THE DESIGN AND IMPLEMENTATION OF AN INTEL 8085 MICROPROCESSOR TRAINER

BY

BRIAN CHUKWUJEKWU EJIKE

20111771403

(ECE OPTION)

Submitted to:

The Department of Electrical and Electronic Engineering,

School of Engineering and Engineering Technology,

Federal University of Technology, Owerri

In partial fulfilment of the requirements for the award of the Bachelor of Engineering degree (B. Eng.) in Electrical and Electronic Engineering

November 2016

ABSTRACT

Microprocessor architecture is a course commonly taught to ECE students in universities around the globe. The subject is best taught with practical tools or trainers to aid students in understanding the internals of a microprocessor. The cost of purchasing such tools can however be prohibitive to Third World institutions. This paper details the design and subsequent construction of an effective Intel 8085 trainer to serve as proof-of-concept that electronic engineering departments in Nigerian universities can manufacture these devices independently at relatively low costs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.0 – INTRODUCTION

According to Anil Maini [1], a microprocessor is “a programmable device that accepts binary data from an input device, processes the data according to the instructions stored in the memory and provides results as output”. It executes programs stored in memory and interacts with the outside world through input and output peripherals. Thus, the main components of any microprocessor-based system, as illustrated in Fig. 1.1, are **the microprocessor**, **the memory**, and **the I/O peripherals**. The **buses** convey addresses, data and control signals between these components of the system.

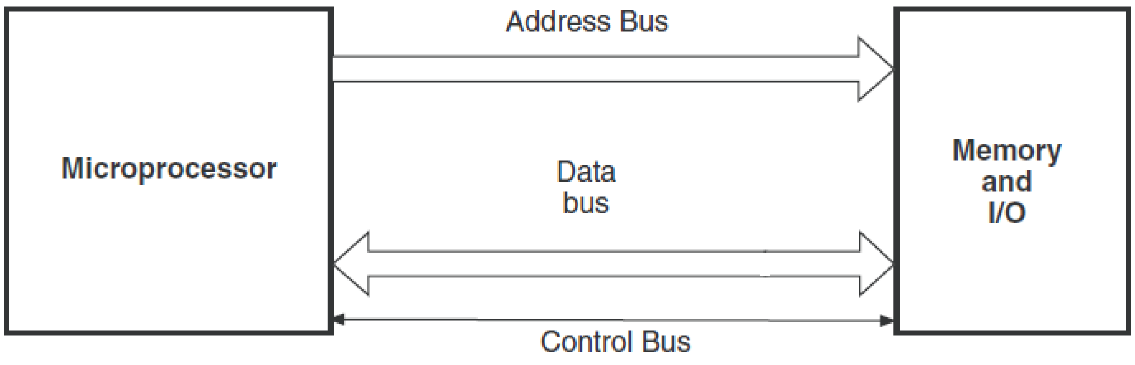


Fig. 1.1 – A typical microprocessor-based system [1]

A microprocessor typically comprises a minimal set of three functional blocks: an Arithmetic and Logic Unit (ALU), a Control Unit (CU) and registers. The ALU is a combinational logic circuit which performs all the integer arithmetic and logical operations in the system. The CU is typically a sequential logic circuit that coordinates the activities of the processor and I/O devices. It fetches instructions from memory by issuing the proper control signals, decodes the instructions and controls their sequential execution. Examples of microprocessors include the Intel 8085, Zilog Z80, and Motorola 6800.

Memory, in this context, refers to primary memory in the form of Random Access Memory (RAM) which temporarily stores the program currently being executed by the microprocessor. A program is no more than a sequence of instructions, each from the *instruction set* of the microprocessor, intended to achieve some purpose. Each location in RAM has a unique address and can holds a word which is typically 8 bits (a byte) in size. So, a 2K x 8 SRAM is a static RAM chip that can hold 2048 words each one-byte large, yielding a total storage space of 2048 bytes. Some popular RAM chips include 6116, 6264 and 62256 SRAMs.

