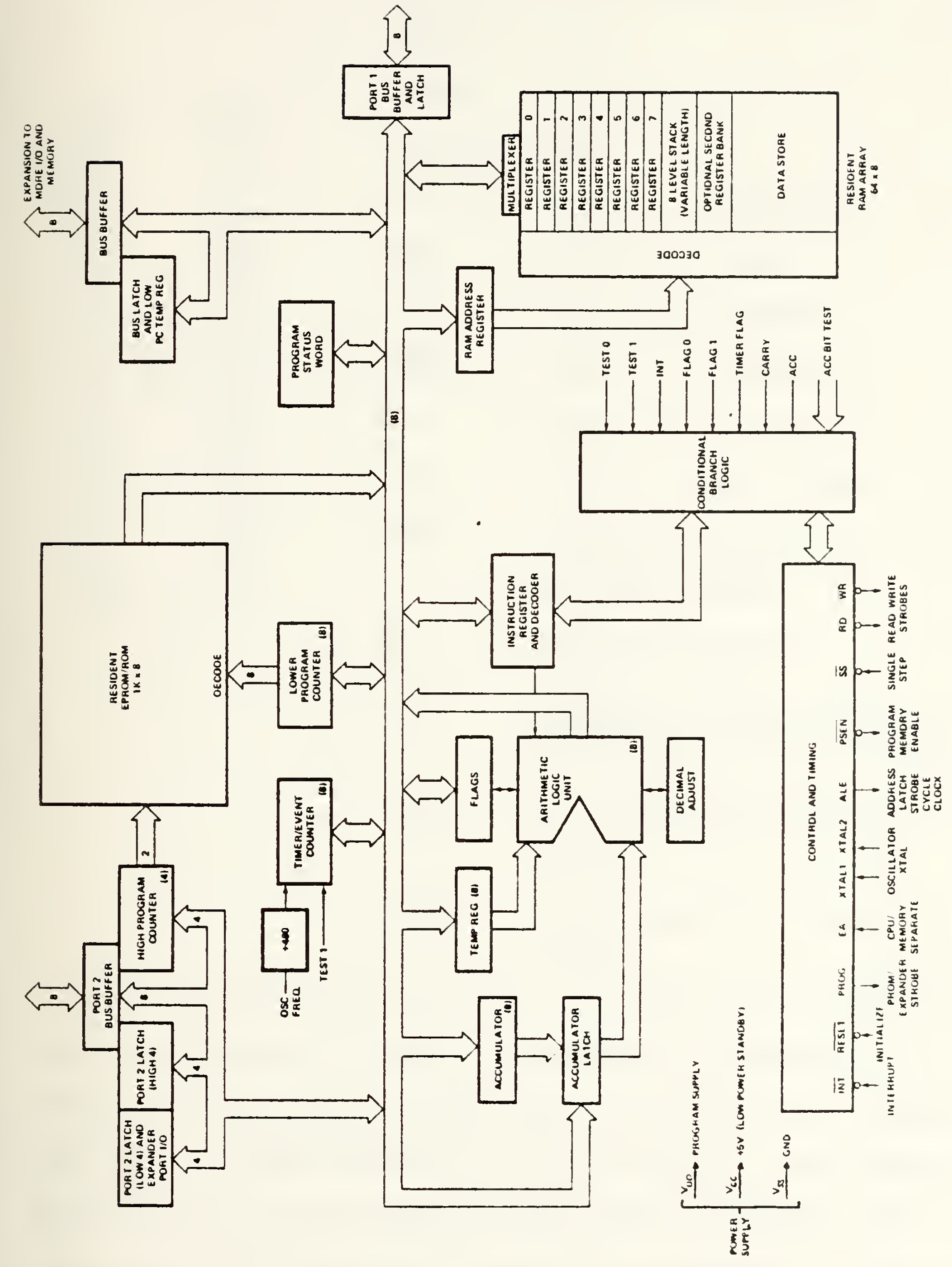
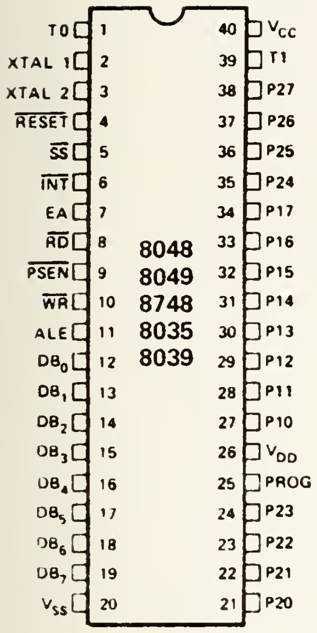


Figure 1 - Intel 8748 Architecture[1]



V_{ss}	Ground
V_{DD}	Programming power supply
V_{cc}	5 volt power supply
P10-P17	Port 1
P20-P27	Port 2
DB_0-DB_7	Bus
T0 & T1	Testable input pins
$\overline{INT}$	Interrupt input
$\overline{RD}$	Output read strobe
$\overline{RESET}$	Input to initialize
$\overline{WR}$	Output write strobe
ALE	Address Latch Enable
$\overline{PSEN}$	Program Store Enable. Used during fetch to external memory.
$\overline{SS}$	Single step input
EA	External Access input. Forces all program memory fetches to external memory.
XTAL 1	One side of crystal input or input for external clock.
XTAL 2	Other side of crystal input.

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.

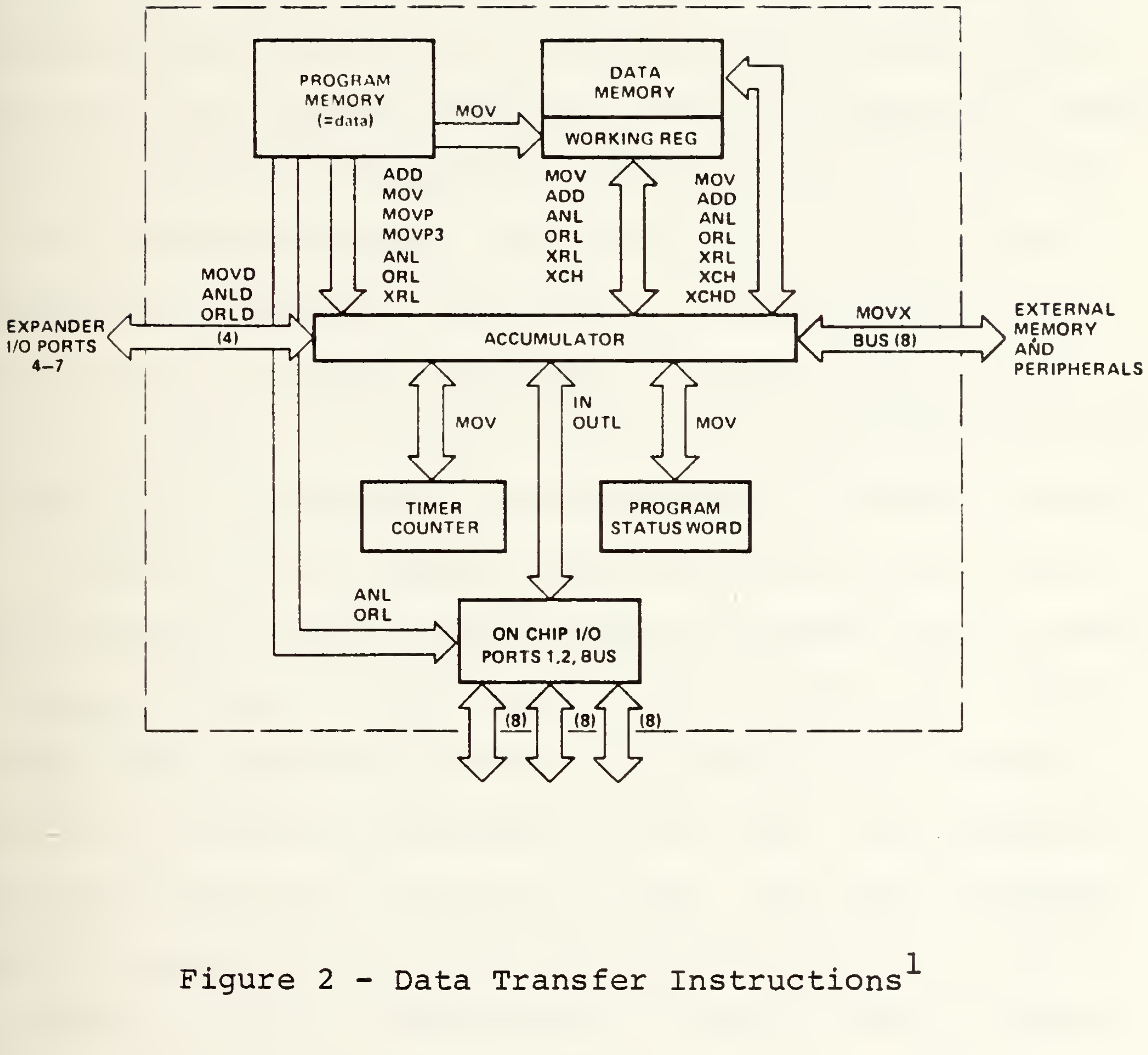


Figure 2 - Data Transfer Instructions[1]

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).

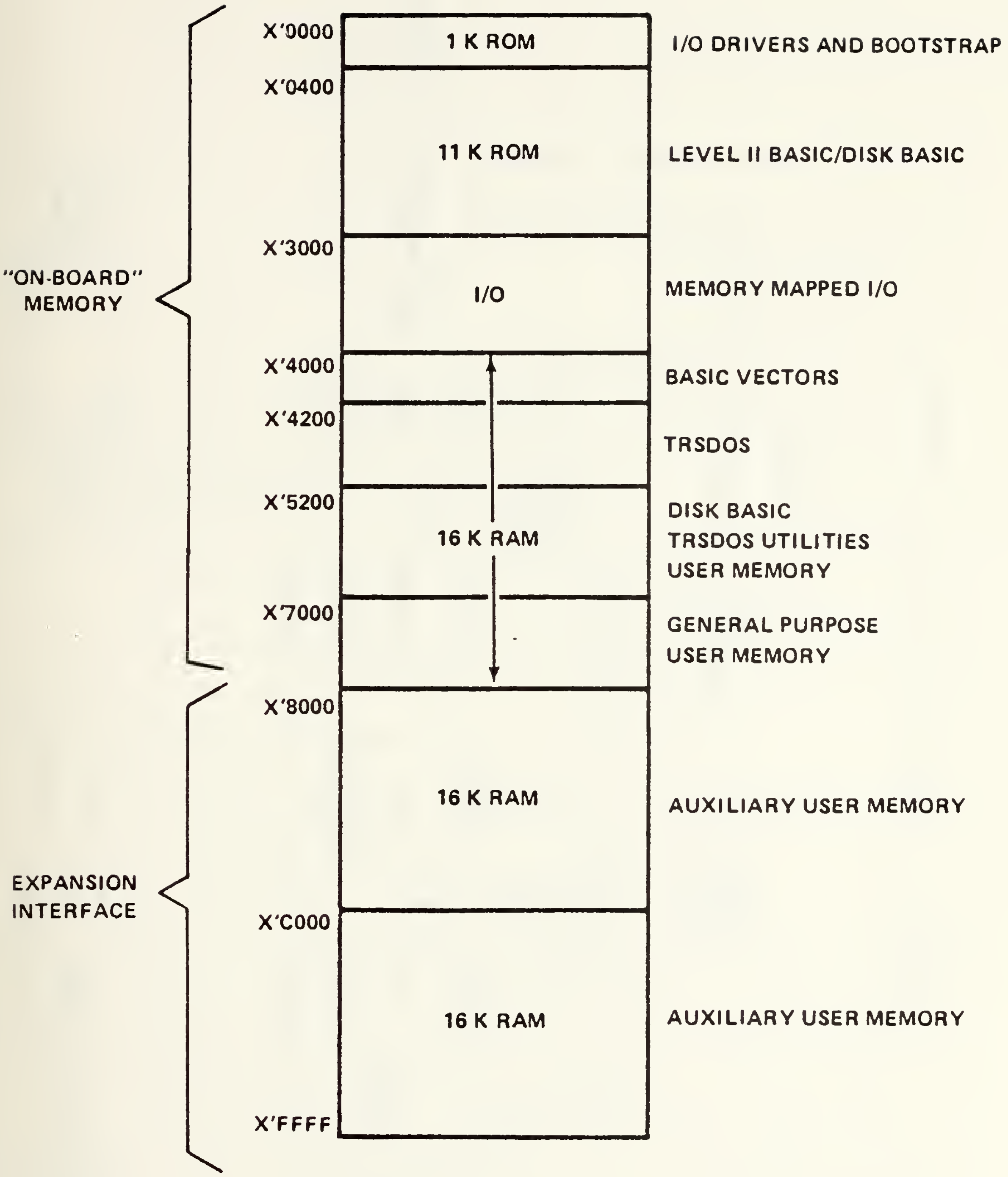


Figure 3 - TRS-80 Memory Map

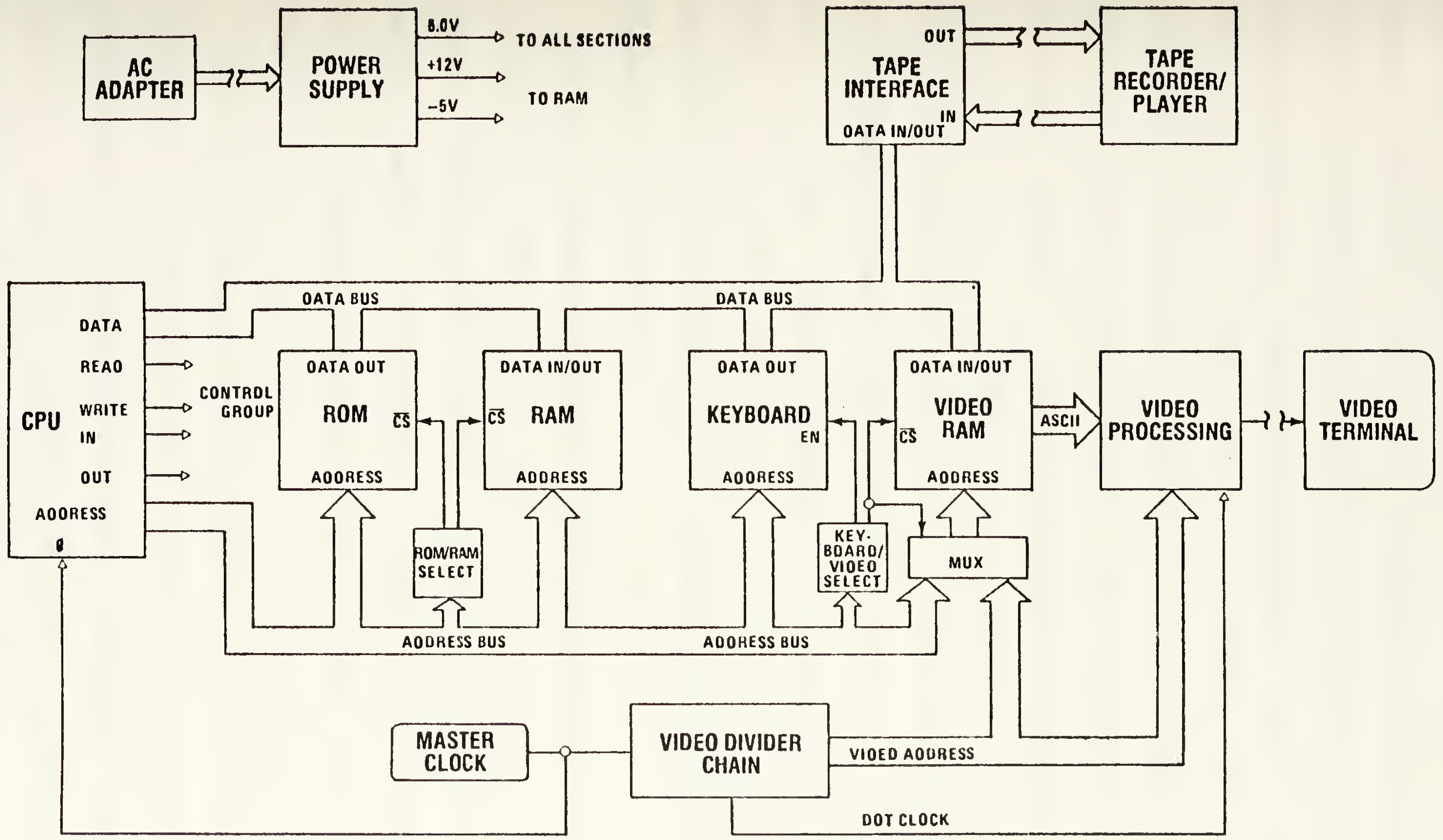


Figure 4 - TRS-80 System Block Diagram[8]

This OS is analagous 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,

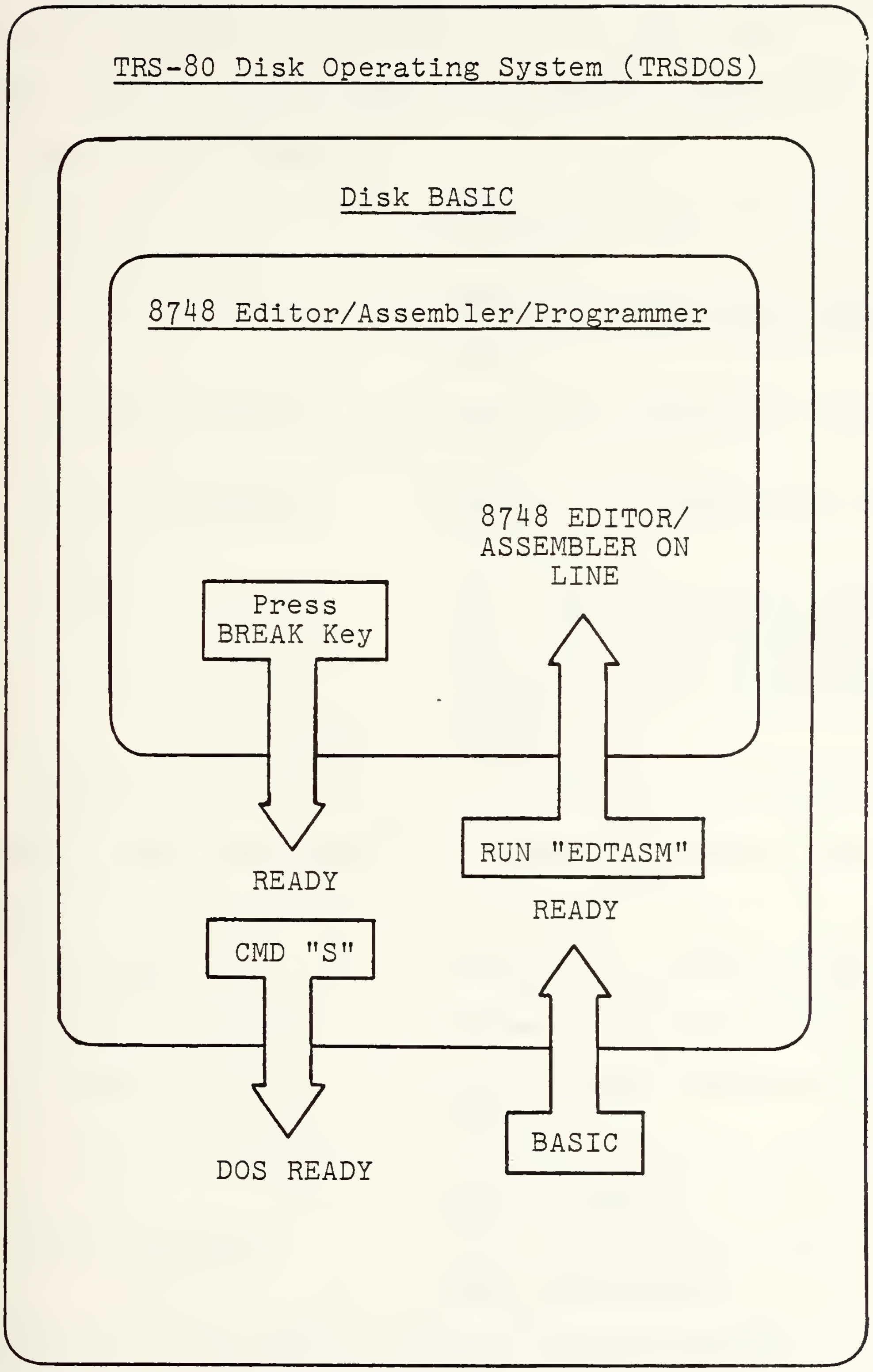
Radio Shack sells several printers which are driven by the TRS-80 parallel output. The first attempt at interfacing was conducted as part of a student thesis. This project involved construction of a parallel to serial conversion device using standard Universal Asynchronous Receiver Transmitter (UART) 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



TRS-80 Disk Operating System (TRSDOS)
Disk BASIC
8748 Editor/Assembler/Programmer
8748 EDITOR/ ASSEMBLER ON LINE
Press BREAK Key
READY
CMD "S"
DOS READY
RUN "EDTASM"
READY
BASIC

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.

IV. MICROPROCESSOR DEVELOPMENT SYSTEM SOFTWARE

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

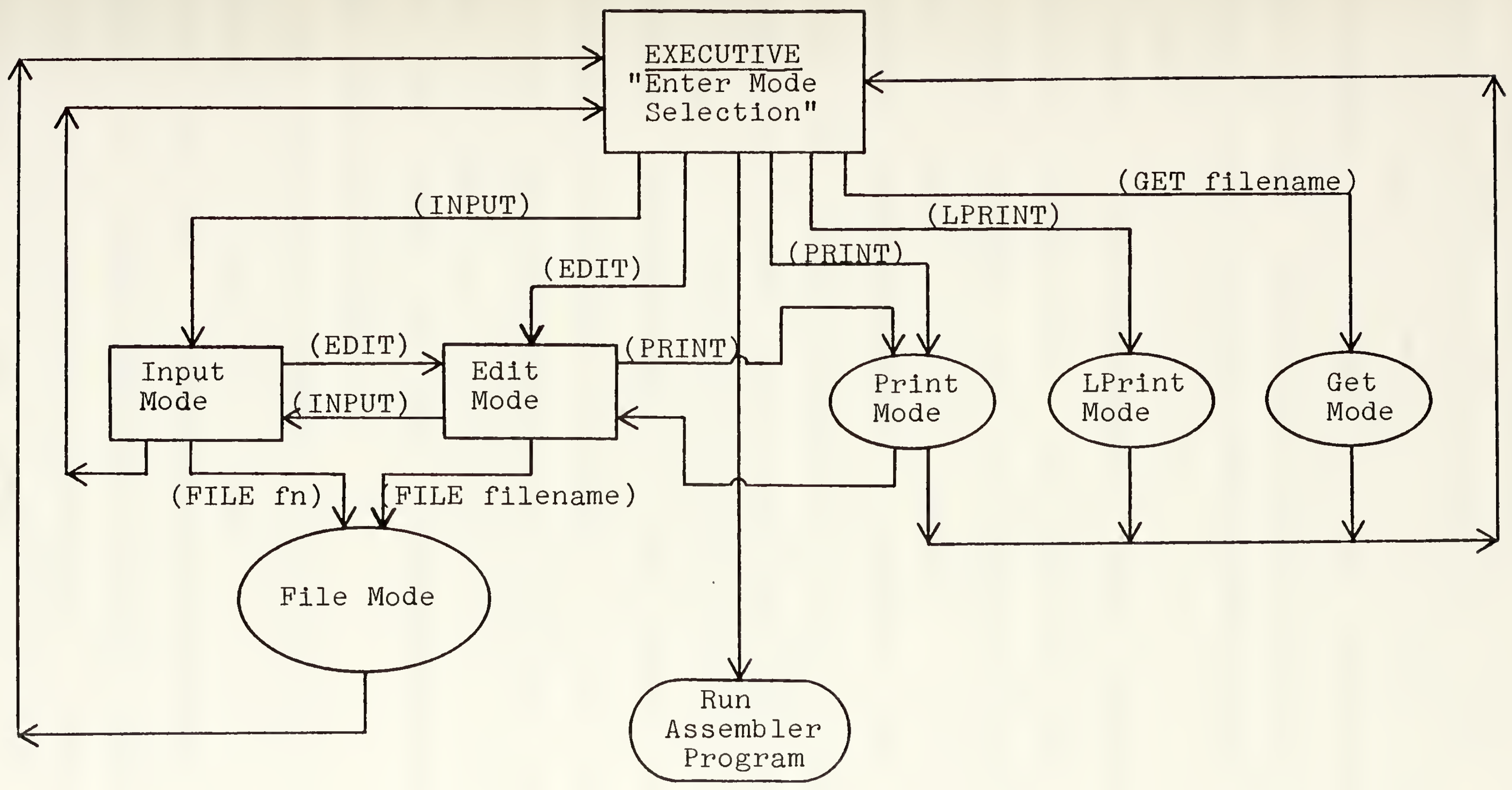


Figure 6 - Editor Program Block Diagram

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 <u>above</u> 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:

```
LINE NUMBER   HEX ADDRESS   MNEMONIC CODE   COMMENTS
```

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 twofold. 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:

Line	Address	Code	Label	Opcode	Operand
03	03	0407		JMP	BACK
04	05	8909		MOV	R1,#9
05	07	59	BACK	ANL	A,R1

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 approp- riate 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 dur- ing 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 as- signs 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 mis- spelled.

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:

<u>Line No</u>. <u>Hex Address</u> <u>Hex Code</u> <u>Label</u> <u>Opcode</u> <u>Operand</u> <u>Comments</u>

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

microcomputer. 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

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 NOT 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
    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 $\overline{\text{OUT}}$ 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 $\overline{\text{IN}}$ line low while outputting the port number on the lower eight