Input and output (I/O) peripherals provide an interface between the microprocessor and the outside world. They serve as an interface to input devices such as sensors and keyboards, as well as an interface to output devices such as LCDs and printers. Within the systems they are used, I/O peripheral chips are typically **memory-mapped**, where their *ports* may be accessed by the microprocessor like any other memory location, or **I/O mapped**, where they are accessed through special instructions and have their own address space. Some common I/O peripheral ICs include the Intel 8255 PPI and Intel 8156.

The buses in a microprocessor-based system link the various components of the system together. A bus, in this context, is no more than a group of signal lines connecting different system components and carrying some unique class of signals between these components. There are three main buses: the address bus, data bus and control bus. The address bus is unidirectional and is used by the CPU to address the memory and I/O peripherals. The data bus is bidirectional and transports the actual data between the components after they have been addressed. Both the address and data bus are parallel buses and their bus width (the number of lines on the bus) is typically a multiple of 8. The control bus may also be bidirectional and is used by the CPU to control the process of communicating with the other system components. For instance, signals such as /RD and /WR may be used by the CPU to read from or write to an SRAM chip, depending on which signal is asserted.

All the components that have been described come together to form one coherent system with the microprocessor at its heart. The most ubiquitous kind of microprocessor-based systems are microcomputers or more specifically, modern PCs and smartphones. Embedded systems, which are commonly found in modern consumer electronics, could also be said to be microprocessor-based systems since the microcontrollers and DSPs often used are little more than a combination of the aforementioned components – CPU, memory, I/O ports and buses – existing on a single chip.

A microprocessor development board is a printed circuit board which contains a microprocessor together with the minimal support logic (memory and I/O) required for a user (an engineer, student or hobbyist) to become acquainted with the on-board microprocessor and learn how to program it [2]. It may also come in form of a kit, which contains the bare PCB and the electronic components intended to be soldered onto the board. The user is expected to perform the assembling as part of the learning process. When used in the context of learning devices, these boards may also be called training development boards or simply, trainers. The design and construction of such a trainer is the primary objective of this project.

1.1 – BACKGROUND OF THE STUDY

Students studying electronic and computer engineering are expected to understand digital electronic systems. Microprocessors, usually at the core of microcontrollers and DSPs, are ubiquitous in today’s electronics which are often embedded systems. Therefore, it is important that an ECE student achieves significant understanding of the working of microprocessors and microprocessor-based systems.

Microprocessor trainers are commonly found in ECE laboratories in institutions around the world. They are also called microprocessor development boards. These devices, in this capacity, serve to educate ECE students about the workings of microprocessors. Usually, a trainer is built around a single microprocessor, such as the Intel 8085 or the Motorola Z80, and allows students to write programs in machine code that are executed by the trainer. The trainer makes available to the user the contents of the microprocessor’s internal registers and RAM for inspection. The trainer may also provide I/O devices such as keypads and LCDs which may be accessed by a user through I/O-mapped peripheral chips. In this manner, a student gains critical knowledge of machine and assembly programming as well as an understanding of the intricacies of a microprocessor’s operations.

Several microprocessor development kits have been released by various manufacturers over the years. They were originally intended to get engineers acquainted with the microprocessor for which each trainer is designed. One of the most popular boards was the KIM-1 development board which was developed by MOS Technology Inc. in the 1976 to introduce new users to the MOS 6502 microprocessor [3]. Development boards for other microprocessors include the SDK-85 and MAG-85 for the Intel 8085 microprocessor,

These development boards are invaluable in ECE laboratories but can be expensive to purchase, especially for institutions in developing countries. Federal tertiary institutions in Nigeria rely heavily on funding not only from the

Federal Government but also from donors and through grants (TETFUND, PTDF, etc.) in order to develop infrastructure. Funding may be not only in monetary form but also in form of much-needed equipment. When such funding is not forthcoming, there is a tendency for university and departmental management to leave things as they stand, apparently not seeing any alternatives. This project seeks to address this issue by proving that it is possible to design and manufacture a minimal cost-effective Intel 8085 trainer that will suffice to instruct ECE students on microprocessors. This approach of manufacturing what is needed may be extended to other areas within the department or the university as a whole.