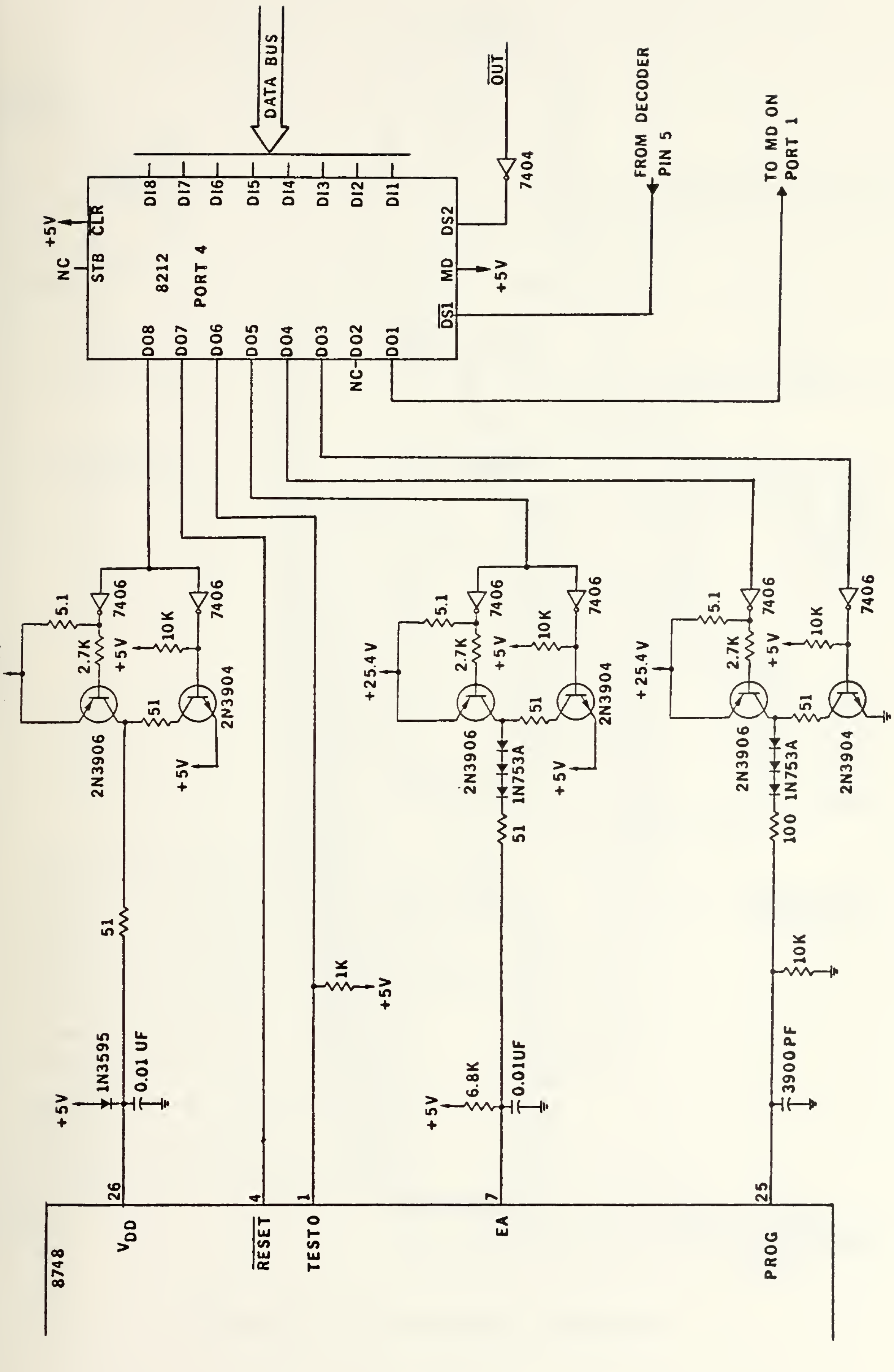


Figure 7 - Programmer Schematic

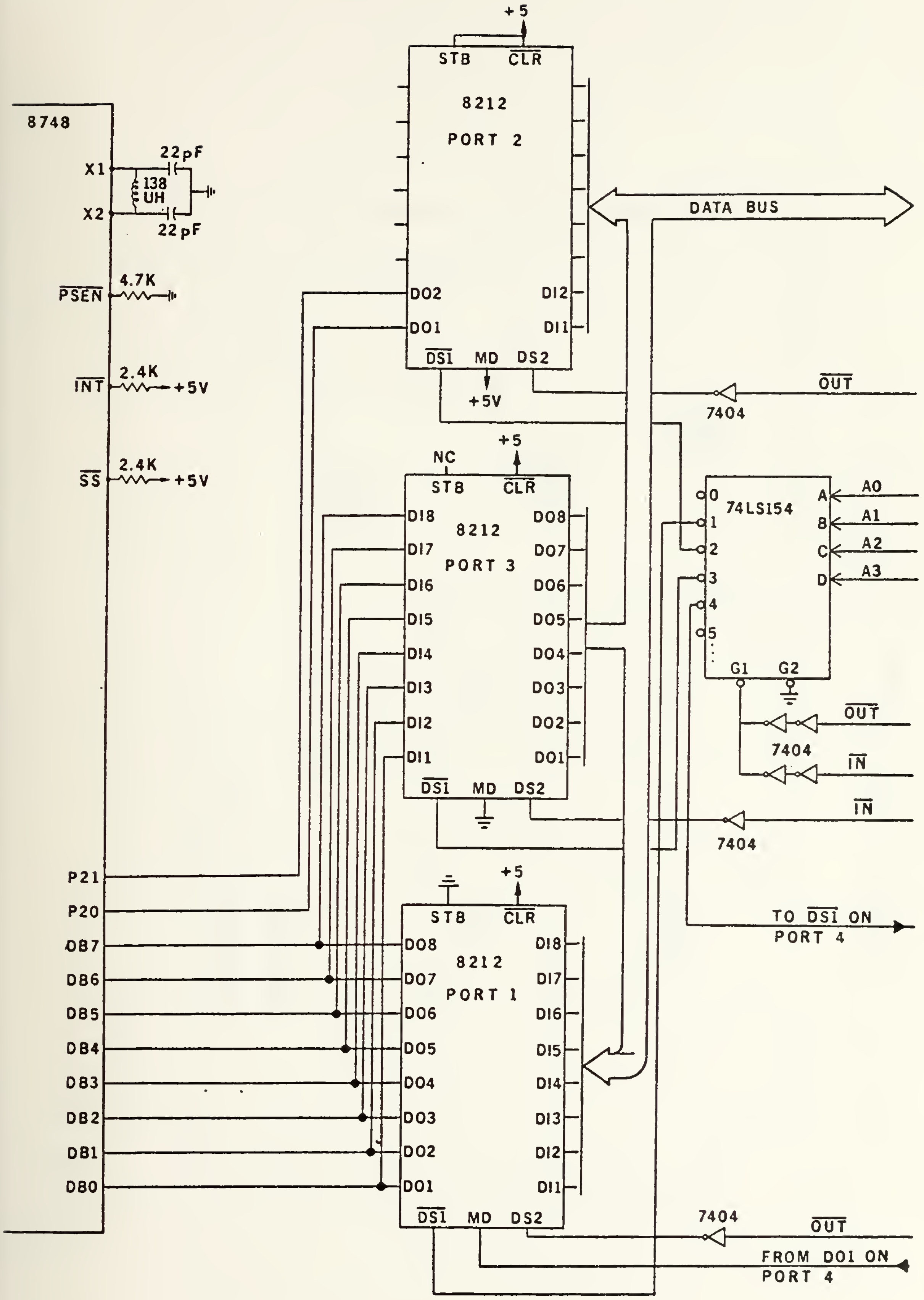


Figure 8 - Programmer Schematic

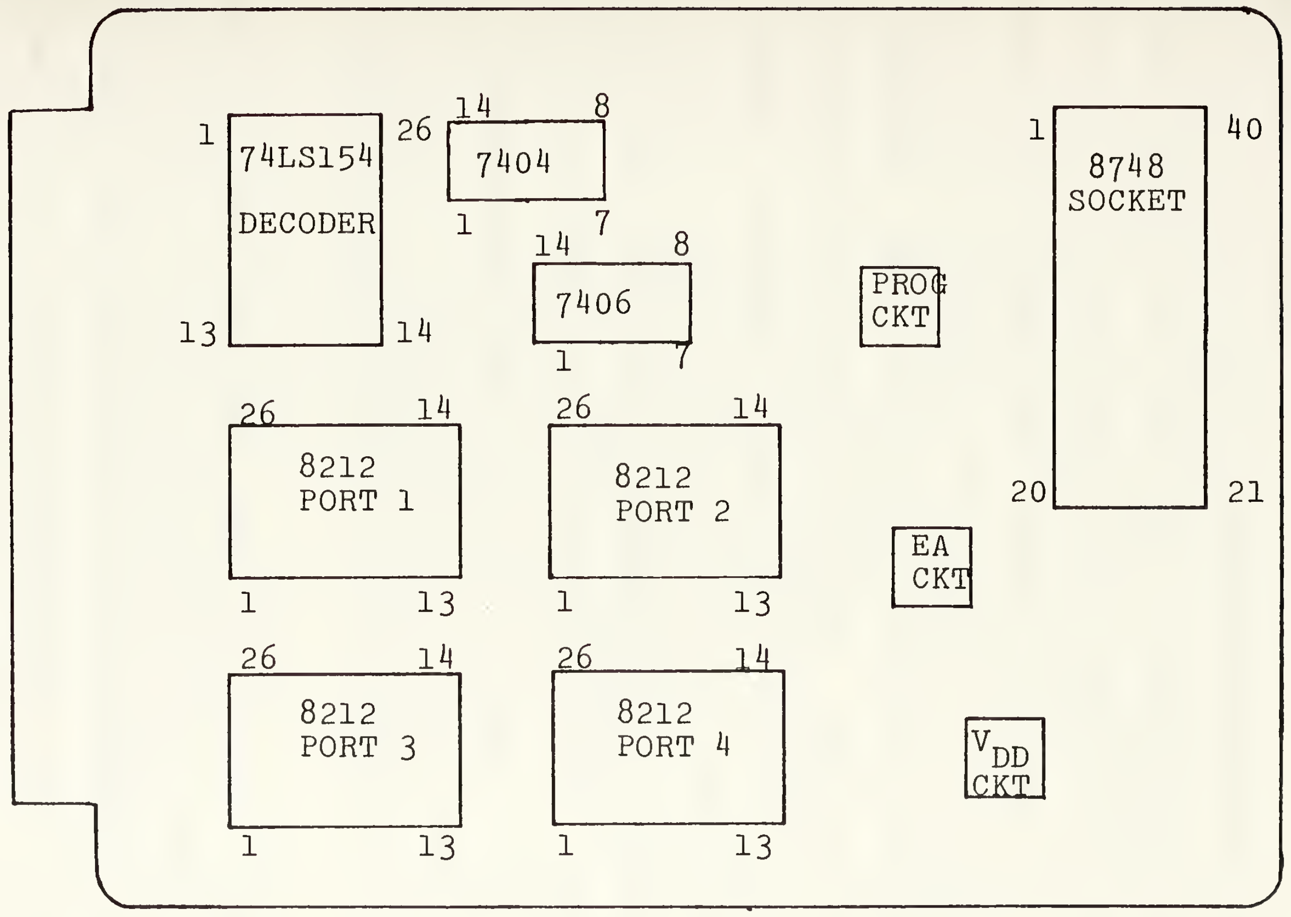


Figure 9 - Programmer Circuit Board

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 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

P/N	SIGNAL NAME	DESCRIPTION
1	RAS*	Row Address Strobe Output for 16-Pin Dynamic Rams
2	SYSRES*	System Reset Output, Low During Power Up Initialize or Reset Depressed
3	CAS*	Column Address Strobe Output for 16-Pin Dynamic Rams
4	A1Ø	Address Output
5	A12	Address Output
6	A13	Address Output
7	A15	Address Output
8	GND	Signal Ground
9	A11	Address Output
10	A14	Address Output
11	A8	Address Output
12	OUT*	Peripheral Write Strobe Output
13	WR*	Memory Write Strobe Output
14	INTAK*	Interrupt Acknowledge Output
15	RD*	Memory Read Strobe Output
16	MUX	Multiplexor Control Output for 16-Pin Dynamic Rams
17	A9	Address Output
18	D4	Bidirectional Data Bus
19	IN*	Peripheral Read Strobe Output
20	D7	Bidirectional Data Bus
21	INT*	Interrupt Input (Maskable)
22	D1	Bidirectional Data Bus
23	TEST*	A Logic "Ø" on TEST* Input Tri-States AØ-A15, DØ-D7, WR*, RD*, IN*, OUT*, RAS*, CAS*, MUX*
24	D6	Bidirectional Data Bus
25	AØ	Address Output
26	D3	Bidirectional Data Bus
27	A1	Address Output
28	D5	Bidirectional Data Bus
29	GND	Signal Ground
30	DØ	Bidirectional Data Bus
31	A4	Address Bus
32	D2	Bidirectional Data Bus
33	WAIT*	Processor Wait Input, to Allow for Slow Memory
34	A3	Address Output
35	A5	Address Output
36	A7	Address Output
37	GND	Signal Ground
38	A6	Address Output
39	GND	Signal Ground
40	A2	Address Output
NOTE: *means Negative (Logical "Ø") True Input or Output		

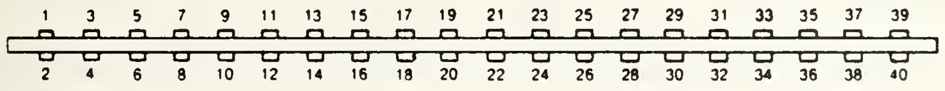
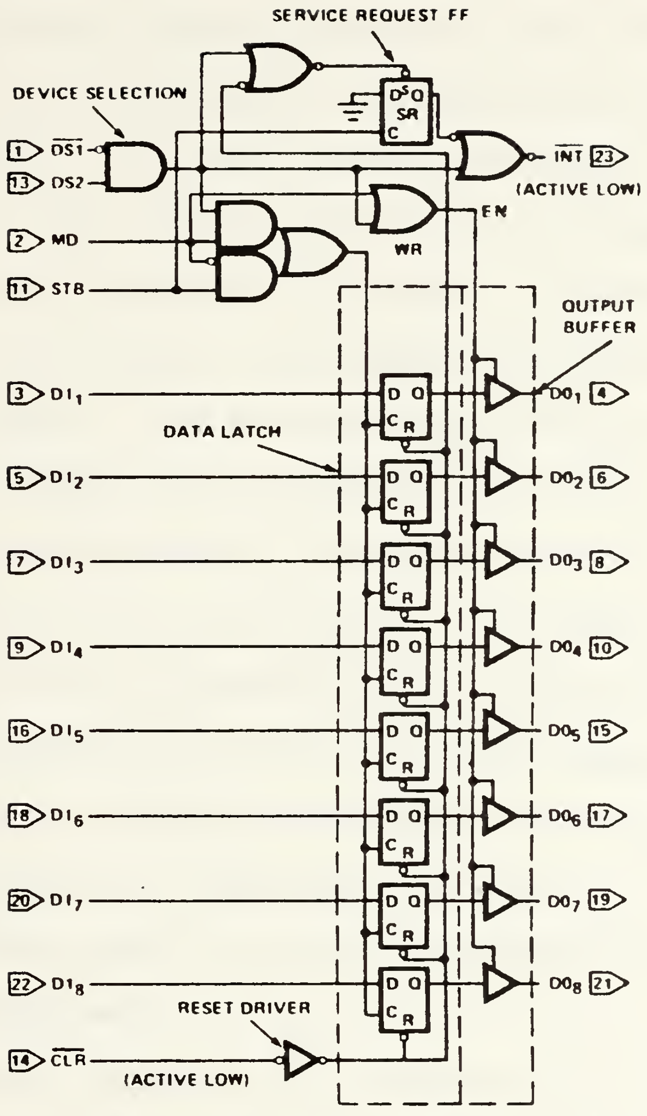


Figure 10 - Pinout of Expansion Interface[8]

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 $\overline{\text{RESET}}$ 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



STB	MD	$(\overline{DS_1} \cdot DS_2)$	DATA OUT EQUALS
0	0	0	3-STATE
1	0	0	3-STATE
0	1	0	DATA LATCH
1	1	0	DATA LATCH
0	0	1	DATA LATCH
1	0	1	DATA IN
0	1	1	DATA IN
1	1	1	DATA IN

$\overline{CLR}$ – RESETS DATA LATCH
SETS SR FLIP FLOP
(NO EFFECT ON OUTPUT BUFFER)

CLR	$(\overline{DS_1} \cdot DS_2)$	STB	*SR	INT
0	0	0	1	1
0	1	0	1	0
1	1	‾_	0	0
1	1	0	1	0
1	0	0	1	1
1	1	‾_	1	0

*INTERNAL SR FLIP FLOP

Figure 11 - Intel 8212 I/O Port Schematic[1]

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.

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
V_{DD}	$\overline{\text{RESET}}$	TESTO	EA	PROG1	PROG2	N.C.	PORT 1 MD

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 $\overline{\text{RESET}}$ 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 $\overline{\text{RESET}}$ 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.

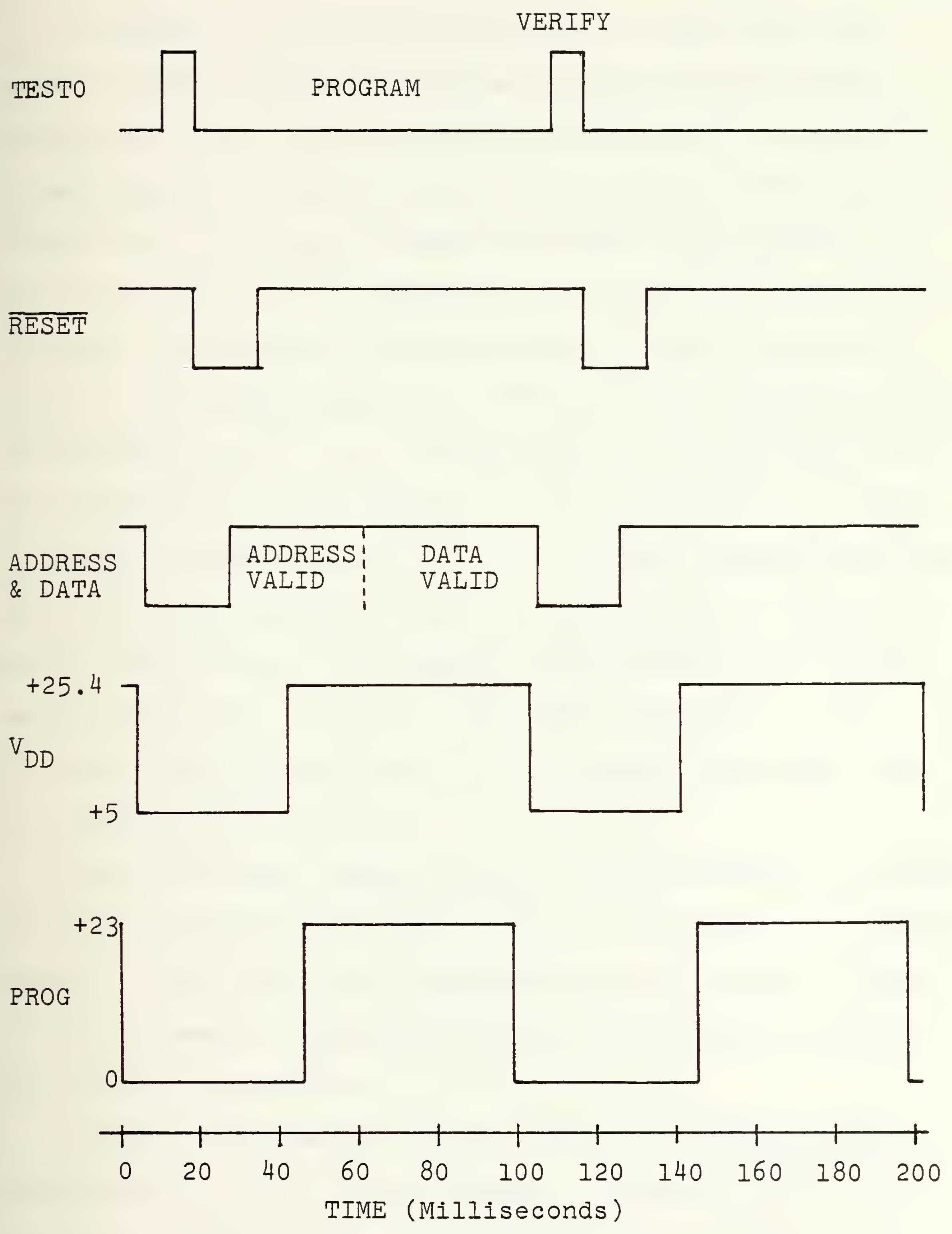


Figure 12 - Programmer Timing Diagram

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 mother board. 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

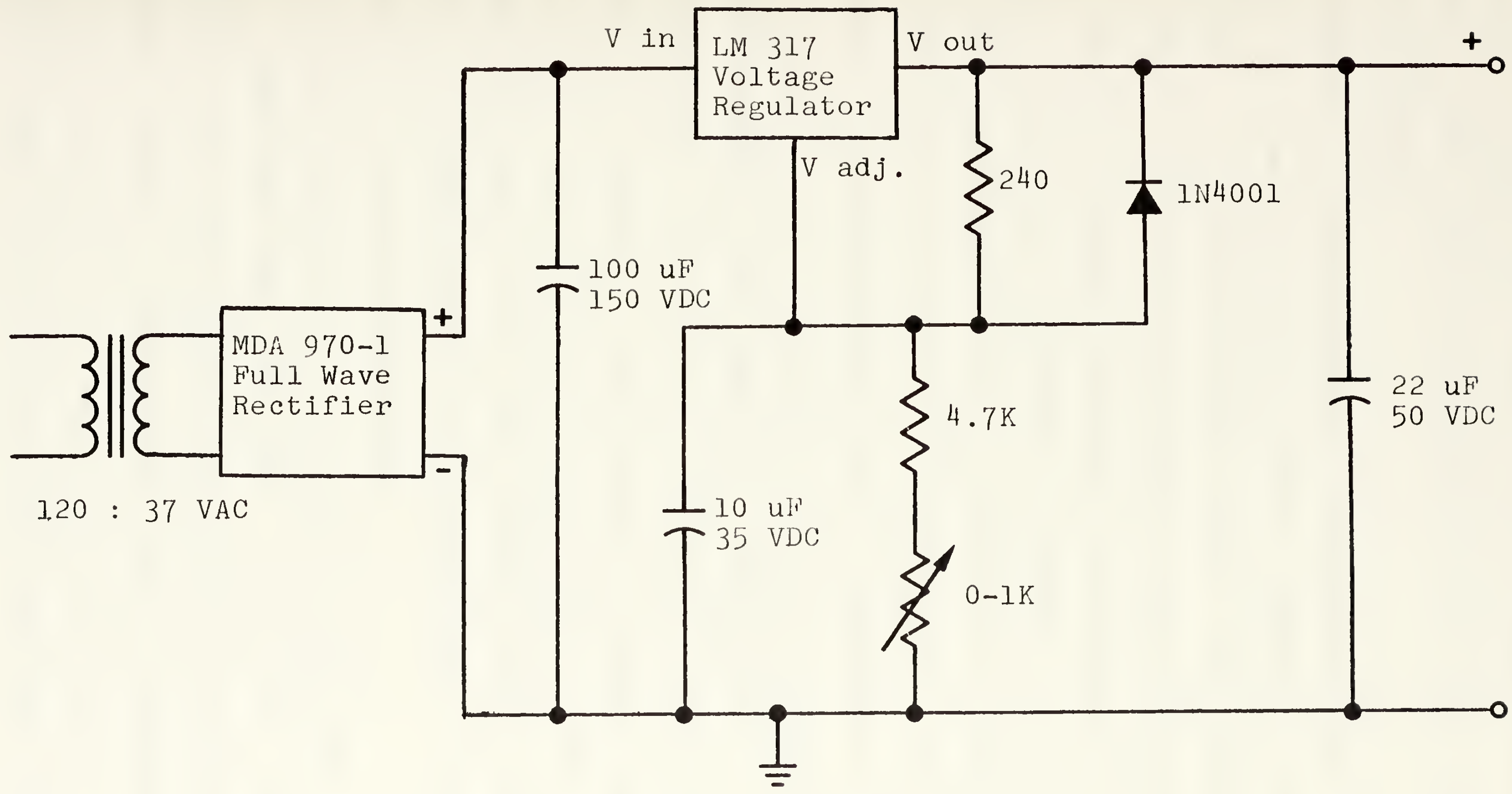


Figure 13 - 25.4 Volt Power Supply

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, $\overline{\text{OUT}}$ and $\overline{\text{IN}}$. 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 <u>raison d'etre</u> of the microprocessor development system and is, not coincidentally, the path of the future.