1.2 – PROBLEM STATEMENT

The prohibitive cost of purchasing microprocessor trainers from third-party sellers, coupled with the general lack of interest of management in improving the state of laboratories, has caused a severe dearth of these equipment in laboratories. The existing trainers are few in number, ancient and function sporadically. It is not uncommon for 20 students to be assigned to a single trainer during a practical session in the lab; usually only a very few of those students actually operate the trainer with the others straining over their shoulders to understand. The result is that many students complete the course with little, if any, real understanding of microprocessor architecture.

Electricity supply is a long-standing problem in Nigeria. Since the existing trainers require mains AC supply to function, the effectiveness of the trainers is much reduced since students are only able to use them when there is mains supply from the national grid or when they can afford to buy gasoline to fuel the standby generator. Alternatives must be exploited to suit the current environment. These are problems which have prompted the initiation of this project and will be addressed accordingly.

1.3 – OBJECTIVES

The main objective of this project is to design and construct a relatively cheap and minimal Intel 8085 trainer which can be used by students of the EEE department to understand microprocessor architecture and programming. The trainer will be designed to be compatible, in terms of user interface, with the present trainers in the electronics laboratory. The specific objectives are:

- The trainer should be able to perform the fundamental task of accepting user programs in machine code, executing them and displaying the results in a modern intuitive manner
- The trainer should have at least one on-board I/O peripheral
- The trainer should be manufactured relatively cheaply, compared to the cost of purchasing existing trainers
- The trainer should consume little enough power that it can be powered from common AA batteries, as well as mains-adapted DC.

1.4 – JUSTIFICATION OF THE STUDY

A tour through a typical practical session in EEE labs will show students clustering around lab equipment in large groups of up to 20 students. The only students in any group that actually learn anything are usually those closest to the equipment and the really motivated students who struggle to observe no matter where they stand in the chaos. The majority simply wait for the practical to be over and copy what others have done. Whatever interest may have been kindled by actively participating in the practical session never comes to light and the students are only interested in learning enough theory in class to pass their exams.

This problem has a root in the severe lack of functional lab equipment and is prevalent throughout the university. As stated earlier, this issue can be traced to the cost of purchasing the equipment through the usual channels and the general apathy of management. However, some of the needed equipment can be manufactured in-house within the department at relatively low costs and with features comparable to those of devices sold in the marketplace. An example of such equipment is the Intel 8085 trainer used for EEE 401 practical sessions. If the department took on the task of designing and manufacturing these trainers, it would not only help replenish the stock of functional trainers but would also help in further educating existing technologists and students on the workings of such a device since they would be involved in the design process.

Though the Intel 8085 is an obsolete microprocessor, it was chosen for this project as the focus primarily because the current EEE 401 syllabus uses the Intel 8085 as a case study into the architecture and programming of microprocessors. Constructing a trainer based on a different microprocessor would render the trainer useless in FUTO until such a time comes when the syllabus is revised. Thus, the trainer manufactured in this project can be deployed immediately for use in the electronic lab. Also, the Intel 8085 is also a microprocessor with a relatively simple architecture, as well as a lot of documentation and tutorials all over the Web, making it very easy for students to conduct their own research beyond what they are taught during lectures.

The final issue addressed here is one of power supply. The existing trainers may only be powered through 240 V mains supply. In a country like Nigeria with very unreliable power supply, students very often have to resort to AC generators in order to have practical sessions that require 240 V AC. The gasoline used to fuel the generators for each practical session is usually purchased with money contributed by the students. Batteries, rechargeable or non-rechargeable, would be a cheaper alternative. They would also be more accessible since any student could come to the lab at any time with their own battery packs and operate the trainer, without having to worry about a generator and its obligatory fuel. This maximizes the potential of the trainers and increases the chances that students can learn from the devices.

1.5 – SCOPE OF THE STUDY

This project entails the design and construction of a minimal Intel 8085 trainer with the basic features of:

- Accepting Intel 8085 programs in machine language as input
- Running input programs
- Allowing access to the internal registers of the Intel 8085, as well as the on-board RAM
- Providing I/O-mapped I/O with at least one input device and one output device.

The 8085 hardware interrupts have not been made user-accessible. Hardware single-stepping is also not implemented, unlike in existing trainers. Programs can only be loaded into the trainer by hand; there is no serial port available to download code from a PC. The aforementioned features will not be implemented primarily due to time constraints. However, a PCB will be designed and fabricated for the trainer to give a finished look to the project.

CHAPTER 2

LITERATURE REVIEW

2.1 – REVIEW OF RELATED LITERATURE

Microprocessor development boards have been around for nearly as long as there have been microprocessors. They were initially manufactured by microprocessor manufacturers to enable users (primarily engineers) and technicians learn to use their microprocessors. For instance, each time Intel launched a new microprocessor, they simultaneously provided a System Design Kit (SDK) which helped users familiarize themselves with the microprocessor. The SDK single-board computers allowed the user to enter object code from a keyboard or upload it through a communication port, and then test run the code. The SDK boards provided a system monitor ROM to operate the keyboard and other interfaces. Kits varied in their specific features but generally offered optional memory and interface configurations, a serial terminal link, audio cassette storage, EPROM program memory.

The Intel SDK-85 in Fig. 2.1 was released in 1978 and was the first microprocessor development board for the Intel 8085.

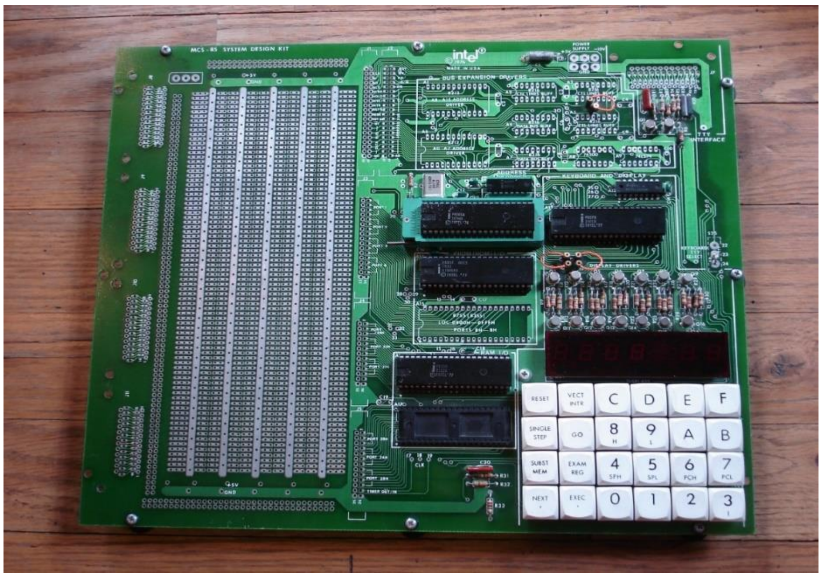


Fig. 2.1 – An assembled SDK-85

It was a complete single board computer centred on the Intel 8085. Some of its specifications [4] are as follows:

- **CPU**
    - Intel 8085 microprocessor running at 3 MHz
    - 1.3-microsecond instruction cycle
- **Memory**
    - 2K bytes of ROM containing a monitor program
    - 256 bytes of RAM
- **Interrupts:** RST 7.5, RST 6.5, INTR
- 38 parallel I/O lines and a 110-baud serial I/O port
- **Input method:** Keypad or serial I/O
- **Display:** 6 single-digit seven-segment displays

Other Intel SDKs include the SDK-51, SDK-86 and SDK-x86 series. Another popular microprocessor development board was the *KIM-1*. It was a single-board computer based on the MOS6502 and manufactured in 1976 by MOS Technologies, Inc. The *KIM-1* was very popular among hobbyists, especially due to its relatively low price and easy expandability. However, microprocessor development boards are not always produced by microprocessor manufacturers. 8085 single-board computers such as the *8085AAT* and *Explorer/85* were produced by third party companies Paccom and Netronics. It should be noted that all microprocessor development boards may also be considered single board computers.