APPENDIX A

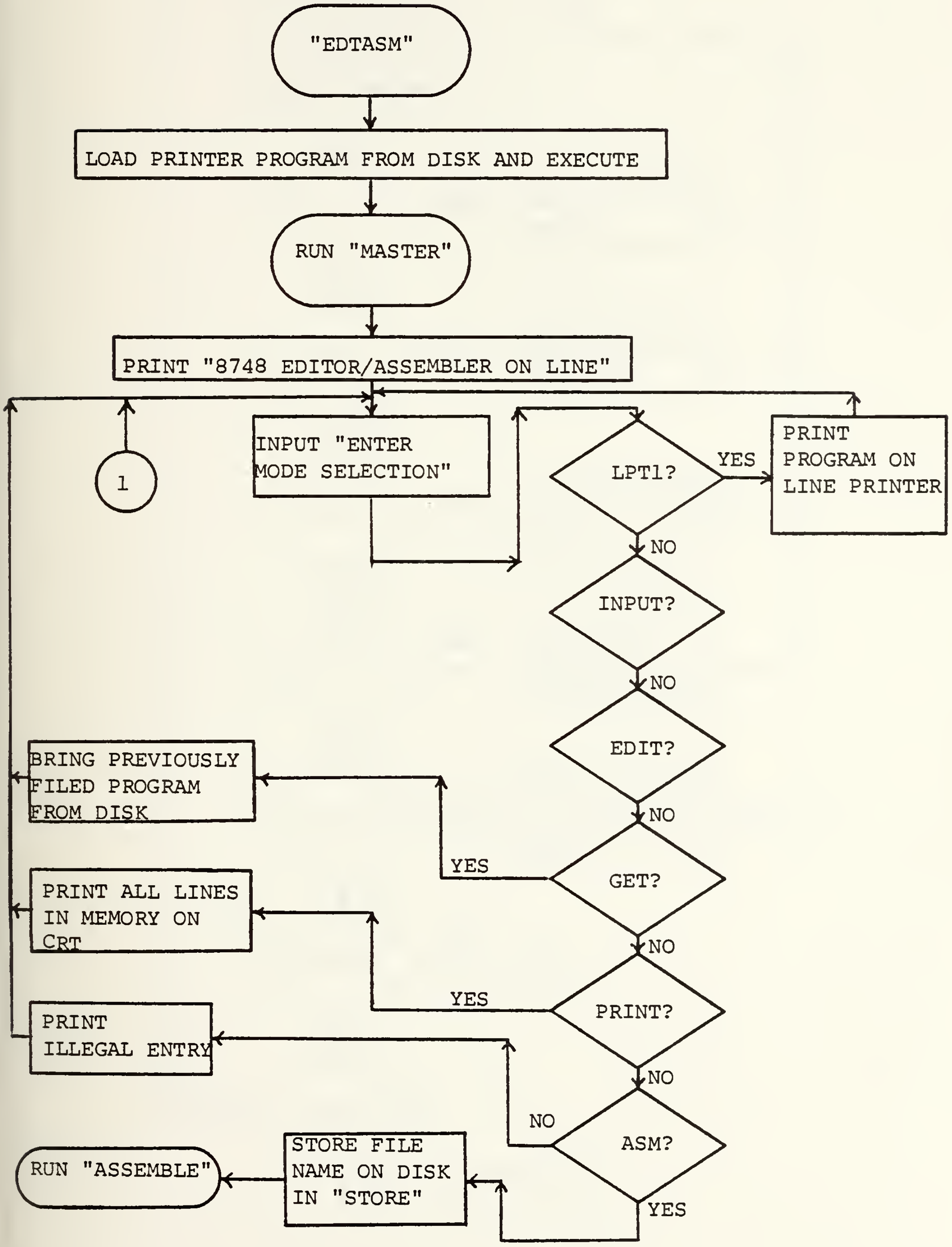
PROGRAM LISTING FOR PRINTER SUBROUTINE

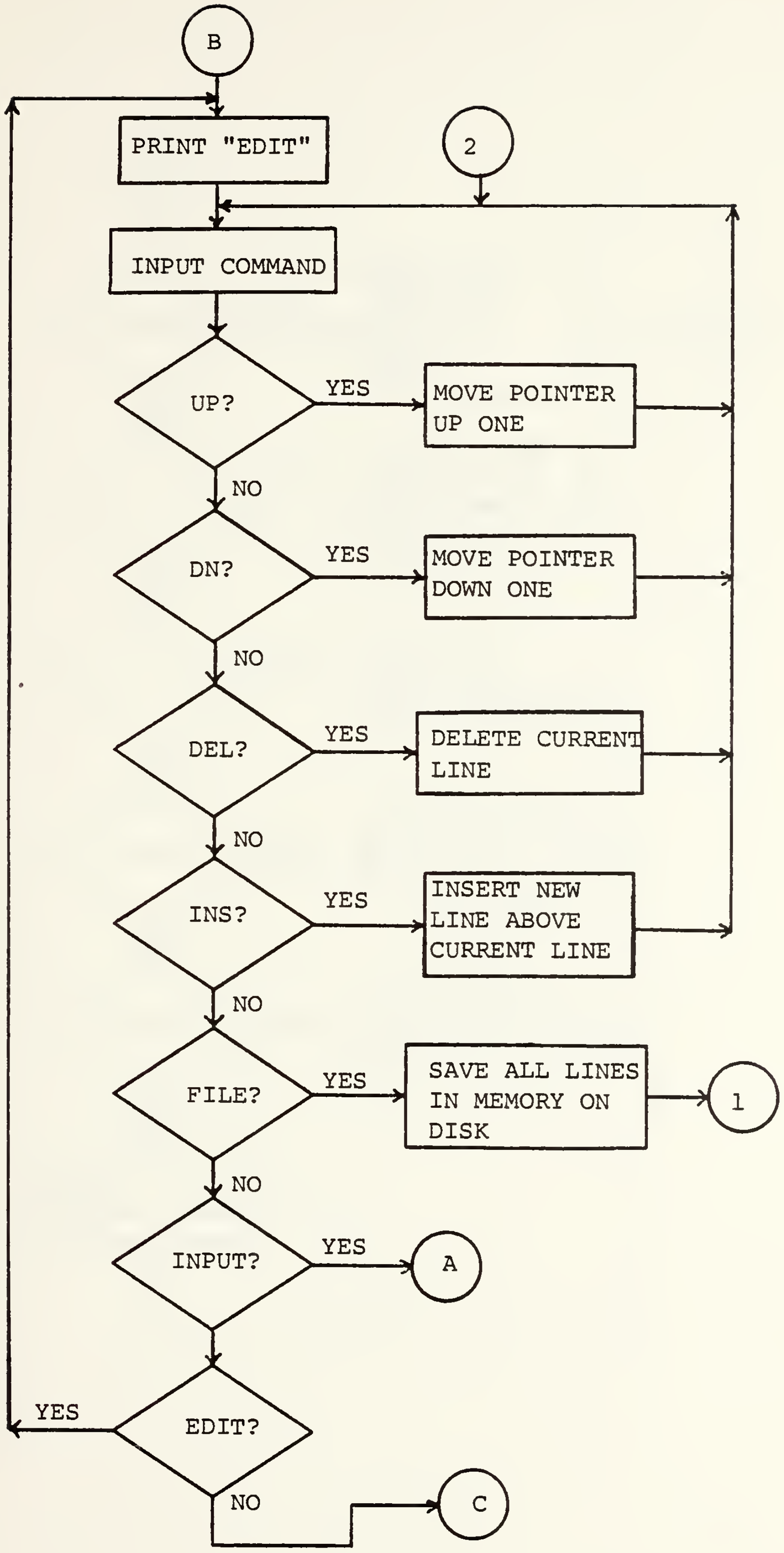
```
Location      Hex Code     Label      Opcode    Operand

 FF00          E5          INIT        PUSH      HL
 FF01          C5                      PUSH      BC
 FF02          F5                      PUSH      AF
 FF03          3A48FF                  LD        A,(FLAG)
 FF06          FE01                    CP        01H
 FF08          2820                    JR        Z,RESTOR
 FF0A          3E01                    LD        A,01H
 FF0C          3248FF                  LD        (FLAG),A
 FF0F          D3E8                    OUT       E8,A
 FF11          DBE9                    IN        AE9
 FF13          E6F8                    AND       0F8H
 FF15          F604                    OR        04H
 FF17          3247FF                  LD        (SWTIMG),A
 FF1A          D3EA                    OUT       EA,A
 FF1C          DBE9        BAUDST      IN        A,E9
 FF1E          E607                    AND       07H
 FF20          213FFF                  LD        HL,BDTABL
 FF23          0600                    LD        B,00H
 FF25          4F                      LD        C,A
 FF26          09                      ADD       HL,BC
 FF27          7E                      LD        A,(HL)
 FF28          D3E9                    OUT       E9,A
 FF2A          F1          RESTOR      POP       AF
 FF2B          C1                      POP       BC
 FF2C          E1                      POP       HL
 FF2D          DBEA        STATIN      IN        A,EA
 FF2F          CB77                    BIT       6,A
 FF31          28FA                    JR        Z,STATIN
 FF33          79          LD          A,C
 FF34          D3EB                    OUT       EB,A
 FF36          FEOD                    CP        0DH
 FF38          2004                    JR        NZ,RETRN
 FF3A          C35OFF                  JMP       FF50
 FF3D          00                      NOP
 FF3E          C9          RETRN       RET
 FF3F          22          BDTABL      DEFB      22H
 FF40          44                      DEFB      44H
 FF41          55                      DEFB      55H
 FF42          66                      DEFB      66H
 FF43          77                      DEFB      77H
 FF44          AA                      DEFB      0AAH
 FF45          CC                      DEFB      0CCH
 FF46          EE                      DEFB      0EEH
 FF47          00          SWTIMG      DEFB      00H
 FF48          00          FLAG        DEFB      00H
 FF49          00                      NOP
```

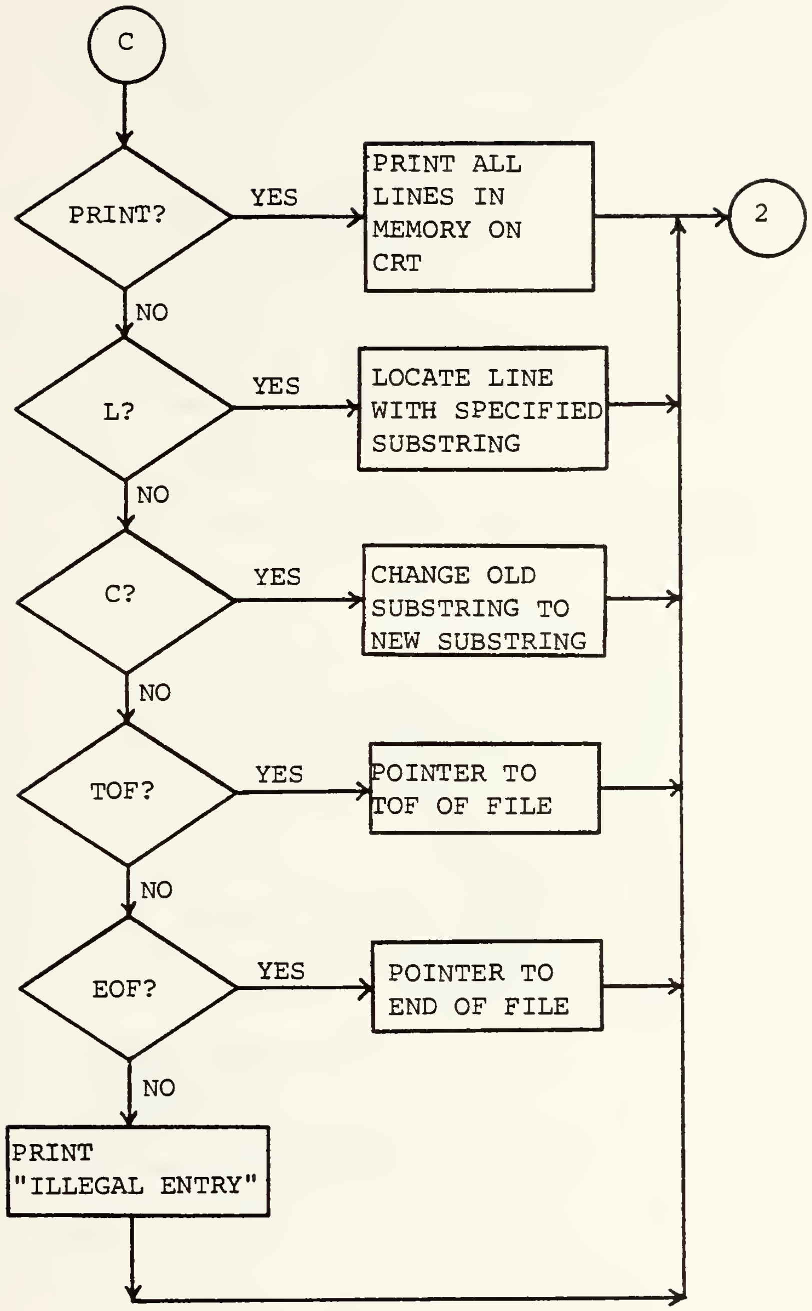
```
FF50           00                    NOP
FF51           00                    NOP
FF52           00                    NOP
FF53           00                    NOP
FF54           00                    NOP
FF55           00                    NOP
FF56           00                    NOP
FF57           E5                    PUSH      HL
FF58           21FF44                LD        HL,44
FF5A           2B                    DEC       HL
FF5B           7C                    LD        A,H
FF5C           B5                    OR        L
FF5D           C254FF                JP        NZ,FF54
FF60           E1                    POP       HL
FF61           OEOA                  LD        C,OAH
FF63           C32DFF                JP        FF2D
```

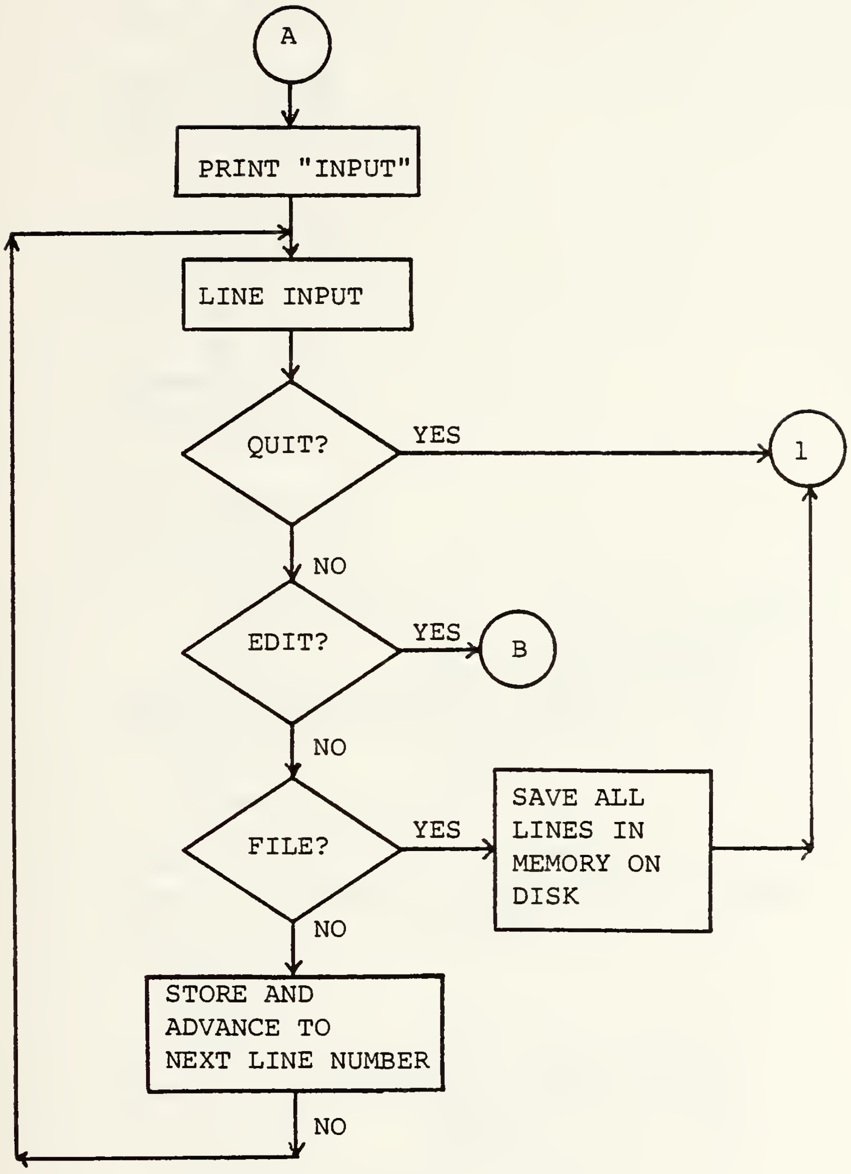
APPENDIX B

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?
YES
PRINT PROGRAM ON LINE PRINTER
NO
INPUT?
NO
EDIT?
NO
BRING PREVIOUSLY FILED PROGRAM FROM DISK
YES
GET?
NO
PRINT ALL LINES IN MEMORY ON CRT
YES
PRINT?
NO
PRINT ILLEGAL ENTRY
NO
ASM?
YES
STORE FILE NAME ON DISK IN "STORE"
RUN "ASSEMBLE"

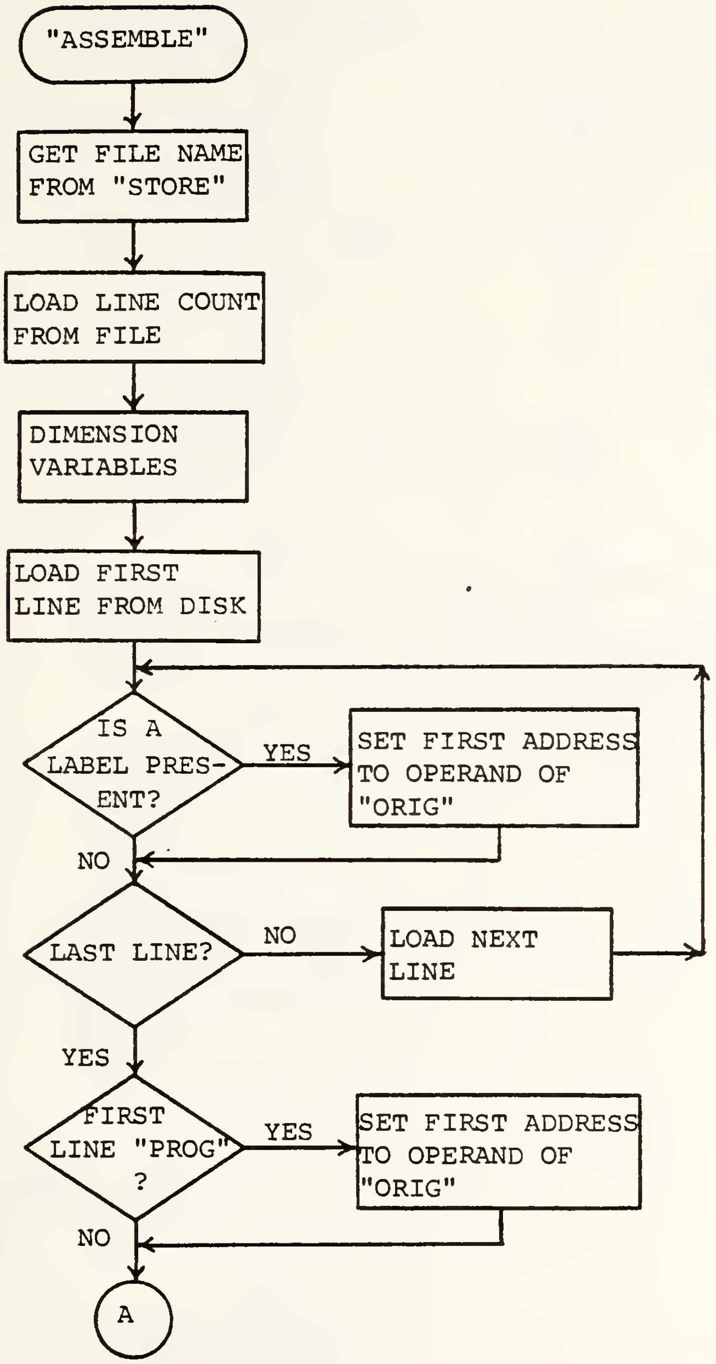

B
PRINT "EDIT"
2
INPUT COMMAND
UP?
YES
MOVE POINTER UP ONE
NO
DN?
YES
MOVE POINTER DOWN ONE
NO
DEL?
YES
DELETE CURRENT LINE
NO
INS?
YES
INSERT NEW LINE ABOVE CURRENT LINE
NO
FILE?
YES
SAVE ALL LINES IN MEMORY ON DISK
1
NO
INPUT?
YES
A
YES
EDIT?
NO
C

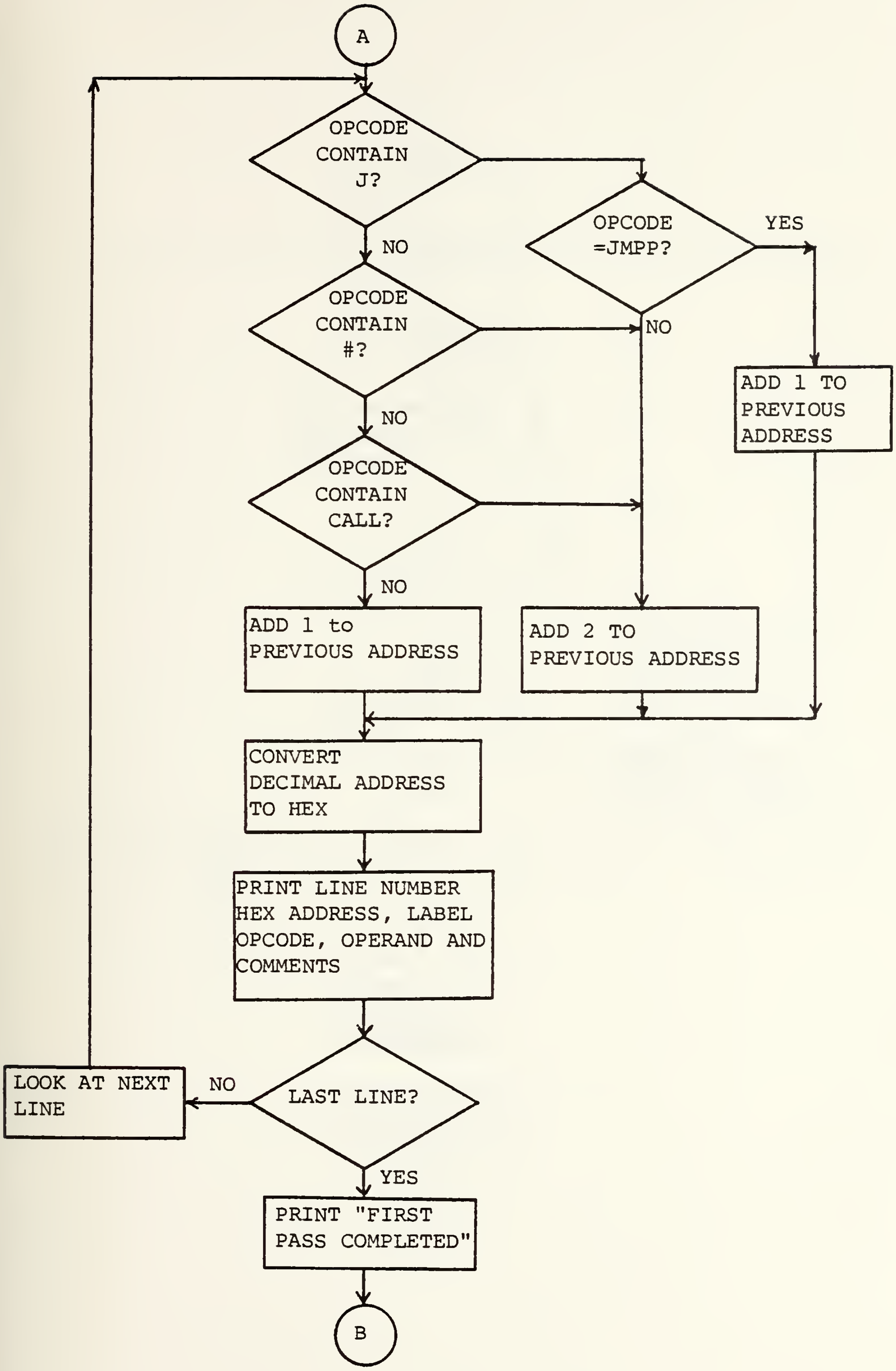

C
PRINT?
YES
PRINT ALL LINES IN MEMORY ON CRT
2
NO
L?
YES
LOCATE LINE WITH SPECIFIED SUBSTRING
NO
C?
YES
CHANGE OLD SUBSTRING TO NEW SUBSTRING
NO
TOF?
YES
POINTER TO TOF OF FILE
NO
EOF?
YES
POINTER TO END OF FILE
NO
PRINT "ILLEGAL ENTRY"

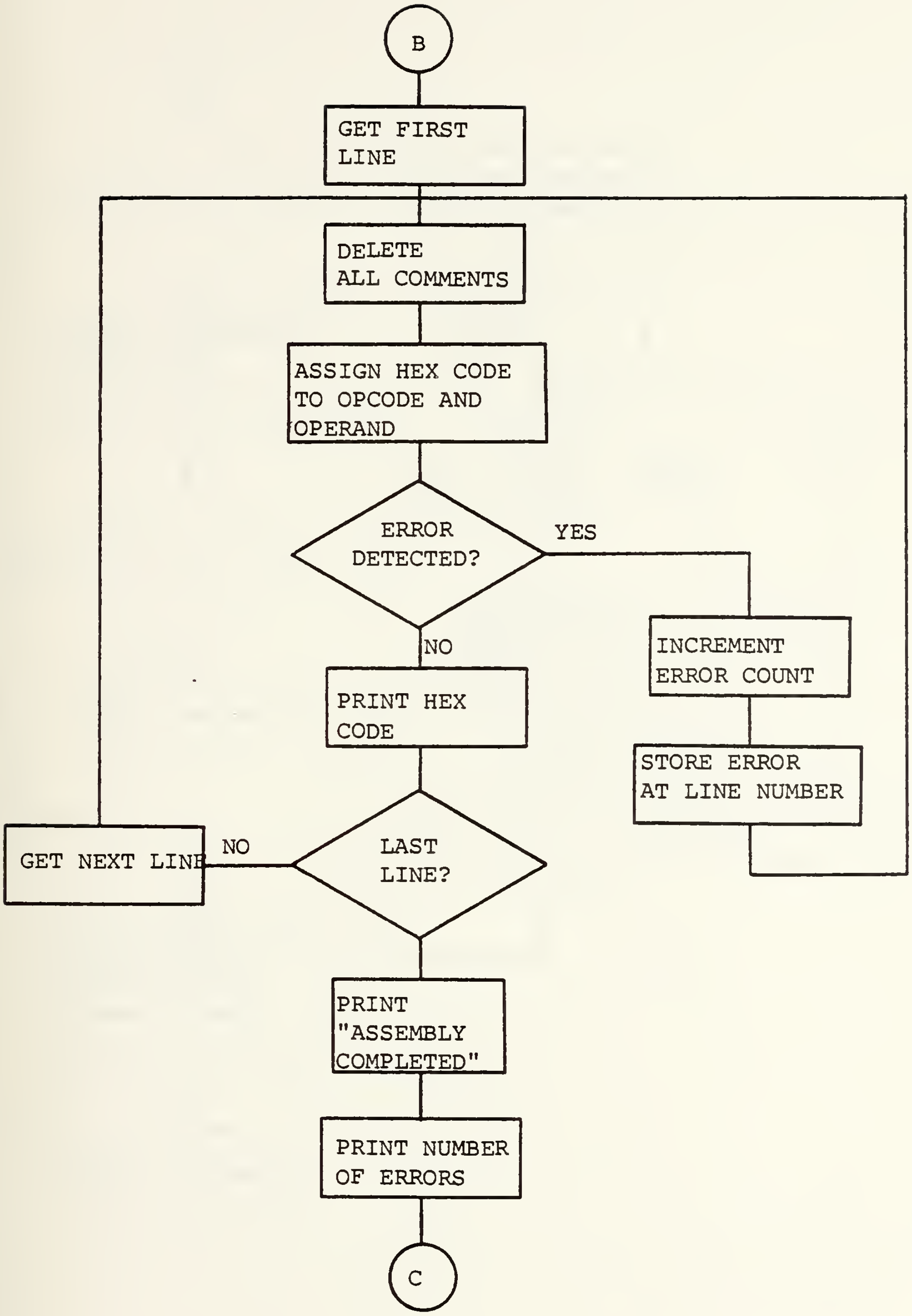

A
PRINT "INPUT"
LINE INPUT
QUIT?
YES
1
NO
EDIT?
YES
B
NO
FILE?
YES
SAVE ALL LINES IN MEMORY ON DISK
NO
STORE AND ADVANCE TO NEXT LINE NUMBER
NO

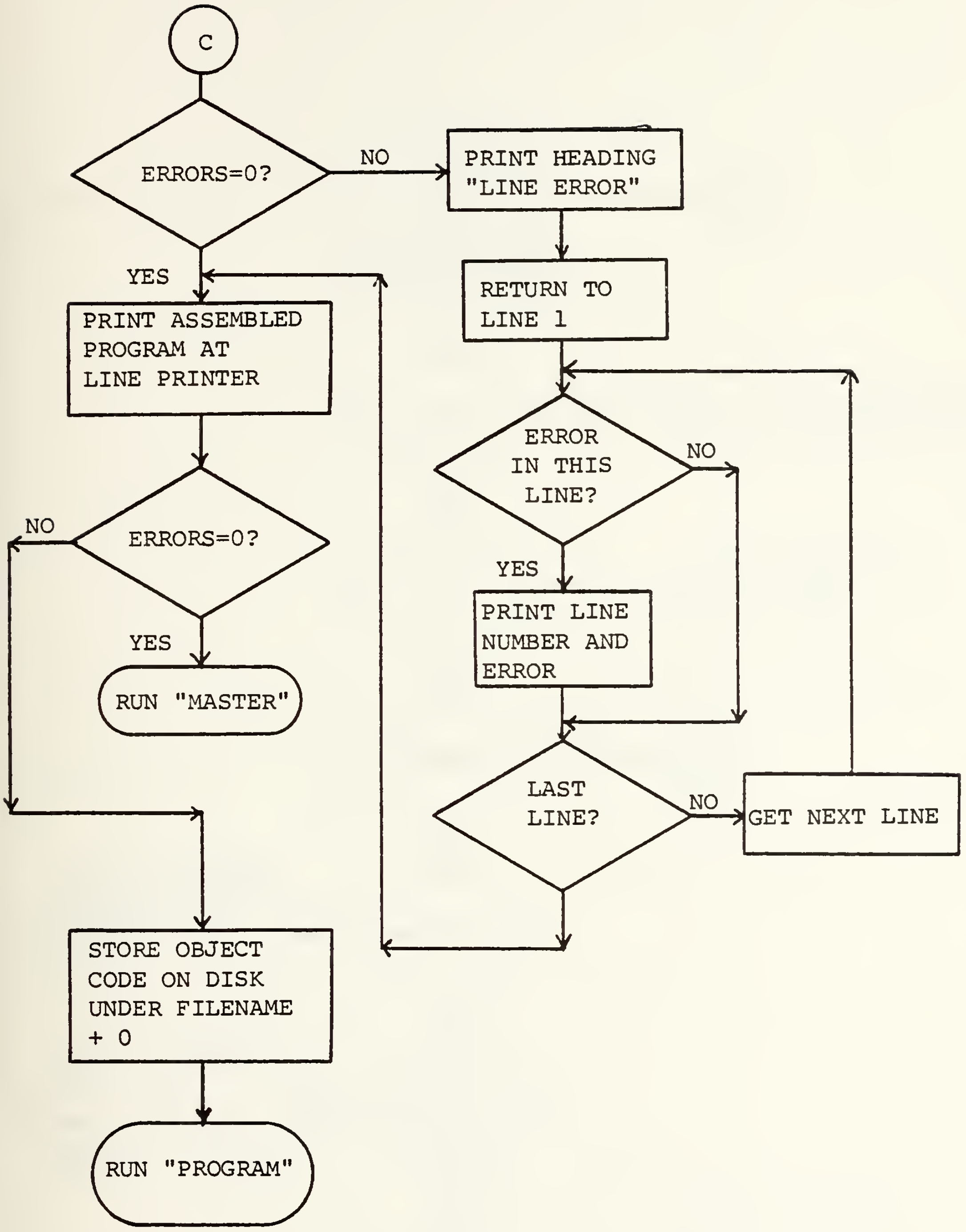
APPENDIX C

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 PRES-ENT?
YES
SET FIRST ADDRESS TO OPERAND OF "ORIG"
NO
LAST LINE?
NO
LOAD NEXT LINE
YES
FIRST LINE "PROG" ?
YES
SET FIRST ADDRESS TO OPERAND OF "ORIG"
NO
A