Vintage computing enthusiasts also produce their own development boards. Fig. 2.2 shows an SBC printed circuit board designed and fabricated in 2010 by the enthusiasts at *The Glitch Works* [5]

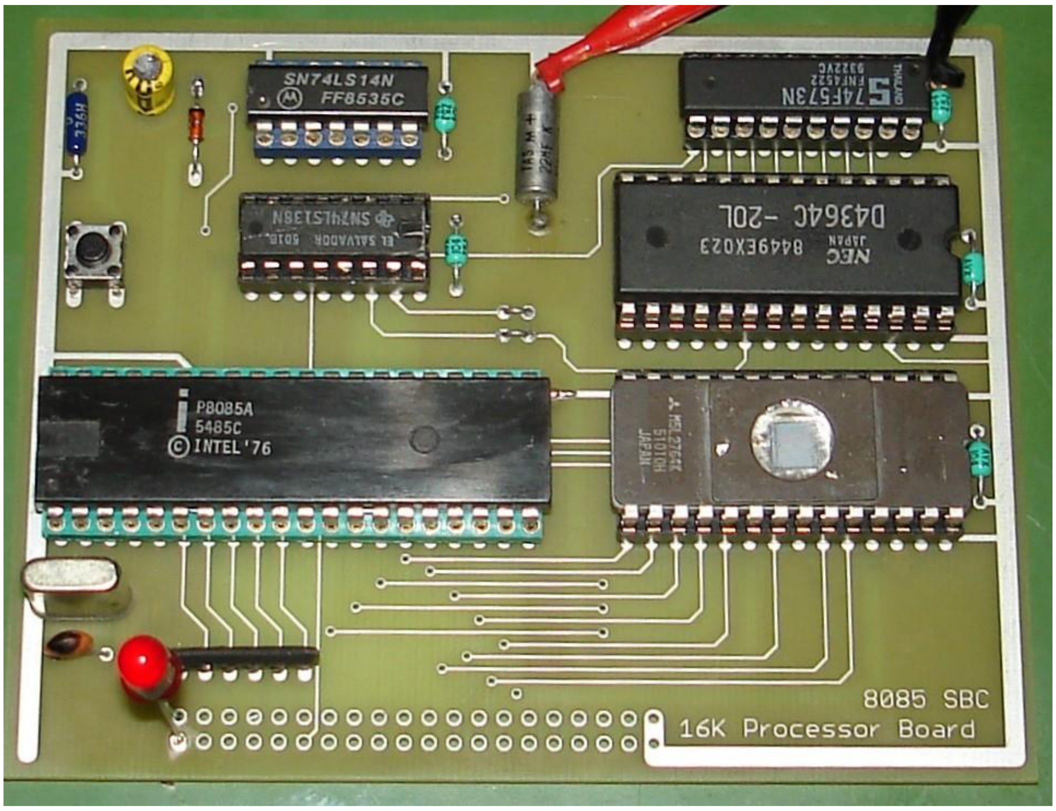


Fig. 2.2 – An 8085 single board computer

It features a 2K x 8 EEPROM for program storage and two 1K x 8 static RAMs for stack and variable space.

Fig. 2.3 is a MAG-85 [6], an Intel 8085 trainer built in 2010 by Mark Graybill, who is a retro-computing hobbyist. It features 8K of memory, a keypad for entering hexadecimal digits and commands, an LCD display as well as switches and LEDs for I/O.

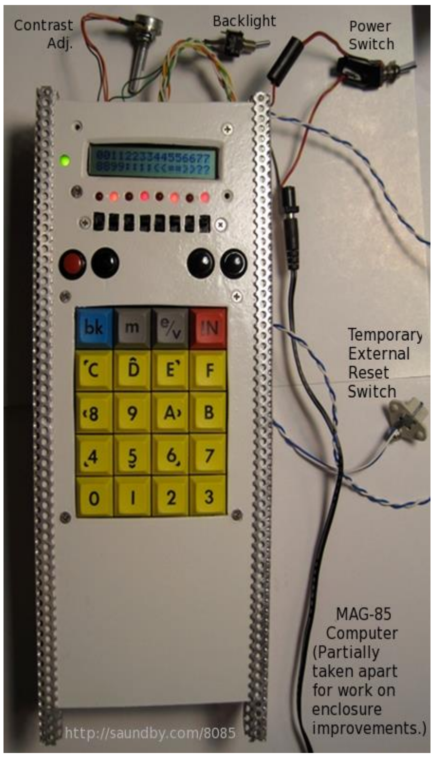

Contrast Adj.
Backlight
Power Switch
Temporary External Reset Switch
MAG-85 Computer (Partially taken apart for work on enclosure improvements.)
http://saundby.com/8085

Fig. 2.3 – The MAG-85 trainer

Intel 8085 trainers are still manufactured in bulk quantities mostly in India where the curriculum still includes the study of the Intel 8085 architecture, despite its obsolescence. Some of these trainers include the PS-85, shown in Fig. 2.4, by Pantech Solutions and the M85 series by Kitek Technologies.