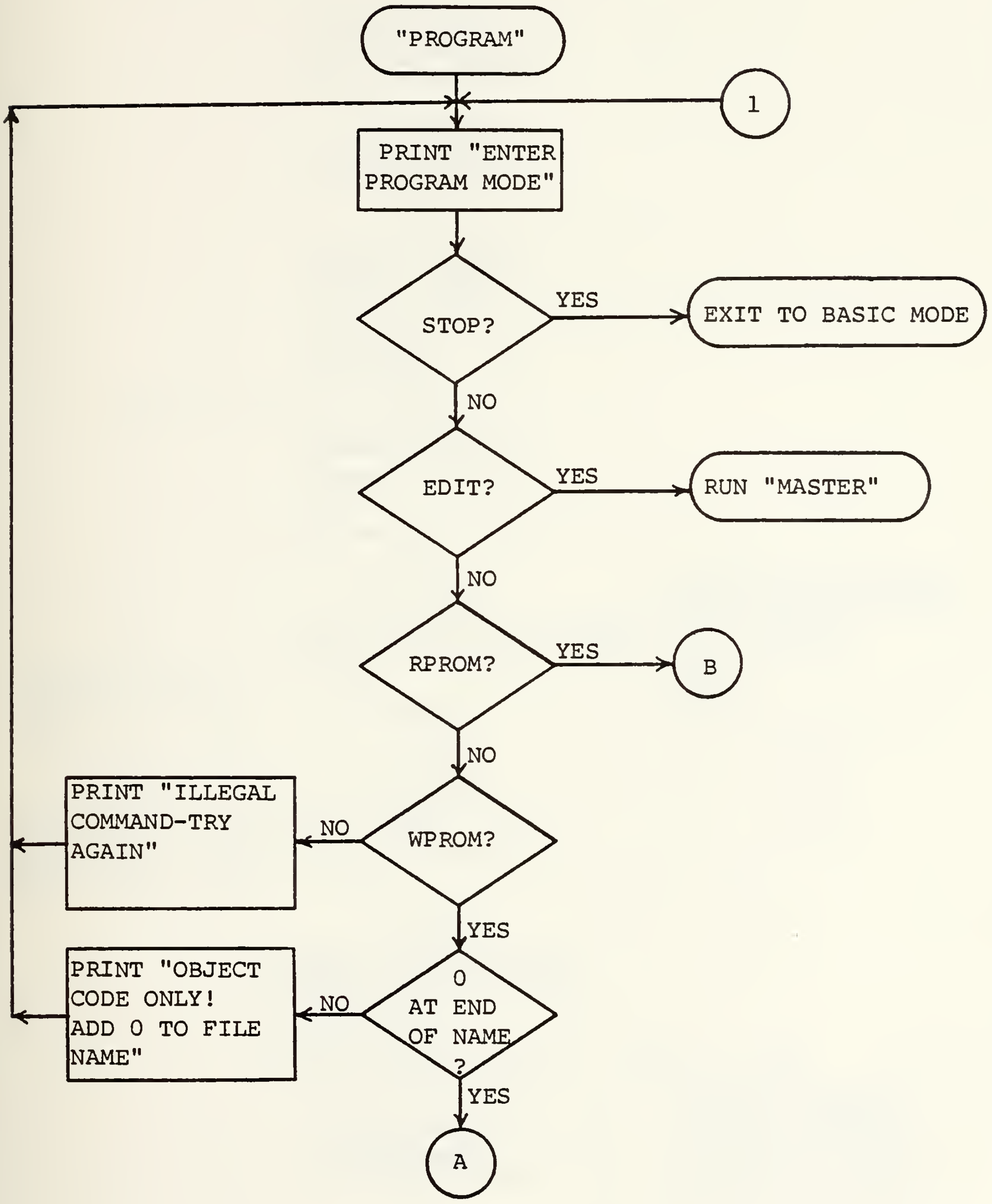

A
OPCODE CONTAIN J?
NO
OPCODE =JMPP?
YES
NO
OPCODE CONTAIN #?
NO
ADD 1 TO PREVIOUS ADDRESS
OPCODE CONTAIN CALL?
NO
ADD 1 to PREVIOUS ADDRESS
ADD 2 TO PREVIOUS ADDRESS
CONVERT DECIMAL ADDRESS TO HEX
PRINT LINE NUMBER HEX ADDRESS, LABEL OPCODE, OPERAND AND COMMENTS
LOOK AT NEXT LINE
NO
LAST LINE?
YES
PRINT "FIRST PASS COMPLETED"
B

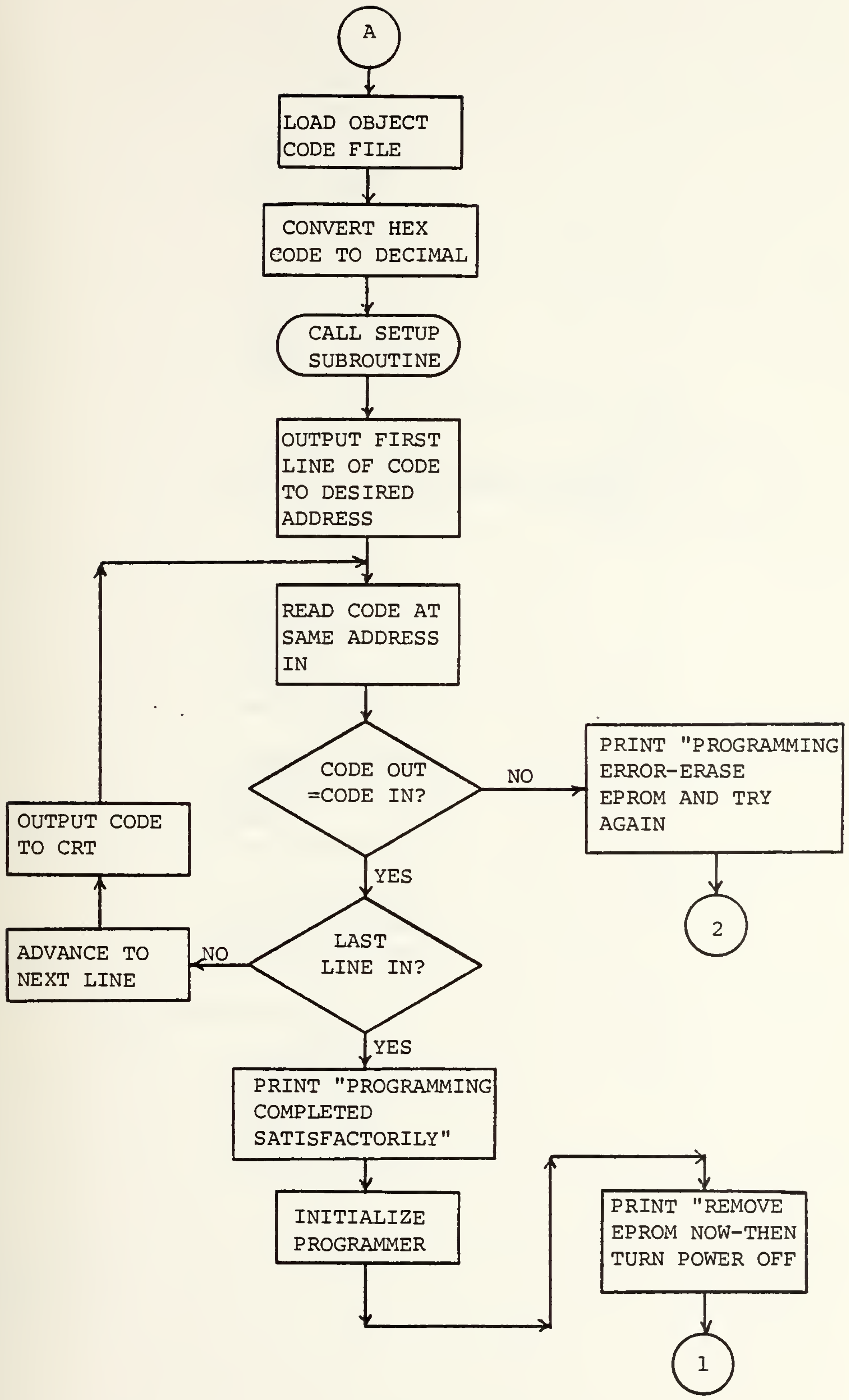

B
GET FIRST LINE
DELETE ALL COMMENTS
ASSIGN HEX CODE TO OPCODE AND OPERAND
ERROR DETECTED?
YES
NO
INCREMENT ERROR COUNT
STORE ERROR AT LINE NUMBER
PRINT HEX CODE
LAST LINE?
NO
GET NEXT LINE
PRINT "ASSEMBLY COMPLETED"
PRINT NUMBER OF ERRORS
C

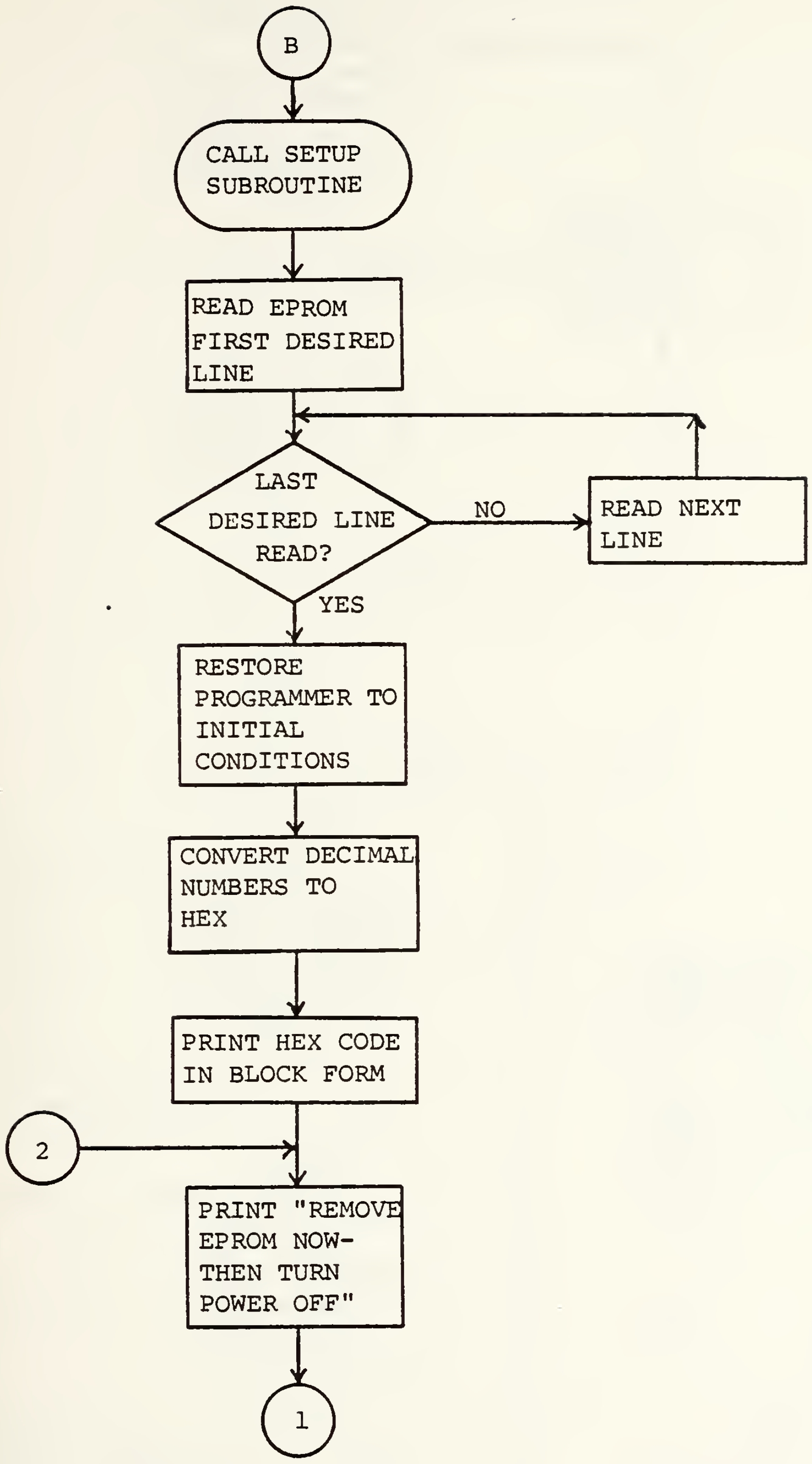

C
ERRORS=0?
NO
PRINT HEADING "LINE ERROR"
YES
PRINT ASSEMBLED PROGRAM AT LINE PRINTER
RETURN TO LINE 1
ERROR IN THIS LINE?
NO
YES
PRINT LINE NUMBER AND ERROR
NO
ERRORS=0?
YES
RUN "MASTER"
LAST LINE?
NO
GET NEXT LINE
STORE OBJECT CODE ON DISK UNDER FILENAME + 0
RUN "PROGRAM"

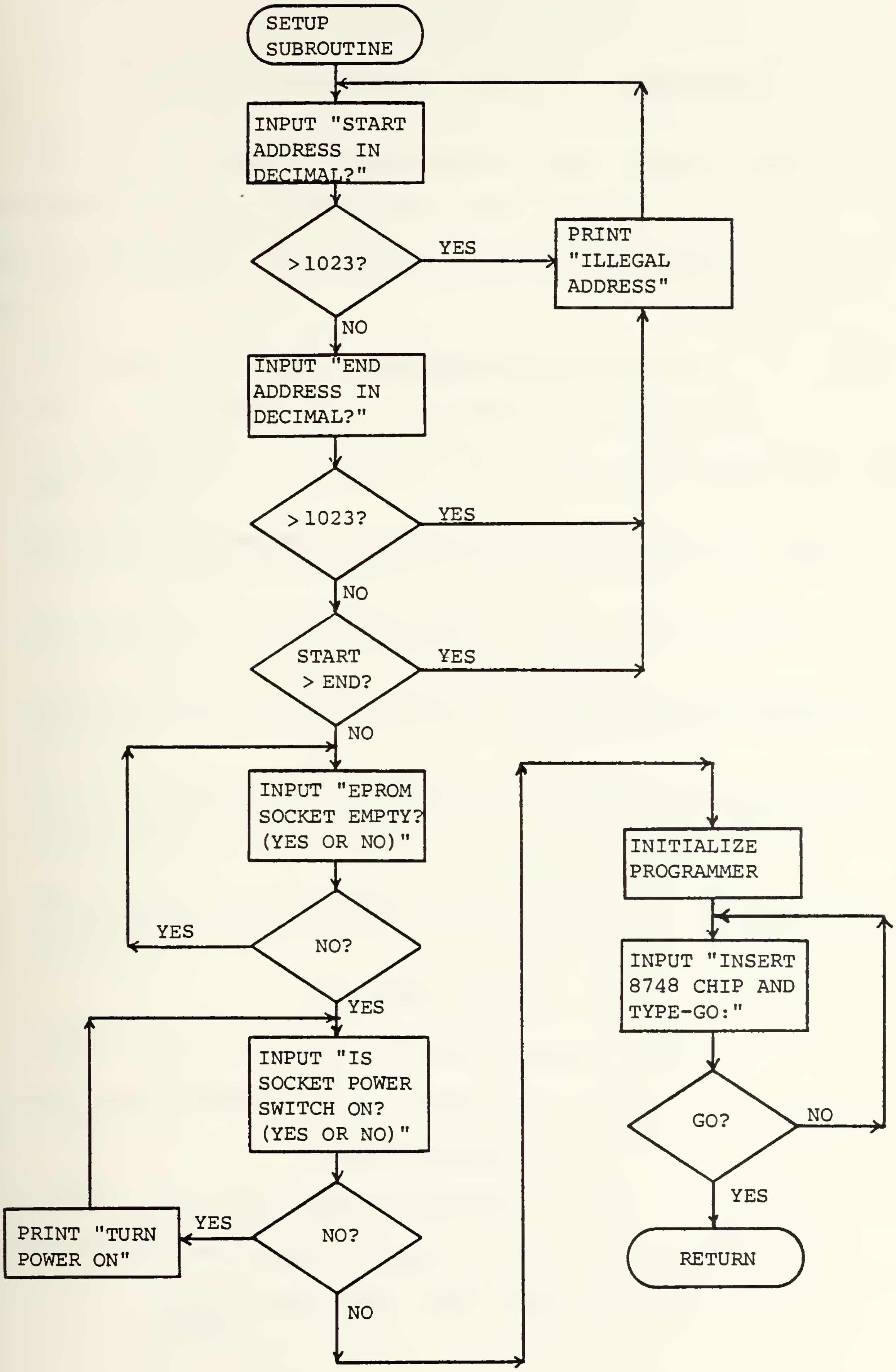
APPENDIX D

FLOWCHART FOR "PROGRAM" 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 "PROGRAMMING ERROR-ERASE EPROM AND TRY AGAIN
2
YES
LAST LINE IN?
NO
ADVANCE TO NEXT LINE
OUTPUT CODE TO CRT
YES
PRINT "PROGRAMMING COMPLETED SATISFACTORILY"
INITIALIZE PROGRAMMER
PRINT "REMOVE EPROM NOW-THEN TURN POWER OFF
1


B
CALL SETUP SUBROUTINE
READ EPROM FIRST DESIRED LINE
LAST DESIRED LINE READ?
NO
READ NEXT LINE
YES
RESTORE PROGRAMMER TO INITIAL CONDITIONS
CONVERT DECIMAL NUMBERS TO HEX
PRINT HEX CODE IN BLOCK FORM
2
PRINT "REMOVE EPROM NOW-THEN TURN POWER OFF"
1


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?
NO
INITIALIZE PROGRAMMER
INPUT "INSERT 8748 CHIP AND TYPE-GO:"
GO?
NO
YES
RETURN

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 necessary.

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 followed 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.

<u>Modes available</u>:

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.)

<u>Edit Commands</u>:

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/yyy/	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.

<u>Input operations</u>:

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	Alphanumerics 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 executive 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. Unfortunately 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 FFØØ - FF5F
2 '    DCB: +Ø => DCB TYPE  PRINTER STARTS @ 4Ø25
                1 => DRIVER ADDR LSB
                2 => DRIVER ADDR MSB
                3 => LINES/PAGE
                4 => LINE COUNTER
3 POKE 16421,2:POKE 16422,Ø:POKE 16423,255
4 FOR I = Ø TO 95
5 X=-256 +I
6 READ Y
7 POKE X,Y
8 NEXT I
9 RUN"MASTER"
1Ø END
11 DATA 229,197,245,58,72,255,254,1,4Ø
12 DATA 32,62,1,5Ø,72,255,211,232,219,233
13 DATA 23Ø,248,246,4,5Ø,71,255,211,234
14 DATA 219,233,23Ø,7,33,63,255,6,Ø,79,9
15 DATA 126,211,233,241,193,225,219,234
16 DATA 2Ø3,119,4Ø,25Ø,121,211,235,254
17 DATA 13,32,4,195,8Ø,255,ØØ,2Ø1,34,68
18 DATA 85,1Ø2,119,17Ø,2Ø4,238,Ø,Ø
19 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,229,33,255,68,43
2Ø DATA 124,181,194,84,255,225,14,1Ø,195,45,255
```

APPENDIX G

Editor Program - "MASTER"

```
1 '****"MASTER"****
2 ' ---EXECUTIVE ROUTINE----
3 CLEAR 10000
4 DEFINT E-Q
5 DEFSTR 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="GO"
12 LINE INPUT "ENTER MODE SELECTION--";Y
13 IF Y="LPT1" GOTO 67
14 IF Y="INPUT" GOTO 53
15 IF Y="EDIT" GOTO 27
16 IF LEFT$(Y,3)="GET" GOTO 83
17 IF Y<>"PRINT" GOTO 19
18 GOSUB 119  :GOTO 11
19 IF LEFT$(Y,3)<>"ASM" GOTO 24
20 Z=MID$(Y,5,10)
21 OPEN"O",1,"STORE"
22 PRINT#1,Z: CLOSE
23 RUN"ASSEMBLE"
24 PRINT "ILLEGAL ENTRY-TRY AGAIN"
25 GOTO 11
26 ' ---EDIT ROUTINE---
27 PRINT "EDIT"
28 M=I
29 PRINT X(M)
30 LINE INPUT "";Z
31 IF Z<>"UP" GOTO 34
32 IF M=0 GOTO 29
33 M=M-1: GOTO 29
34 IF Z<>"DN" GOTO 36
35 M=M+1: GOTO 29
36 IF Z="DEL" GOTO 113
37 IF Z="INS" GOTO 104
38 IF LEFT$(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
```

```
47 M=Ø: GOTO 29
48 IF Z<>"EOF" GOTO 5Ø
49 M=J: GOTO 29
5Ø PRINT "ILLEGAL ENTRY-TRY AGAIN"
51 GOTO 29
52 ' ----INPUT ROUTINE----
53 PRINT "INPUT"
54 FOR I=J TO 1ØØØ
55 LINE INPUT "";X(I)
56 IF X(I)="QUIT" GOTO 1Ø
57 IF X(I)<>"EDIT" GOTO 59
58 X(I)="EOF" : GOTO 27
59 IF LEFT$(X(I),4)<>"FILE" GOTO 62
6Ø Y=MID$(X(I),6,1Ø): J=I
61 X(I)="EOF":GOTO 76
62 J=I
63 NEXT I
64 PRINT "YOU HAVE USED ALL 8748 MEMORY"
65 END
66 '-----LPT1 ROUTINE-----
67 FOR O=Ø TO J
68 LPRINT X(O)
69 NEXT O
7Ø GOTO 11
71 '-----FILE ROUTINE-----
72 Y=MID$(Z,6,1Ø)
73 IF LEN(Y)<>Ø GOTO 76
74 PRINT "FILENAME REQUIRED!"
75 GOTO 27
76 OPEN"O",1,Y
77 PRINT#1,J
78 FOR K=Ø TO J
79 PRINT#1,X(K)
8Ø NEXT K: CLOSE
81 GOTO 11
82 '-----GET ROUTINE-----
83 Z=MID$(Y,5,1Ø)
84 OPEN"I",1,Z
85 INPUT#1,J
86 FOR K=Ø TO J
87 LINE INPUT#1,X(K)
88 NEXT K: CLOSE
89 GOTO 11
9Ø '----LOCATE ROUTINE----
91 FOR K=4 TO 16
92 IF MID$(Z,K,1)<>"/" GOTO 1ØØ
93 N=K-4
94 XL=MID$(Z,4,N)
95 FOR M=Ø TO J
96 L=INSTR(X(M),XL)
97 IF L<>Ø GOTO 29
98 NEXT M
99 PRINT"STRING NOT LOCATED": GOTO 29
1ØØ NEXT K
```

```
101 PRINT"RIGHT HAND DELINEATOR NOT FOUND"
102 GOTO 29
103 ' ----INSERT ROUTINE----
104 FOR L=M TO J+1
105 W(M)="Ø"
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=Ø TO J
120 PRINT X(K)
121 NEXT K
122 RETURN
123 END
124 ' ----TOF ROUTINE----
125 M=Ø
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$(Z,O,1)<>"/" GOTO 145
135 Q=O-P
136 L=INSTR(X(M),XO)
137 IF L=Ø GOTO 151
138 H2=L+N
139 L=L-1
140 X1=MID$(X(M),1,L)
141 X2=MID$(Z,P,Q)
142 X3=MID$(X(M),H2,5Ø)
143 X(M)=X1+X2+X3
144 GOTO 29
145 NEXT O
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 GCTO 29
```

APPENDIX H

Assembler Program - "ASSEMBLE"

```
1 '****"ASSEMBLE"****
2 '----ASSEMBLY ROUTINE----
3 CLEAR 10000
4 DEFINT H-P
5 DEFSTR A,U,V,W,X,Y,Z,T
6 DIM D(100),HX$(100),X(100)
7 DIM U(100),V(100),Y(100),Z(100)
8 OPEN"I",1,"STORE"
9 INPUT#1,Y : CLOSE
10 OPEN"I",1,Y
11 INPUT#1,J
12 FOR I=0 TO J-1
13 LINE INPUT#1,X(I)
14 IF INSTR(X(I),":")=0 GOTO 17
15 B=INSTR(X(I),":")
16 V(I)=MID$(X(I),1,(B-1))
17 NEXT I:CLOSE
18 IF INSTR(8,X(0),"ORIG")=0 GOTO 20
19 W(0)=MID$(X(0),17,4):D(0)=VAL(W(0))-1
20 FOR K=0 TO J-1
21 T=X(K):IF INSTR(T,";")=0 GOTO 23
22 H=INSTR(T,";"):T=MID$(T,1,H)
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 D(K+1)=D(K)+BK : E=D(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 COMPLETED"
33 FOR L=0 TO J-1
34 T=X(L):IF INSTR(X(L),";")=0 GOTO 36
35 H=INSTR(X(L),";"):T=MID$(X(L),1,H)
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,"ORL")<>0 GOTO 378
40 IF INSTR(8,T,"OUTL")<>0 GOTO 405
41 IF INSTR(8,T,"ADD ")<>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(8,T,"JC ")<>0 GOTO 279
48 IF INSTR(8,T,"JF")<>0 GOTO 282
49 IF INSTR(8,T,"JMP")<>0 GOTO 289
50 IF INSTR(8,T,"JN")<>0 GOTO 296
51 IF INSTR(8,T,"CPL")<>0 GOTO 210
52 IF INSTR(8,T,"MOVD")<>0 GOTO 357
53 IF INSTR(8,T,"MOVP")<>0 GOTO 365
54 IF INSTR(8,T,"MOVX")<>0 GOTO 370
55 IF INSTR(8,T,"JB")<>0 GOTO 271
56 IF INSTR(8,T,"ADDC")<>0 GOTO 118
57 IF INSTR(8,T,"JT")<>0 GOTO 307
58 IF INSTR(8,T,"JZ ")<>0 GOTO 314
59 IF INSTR(8,T,"NOP")=0 GOTO 61
60 Z(L)="00":GOTO 140
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 68
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 COMPLETED"
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 BK=2 : GOTO 27
93 I1=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 F(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 B=INSTR(8,T,",")
121 IF B=0 GOTO 125
122 IF INSTR(B,T,",R")<>0 GOTO 133
123 IF INSTR(B,T,",@")<>0 GOTO 143
124 IF INSTR(B,T,",#")<>0 GOTO 126
125 ERS=ERS+1 : U(L)="SYNTAX ERROR" : GOTO 80
126 W=MID$(T,(B+2),3)
127 E=VAL(W)
128 IF E<256 GOTO 131
129 ERS=ERS+1
130 U(L)="DATA EXCEEDS BYTE SIZE":GOTO 80
131 IF Q=0 THEN Y(L)="03" ELSE Y(L)="19"
132 GOTO 138
133 W=MID$(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 END
143 W=MID$(T,(B+3),1)
144 IF VAL(W)<2 GOTO 147
145 ERS=ERS+1
146 U(L)="R EXCEEDS 1":GOTO 80
147 IF Q=0 THEN E=96+VAL(W) ELSE E=112+VAL(W)
148 GOTO 138  : END
149 IF INSTR(8,T,"A,")=0 GOTO 163
150 B=INSTR(8,T,",")
151 IF INSTR(B,T,",R")=0 GOTO 155
152 W=MID$(T,(B+2),1) : E=VAL(W)
153 IF E>7 GOTO 135
154 E=88+E : GOTO 138
```

```
155 IF INSTR(B,T,",@")=Ø GOTO 159
156 W=MID$(T,(B+3),1)
157 IF VAL(W)>1 GOTO 145
158 E=8Ø+VAL(W) : GOTO 138
159 IF INSTR(B,T,",#")=Ø GOTO 164
16Ø W=MID$(T,(B+2),3) : E=VAL(W)
161 IF E>255 GOTO 129
162 Y(L)="53" : GOTO 138
163 B=INSTR(8,T,",")
164 IF INSTR(8,T,"BUS,")=Ø GOTO 168
165 W=MID$(T,(B+2),3) : E=VAL(W)
166 IF E>255 GOTO 129
167 Y(L)="98" : GOTO 138
168 IF INSTR(8,T,"P")=Ø GOTO 125
169 IF INSTR(8,T,"#")=Ø GOTO 178
17Ø W=MID$(T,(B-1),1)
171 IF VAL(W)<3 AND VAL(W)>Ø GOTO 174
172 ERS=ERS+1
173 U(L)="INCORRECT PORT #":GOTO 8Ø
174 E=152+VAL(W) : GOSUB 93
175 Y(L)=A(2)+A(1)
176 W=MID$(T,(B+2),3) : E=VAL(W)
177 IF E>255 GOTO 129  ELSE GOTO 138
178 IF INSTR(8,T,"ANLD")=Ø OR INSTR(8,T,",A")=Ø GOTO 125
179 W=MID$(T,(B-1),1) : E=VAL(W)-4
18Ø IF E<Ø OR E>3 GOTO 172
181 E=156+E : GOTO 138
182 '--"CALL"--
183 GOSUB 189
184 C=VAL(A(3))*2+1
185 Y(L)=STR$(C)+"4"
186 Y(L)=RIGHT$(Y(L),2)
187 GOTO 14Ø
188 '--ADDRESS SUBROUTINE--
189 IF INSTR(18,T,";")=Ø GOTO 191
19Ø B=INSTR(18,T," "):W=MID$(T,17,(B-17)):GOTO 192
191 W=MID$(T,17,8)
192 FOR M=Ø 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 8Ø
197 E=D(M) : GOSUB 93
198 Z(L)=A(2)+A(1)
199 RETURN
2ØØ '--"CLR"--
2Ø1 IF INSTR(13,T,"A")=Ø GOTO 2Ø3
2Ø2 Z(L)="27" : GOTO 14Ø
2Ø3 IF INSTR(13,T,"C")=Ø GOTO 2Ø5
2Ø4 Z(L)="57" : GOTO 14Ø
2Ø5 IF INSTR(13,T,"F1")=Ø GOTO 2Ø7
2Ø6 Z(L)="A5" : GOTO 14Ø
2Ø7 IF INSTR(13,T,"FØ")=Ø GOTO 125
2Ø8 Z(L)="85" : GOTO 14Ø
```

```
209 '--"CPL"--
210 IF INSTR(13,T,"A")=Ø GOTO 212
211 Z(L)="37" : GOTO 14Ø
212 IF INSTR(13,T,"C")=Ø GOTO 214
213 Z(L)="A7" : GOTO 14Ø
214 IF INSTR(13,T,"FØ")=Ø GOTO 216
215 Z(L)="95" : GOTO 14Ø
216 IF INSTR(13,T,"F1")=Ø GOTO 125
217 Z(L)="B5" : GOTO 14Ø
218 '--DEC--
219 IF INSTR(13,T," A")=Ø GOTO 221
22Ø Z(L)="Ø7" : GOTO 14Ø
221 IF INSTR(13,T," R")=Ø GOTO 125
222 B=INSTR(13,T,"R")
223 W=MID$(T,(B+1),1) : E=VAL(W)
224 IF E>7 GOTO 135
225 E=E+2ØØ : GOTO 138
226 '--DIS--
227 IF INSTR(13,T," I")=Ø GOTO 229
228 Z(L)="15" : GOTO 14Ø
229 IF INSTR(13,T,"TCNTI")=Ø GOTO 125
23Ø Z(L)="35" : GOTO 14Ø
231 '--DJNZ--
232 B=INSTR(13,T,",")
233 W=MID$(T,(B-1),1) : E=VAL(W)
234 IF E>7 GOTO 135
235 E=E+232 : GOSUB 93
236 Y(L)=A(2)+A(1)
237 W=MID$(T,(B+1),8)
238 FOR M=Ø TO J-1
239 IF W=V(M) GOTO 242
24Ø NEXT M
241 GOTO 195
242 E=D(M):GOSUB 93:Z(L)=A(2)+A(1):GOTO 14Ø
243 '--EN--
244 IF INSTR(13,T,"TCNTI")=Ø GOTO 246
245 Z(L)="25" : GOTO 14Ø
246 IF INSTR(13,T," I")=Ø GOTO 248
247 Z(L)="Ø5" : GOTO 14Ø
248 IF INSTR(13,T,"CLK")=Ø GOTO 25Ø
249 Z(L)="75" : GOTO 14Ø
25Ø IF INSTR(8,T,"END")=Ø GOTO 125
251 GOTO 81
252 '--IN--
253 IF INSTR(13,T,"A,P")=Ø GOTO 125
254 B=INSTR(13,T,"P")
255 W=MID$(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")=Ø GOTO 261
26Ø Z(L)="17" : GOTO 14Ø
261 B=INSTR(13,T,"R")
262 IF INSTR(13,T," R")=Ø GOTO 266
```

```
263 W=MID$(T,(B+1),1) : E=VAL(W)
264 IF E>7 GOTO 135
265 E=E+24 : GOTO 138
266 IF INSTR(13,T,"@R")=0 GOTO 125
267 W=MID$(T,(B+1),1)
268 E=VAL(W) : IF E>1 GOTO 145
269 E=E+16 : GOTO 138
270 '--JB--
271 B=INSTR(8,T,"B")
272 W=MID$(T,(B+1),1) : E=VAL(W)
273 IF E<8 GOTO 275
274 ERS=ERS+1 : U(L)="BIT > 7"
275 E=18+(E*32) : GOSUB 93
276 Y(L)=A(2)+A(1)
277 GOSUB 189  : GOTO 140
278 '--JC--
279 Y(L)="F6"
280 GOSUB 189:GOTO 140
281 '--JF--
282 IF INSTR(8,T,"JF0 ")=0 GOTO 285
283 Y(L)="B6"
284 GOSUB 189:GOTO 140
285 IF INSTR(8,T,"JF1 ")=0 GOTO 125
286 Y(L)="76"
287 GOSUB 189:GOTO 140
288 '--JMP--
289 IF INSTR(8,T,"JMPP")=0 GOTO 291
290 Z(L)="B3":GOTO 140
291 IF INSTR(8,T,"JMP ")=0 GOTO 125
292 GOSUB 189
293 E=VAL(A(3))*2:GOSUB 93
294 Y(L)=A(1)+"4":GOTO 140
295 '--JN--
296 IF INSTR(8,T,"JNC ")=0 GOTO 298
297 Y(L)="E6":GOSUB 189:GOTO 140
298 IF INSTR(8,T,"JNI ")=0 GOTO 300
299 Y(L)="86":GOSUB 189:GOTO 140
300 IF INSTR(8,T,"JNT0")=0 GOTO 302
301 Y(L)="26":GOSUB 189:GOTO 140
302 IF INSTR(8,T,"JNT1")=0 GOTO 304
303 Y(L)="46":GOSUB 189:GOTO 140
304 IF INSTR(8,T,"JNZ ")=0 GOTO 125
305 Y(L)="96":GOSUB 189:GOTO 140
306 '--JT--
307 IF INSTR(8,T,"JTF ")=0 GOTO 309
308 Y(L)="16":GOSUB 189:GOTO 140
309 IF INSTR(8,T,"JT0 ")=0 GOTO 311
310 Y(L)="36":GOSUB 189:GOTO 140
311 IF INSTR(8,T,"JT1 ")=0 GOTO 125
312 Y(L)="56":GOSUB 189:GOTO 140
313 '--JZ--
314 Y(L)="C6":GOSUB 189:GOTO 140
315 '--MOV--
316 IF INSTR(13,T,"R")=0 GOTO 342
```