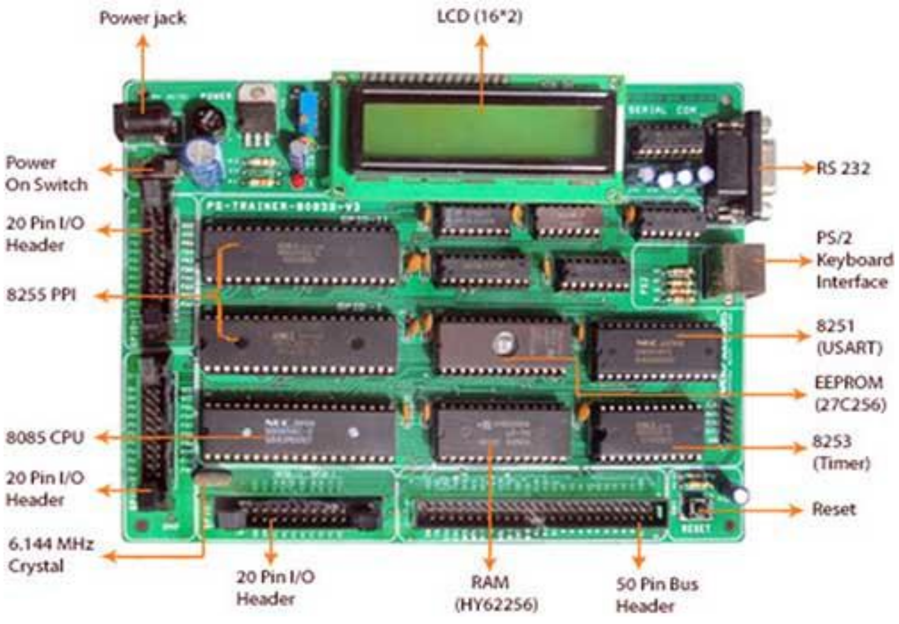


Fig. 2.4 – The PS-85 trainer

2.2 – COMPONENTS OF A MICROPROCESSOR TRAINER

Microprocessor development boards differ a lot but those intended for student use as trainers usually contain the following minimal components:

- **A Central Processing Unit (CPU)**

This is responsible for running user programs. It is usually the microprocessor being studied by the students. The subject microprocessor of this project is the Intel 8085, specifically the 8085AH-2. It is a complete 8-bit general-purpose microprocessor. The 8085AH-2 runs on a single +5V power supply and a maximum clock frequency of 5 MHz [7]. It has a multiplexed data bus; the address is split between the 8-bit upper address bus and the 8-bit data bus. It is supplied in a 40-pin DIL package with a pin-out as shown in Fig. 2.5.