```
317 B=INSTR(13,T,"R")
318 W=MID$(T,(B+1),1):E=VAL(W)
319 IF INSTR(13,T,"A,R")=Ø GOTO 322
32Ø IF E>7 GOTO 135
321 E=E+248:GOTO 138
322 IF INSTR(13,T,"A,@R")=Ø GOTO 325
323 IF E>1 GOTO 146
324 E=E+24Ø:GOTO 138
325 IF INSTR(13,T,",A")<>Ø AND INSTR(13,T,"@R")<>Ø GOTO 329
326 IF INSTR(13,T,",A")=Ø GOTO 331
327 IF E>7 GOTO 135
328 E=E+168:GOTO 138
329 IF E>1 GOTO 146
33Ø E=E+16Ø:GOTO 138
331 IF INSTR(13,T,",#")=Ø GOTO 125
332 IF INSTR(13,T,"@R")=Ø GOTO 34Ø
333 IF E>1 GOTO 146
334 E=E+176
335 GOSUB 93
336 Y(L)=A(2)+A(1)
337 W=MID$(T,(B+4),3):E=VAL(W)
338 IF E>255 GOTO 129
339 GOTO 138
34Ø IF E>7 GOTO 135
341 E=E+184:GOTO 335
342 IF INSTR(13,T,"A,#")=Ø GOTO 348
343 B=INSTR(13,T,",")
344 Y(L)="23"
345 W=MID$(T,(B+2),3) : E=VAL(W)
346 IF E>255 GOTO 129
347 GOTO 138
348 IF INSTR(13,T,"A,PSW")=Ø GOTO 35Ø
349 Z(L)="C7":GOTO 14Ø
35Ø IF INSTR(13,T,"A,T")=Ø GOTO 352
351 Z(L)="42":GOTO 14Ø
352 IF INSTR(13,T,"PSW,A")=Ø GOTO 354
353 Z(L)="D7"
354 IF INSTR(13,T,"T,A")=Ø GOTO 125
355 Z(L)="62":GOTO 14Ø
356 '--MOVD--
357 B=INSTR(13,T,"P")
358 W=MID$(T,(B+1),1):E=VAL(W)-4
359 IF E<Ø OR E>3 GOTO 172
36Ø IF INSTR(13,T,"A,P")=Ø GOTO 362
361 E=12+E:GOTO 138
362 IF INSTR(13,T,",A")=Ø GOTO 125
363 E=E+6Ø:GOTO 138
364 '--MOVP--
365 IF INSTR(8,T,"MOVP ")=Ø GOTO 367
366 Z(L)="A3":GOTO 14Ø
367 IF INSTR(8,T,"MOVP3")=Ø GOTO 125
368 Z(L)="E3":GOTO 14Ø
369 '--MOVX--
37Ø B=INSTR(13,T,"R")
```

```
371 W=MID$(T,(B+1),1):E=VAL(W)
372 IF E>1 GOTO 145
373 IF INSTR(13,T,"A,@R")=0 GOTO 375
374 E=128+E:GOTO 138
375 IF INSTR(13,T,",A")=0 GOTO 125
376 E=144+E:GOTO 138
377 '--ORL--
378 B=INSTR(13,T,",")
379 IF INSTR(13,T,",#")=0 GOTO 392
380 W=MID$(T,(B+2),3):E=VAL(W)
381 IF E>255 GOTO 129
382 GOSUB 93
383 Z(L)=A(2)+A(1)
384 IF INSTR(13,T,"A,")=0 GOTO 386
385 Y(L)="43":GOTO 140
386 IF INSTR(13,T,"BUS,")=0 GOTO 388
387 Y(L)="88":GOTO 140
388 IF INSTR(13,T," P")=0 GOTO 125
389 W=MID$(T,(B-1),1):E=VAL(W)
390 IF E<1 OR E>2 GOTO 172
391 E=E+136:GOSUB 93:Y(L)=A(2)+A(1):GOTO 140
392 IF INSTR(8,T,"ORLD ")=0 GOTO 396
393 W=MID$(T,(B-1),1):E=VAL(W)-4
394 IF E<0 OR E>3 GOTO 172
395 E=E+140:GOSUB 93:Y(L)=A(2)+A(1):GOTO 140
396 IF INSTR(13,T,"A,R")=0 GOTO 400
397 W=MID$(T,(B+2),1):E=VAL(W)
398 IF E>7 GOTO 135
399 E=E+72 : GOTO 138
400 IF INSTR(13,T,"A,@R")=0 GOTO 125
401 W=MID$(T,(B+3),1):E=VAL(W)
402 IF E>1 GOTO 145
403 E=E+64 : GOTO 138
404 '--OUTL--
405 IF INSTR(13,T,"BUS,A")=0 GOTO 407
406 Z(L)="02":GOTO 140
407 IF INSTR(13,T,",A")=0 GOTO 125
408 IF INSTR(13,T,"P0,A")<>0 GOTO 413
409 B=INSTR(13,T,",")
410 W=MID$(T,(B-1),1):E=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 INSTR(8,T,"RETR")=0 GOTO 417
416 Z(L)="93":GOTO 140
417 IF INSTR(8,T,"RET")=0 GOTO 125
418 Z(L)="83":GOTO 140
419 '--RL--
420 IF INSTR(13,T," A")=0 GOTO 125
421 IF INSTR(8,T,"RLC ")=0 GOTO 423
422 Z(L)="F7":GOTO 140
423 IF INSTR(8,T,"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)="77":GOTO 140
431 '--SEL--
432 IF INSTR(12,T,"MB0")=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,"RB0")=0 GOTO 438
437 Z(L)="C5":GOTO 140
438 IF INSTR(12,T,"RB1")=0 GOTO 125
439 Z(L)="D5":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$(T,(B+1),1):E=VAL(W)
448 IF INSTR(8,T,"XCHD ")=0 OR INSTR(12,T,"A,@R")=0 GOTO 451
449 IF E>1 GOTO 145
450 E=E+48:GOTO 138
451 IF INSTR(12,T,"A,R")=0 GOTO 454
452 IF E>7 GOTO 135
453 E=E+40:GOTO 138
454 IF INSTR(12,T,"A,@R")=0 GOTO 125
455 IF E>1 GOTO 145
456 E=E+32:GOTO 138
457 '--XRL--
458 B=INSTR(12,T,"R")
459 W=MID$(T,(B+1),1):E=VAL(W)
460 IF INSTR(12,T,"A,R")=0 GCTO 463
461 IF E>7 GOTO 135
462 E=E+216:GOTO 138
463 IF INSTR(12,T,"A,@R")=0 GOTO 466
464 IF E>1 GOTO 145
465 E=E+208:GOTO 138
466 IF INSTR(12,T,"A,#")=0 GOTO 125
467 B=INSTR(12,T,"#")
468 W=MID$(T,(B+1),3):E=VAL(W)
469 IF E>255 GOTO 129
470 Y(L)="D3":GOTO 138
471 END
472 '--LINE PRINT LISTING--
473 LPRINT "":LPRINT "ASSEMBLY OF "+"'";Y;"'"+" COMPLETED"
474 LPRINT ""
475 LPRINT "LINE"+"   "+"HEX"+"    "+"CODE"+"    "+"LABEL"+"
 "+"OPCODE"+"   "+"OPERAND"+"      "+"COMMENTS"
476 FOR N=0 TO J-1
477 LPRINT N+1;TAB(6) HX$(N);TAB(12) Y(N)+Z(N);TAB(19) X(N)
```

```
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);"---"+HX$(M)
485 NEXT M
486 IF ERS<>0 GOTO 490
487 LPRINT ""
488 LPRINT TAB(30) ">>> NO ASSEMBLY ERRORS DETECTED <<<"
489 GOTO 492
490 LPRINT "":LPRINT ERS;" ERRORS DETECTED"
491 RUN"MASTER"
492 Y=Y+"O"
493 OPEN "O",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 1ØØØØ
4 DEFINT A,I,J,K,M,N,O,Q,P
5 DEFSTR W,X,Y,Z
6 DIM D(1Ø24),T(1Ø24),X(1Ø24)
7 C=Ø
8 LINE INPUT "ENTER PROGRAM MODE--";Y
9 IF Y="STOP" END
1Ø IF Y="EDIT" RUN"EDTASM"
11 IF Y="RPROM" GOTO 9Ø
12 IF LEFT$(Y,5)="WPROM" GOTO 16
13 PRINT "ILLEGAL COMMAND-TRY AGAIN"
14 GOTO 8
15 '---WPROM ROUTINE---
16 Z=MID$(Y,7,11)
17 IF RIGHT$(Z,1)="O" GOTO 2Ø
18 PRINT "OBJECT CODE ONLY! ADD O TO FILE NAME"
19 GOTO 8
2Ø OPEN "I",1,Z
21 FOR I=Ø TO 1Ø24
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 4,2Ø
3Ø FOR M=1 TO Q
31 OUT 2,N1
32 OUT 4,21
33 OUT 1,I1
34 OUT 4,81
35 OUT 1,D(M)
36 OUT 4,2Ø9
37 OUT 4,221
38 GOSUB 58
39 OUT 4,2Ø9
4Ø OUT 4,81
41 OUT 4,84
42 OUT 4,116
43 T(M)=INP(3)
44 OUT 4,2Ø
45 I1=I1+1
46 IF I1<>256 OR I1<>512 OR I1<>768 GOTO 48
```

```
47 N1=N1+1
48 IF D(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 D(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,I1
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 O=1
140 PRINT IX;TAB(6) X(O)+" "+X(O+1)+" "+X(O+2)+" "+X(O+3)+"
"+X(O+4)+" "+X(O+5)+" "+X(O+6)+" "+X(O+7)+" "+X(O+8)+"
"+X(O+9)+" "+X(O+10)+" "+X(O+11)+" "+X(O+12)+" "+X(O+13)+"
"+X(O+14)+" "+X(O+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
```

```
152 PRINT "ILLEGAL ADDRESS" : GOTO 149
153 PRINT "END ADDRESS IN DECIMAL?"
154 INPUT I2 : Q=I2-I1
155 IX=I1
156 IF I1>255 GOSUB 172
157 IF Q<0 GOTO 152
158 IF I2<1024 GOTO 160
159 PRINT "ILLEGAL ADDRESS" : GOTO 153
160 LINE INPUT "EPROM SOCKET EMPTY?(YES OR NO): ";X3
161 IF X3="NO" GOTO 160
162 LINE INPUT "IS SOCKET POWER SWITCH ON?(YES OR NO): ";X4
163 IF X4="NO" GOTO 164 ELSE GOTO 165
164 PRINT "TURN POWER ON" : GOTO 162
165 OUT 4,36
166 IF LEFT$(Y,5)<>"WPROM" GOTO 169
167 PRINT: PRINT"FOLLOWING IS HEX CODE TO BE PROGRAMMMED:"
168 GOSUB 139
169 LINE INPUT "INSERT 8748 CHIP AND TYPE-GO: ";X5
170 IF X5<>"GO" GOTO 169
171 RETURN
172 IF I1>511 GOTO 174
173 N1=1: I1=I1-256: RETURN
174 IF I1>767 GOTO 176
175 N1=2: I1=I1-512: RETURN
176 N1=3: I1=I1-768
177 RETURN
```

APPENDIX J

Sample Assembler Printout

```
ASSEMBLY OF 'DOUGS' COMPLETED

LINE   HEX    CODE    LABEL    OPCODE   OPERAND     COMMENTS
 1     ØØ                      ORIG     ØØØ
 2     ØØØ    Ø4Ø5             JMP      START      ;POWER UP
 3     ØØ2    ØØ               NOP
 4     ØØ3    Ø428             JMP      INT        ;INTERRUPT
 5     ØØ5    B844    START:   MOV      RØ,#68
 6     ØØ7    B97F             MOV      R1,#127
 7     ØØ9    BAØD             MOV      R2,#13
 8     ØØB    BBØA             MOV      R3,#1Ø
 9     ØØD    ØØ               NOP
 1Ø    ØØE    ØØ               NOP
 11    ØØF    ØØ               NOP
 12    Ø1Ø    23Ø1             MOV      A,#Ø1
 13    Ø12    3A               OUTL     P2,A       ;HI-Z ON DS'
 14    Ø13    ØØ               NOP
 15    Ø14    23FF             MOV      A,#255
 16    Ø16    39               OUTL     P1,A       ;ENABLES P1
 17    Ø17    ØØ               NOP
 18    Ø18    81               MOVX     A,@R1      ; BUS TO HI-Z
 19    Ø19    ØØ               NOP
 2Ø    Ø1A    75               ENTØ     CLK        ;MAKES TØ A CLOCK
 21    Ø1B    Ø5               EN       I
 22    Ø1C    ØØ               NOP
 23    Ø1D    ØØ               NOP
 24    Ø1E    ØØ               NOP
 25    Ø1F    ØØ               NOP
 26    Ø2Ø    ØØ      LOOP:    NOP
 27    Ø21    F8               MOV      A,RØ
 28    Ø22    39               OUTL     P1,A       ;ENABLE 8212
 29    Ø23    Ø42Ø             JMP      LOOP       ;LOOP
 3Ø    Ø25    ØØ               NOP
 31    Ø26    ØØ               NOP
 32    Ø27    ØØ               NOP
 33    Ø28    2366    INT:     MOV      A,#1Ø2     ;INTERRUPT
 34    Ø2A    39               OUTL     P1,A       ;STOP TRS-8Ø
 35    Ø2B    Ø8               INS      A,BUS      ;INPUT
 36    Ø2C    59               ANL      A,R1       ;7 BITS?
 37    Ø2D    DA               XRL      A,R2       ;CHECK
 38    Ø2E    ØØ               NOP
 39    Ø2F    ØØ               NOP
 4Ø    Ø3Ø    9642             JNZ      STOP       ;END IF NO CR
 41    Ø32    ØØ      UNTIL:   NOP
 42    Ø33    8632             JNI      UNTIL      ;LOOP TIL EOC=1
 43    Ø35    ØØ               NOP
 44    Ø36    ØØ               NOP
 45    Ø37    2333             MOV      A,#51
```

```
039   39                 OUTL    P1,A      ;DISABLE 8212
03A   FB                 MOV     A,R3
03B   02                 OUTL    BUS,A     ;OA ON BUS
03C   2300               MOV     A,#00
03E   3A                 OUTL    P2,A      ;P2 LOW
03F   2311               MOV     A,#17
041   3A                 OUTL    P2,A      ;END STROBE
042   81        STOP:    MOVX    A,@R1     ;BUS TO HI-Z
043   00        BACK:    NOP
044   8643               JNI     BACK      ;LOOP TIL EOC=1
046   00        WAIT:    NOP
047   564B               JT1     READY     ;JUMP TO 04A
049   0446               JMP     WAIT      ;LOOP
04B   93        READY:   RETR              ;RETURN
                         END
```

```
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

1. Intel Corporation, MCS-48 Microcomputer User's Manual, 1978.

2. Radio Shack, TRS-80 RS-232-C Interface, TRS-80 Microcomputer System, 1978.

3. Radio Shack, Level II BASIC Reference Manual, TRS-80 Microcomputer System, 1978.

4. Radio Shack, TRSDOS and Disk BASIC Reference Manual, TRS-80 Microcomputer System, 1979.

5. Radio Shack, TRSDOS Version 2.2 and Disk BASIC Version 2.2, May 1979.

6. Intel Corporation, MCS-48 and UPI-41 Assembly Language Manual, 1978.

7. Intel Corporation, Memory Design Handbook, 1977.

8. Radio Shack, TRS-80 Microcomputer Technical Reference Handbook, 1978.

INITIAL DISTRIBUTION LIST

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

Thesis 186582
S41925 Seward
c.1 A microprocessor
development system for
the Intel 8748 micro-
computer.