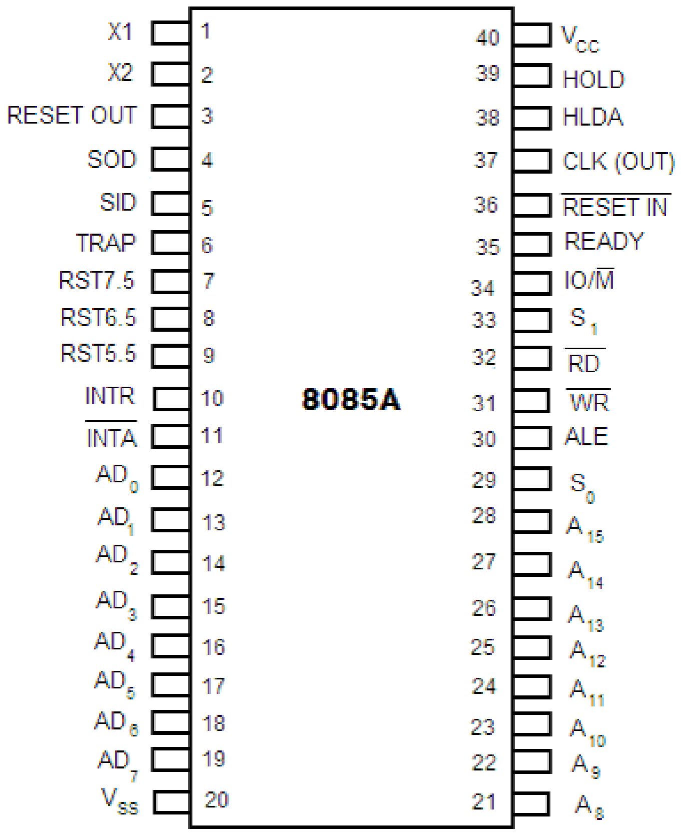
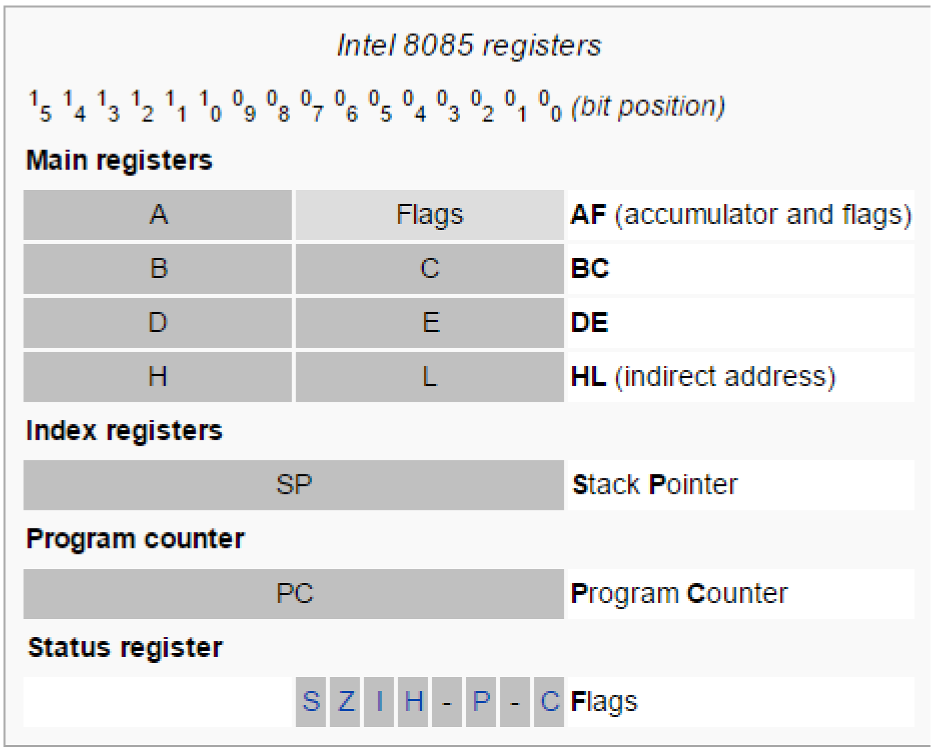


Fig. 2.5 – Intel 8085 pin-out diagram

It should be noted that the 8085AH-2 differs from the 8085AH primarily in the maximum achievable clock speed, which is 5 MHz for the former and 3 MHz for the latter. The 8085 has seven general-purpose 8-bit registers, two 16-bit registers. There are also 5 status flags – sign, zero, parity, carry, auxiliary carry – occupying an 8-bit space. The 8-bit registers may be used independently or paired to form 16-bit registers. The 16-bit registers are the Program Counter and the Stack Pointer. The Program Counter holds the address of the next instruction to be executed while the Stack Pointer holds the address of the top of the stack. Fig. 2.6 illustrates the structure of the 8085 registers.



Intel 8085 registers

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 *(bit position)*

Main registers

A	Flags	**AF** (accumulator and flags)
B	C	**BC**
D	E	**DE**
H	L	**HL** (indirect address)

Index registers

SP	**S**tack **P**ointer

Program counter

PC	**P**rogram **C**ounter

Status register

S	Z	I	H	-	P	-	C	**F**lags

Fig. 2.6 – Intel 8085 registers

The 8085 has a 16-bit address bus, making it capable of addressing up to 64K locations through the bus. The data bus is multiplexed with the lower 8 bits of the address bus. Due to its dual function as the lower address byte and a data bus, the 8085 data bus is designed to operate with an octal latch which is controlled via the ALE pin of the microprocessor. The control bus refers to a set of 10 signal lines which are used by the CPU to coordinate the operations of external devices such as memory and I/O devices [8]. Some control signals are inputs such as HOLD and READY while others are outputs such as /RD and /WR. There are also two status lines S0 and S1 which, in conjunction with the IO/M line, to determine the current machine cycle at any instant of its operation.

The 8085 has one general-purpose maskable interrupt line (INTR), three maskable RST interrupts (RST5.5, RST6.5, RST7.5) and one non-maskable TRAP interrupt [9]. 'Maskable' interrupts are those that can be disabled by the CPU whereas an interrupt is non-maskable if it cannot be disabled, that is to say, it cannot be ignored by the CPU. RST5.5, RST6.5, RST7.5 and TRAP are vectored interrupts; when any of them is triggered, the CPU pushes the program counter's contents onto the stack and branches to the address associated with the triggered interrupt.

The 8085 is a CISC processor [10] with an instruction set containing 74 instructions [11] which may be classified into data transfer operations (such as MOV, MVI, LDA), arithmetic operations (such as ADD, SUB, SBB), logic operations (such as ORA, CMP, XRA), branching instructions (such as JMP, CALL, RET) and machine control instructions (such as HLT, NOP).

- **A monitor**

The monitor is typically not a separate hardware component; it is more a logical component of a microprocessor development board. It is usually a program that oversees the system's operations. The monitor program is responsible for accepting input from the student user, processing the user's input or commands accordingly and displaying the results. All the user interactions with the development board occur through the monitor. The monitor is also crucial for debugging of user programs, typically through single-stepping.

The monitor program usually exists in non-volatile memory and is executed by the system CPU at power-up. This is not always the case though; it may also be a program running on a separate device entirely, usually a microcontroller, which observes and controls the main CPU. In such a system, the monitor could be said to be a separate hardware component and is tapped into the address, data and control buses of the main CPU in order to have control over the latter's operations.

- **Memory**

This is used for storing the monitor program persistently as well as user programs before they are executed by the CPU. The Intel 8085 has 64K address space, with portions of this space are assigned to the different memory chips in the system as well as memory-mapped I/O devices. There are two main classes of memory used in a development system:

  - **ROM:**

  ROM refers to non-volatile read-only memory. Typically, the monitor program of an SDK is stored persistently in ROM by the board manufacturer. EEPROMs are a type of ROM that can be altered electrically. EEPROM is different from ROM and PROM in that it can be not only read from, but also erased and written to, completely electrically [12]. User programs may be stored persistently in EEPROM. Parallel EEPROM is typically used with microprocessors due to the greater speeds and microprocessor-friendly interface. A common EEPROM chip used with the 8085 is the 8755 which is a 2K x 8 EEPROM.

- **RAM:**

RAM refers to volatile read/write memory. It is a class of memory that retains its contents only as long as it is powered. Static RAM consists of flip-flop circuits which do not require refreshing like dynamic RAM. Static RAM is faster than dynamic RAM but also more expensive. Because the complexity of refreshing RAM is not present in SRAM, this makes it ideal for a microprocessor development board, especially considering that SRAM is not typically requires in large capacities for a development board. SRAM is also faster than DRAM.

Typically, user programs are loaded into RAM and executed from there by the CPU. A trainer normally allocates a portion of the available RAM to users for their program storage and a general ‘scratchpad’. The monitor also reserves a portion of RAM for its own purposes in coordinating the development system. Typical RAM chips used with the 8085 include the Intel 8155/8156 which is also a timer and I/O peripheral, the 6264 8K x 8 SRAM, 6116 2K x 8 SRAM and 62256 32K x 8 SRAM. Each of these chips has an 8-bit wide data bus. Of the five chips, only the 8155 and 8156 may be connected directly to the 8085 with no intermediate latch because all the necessary glue logic is within them.

Fig. 2.7 shows a minimal 8085 system containing an 8085AH CPU, 8156H SRAM and 8755A EEPROM.

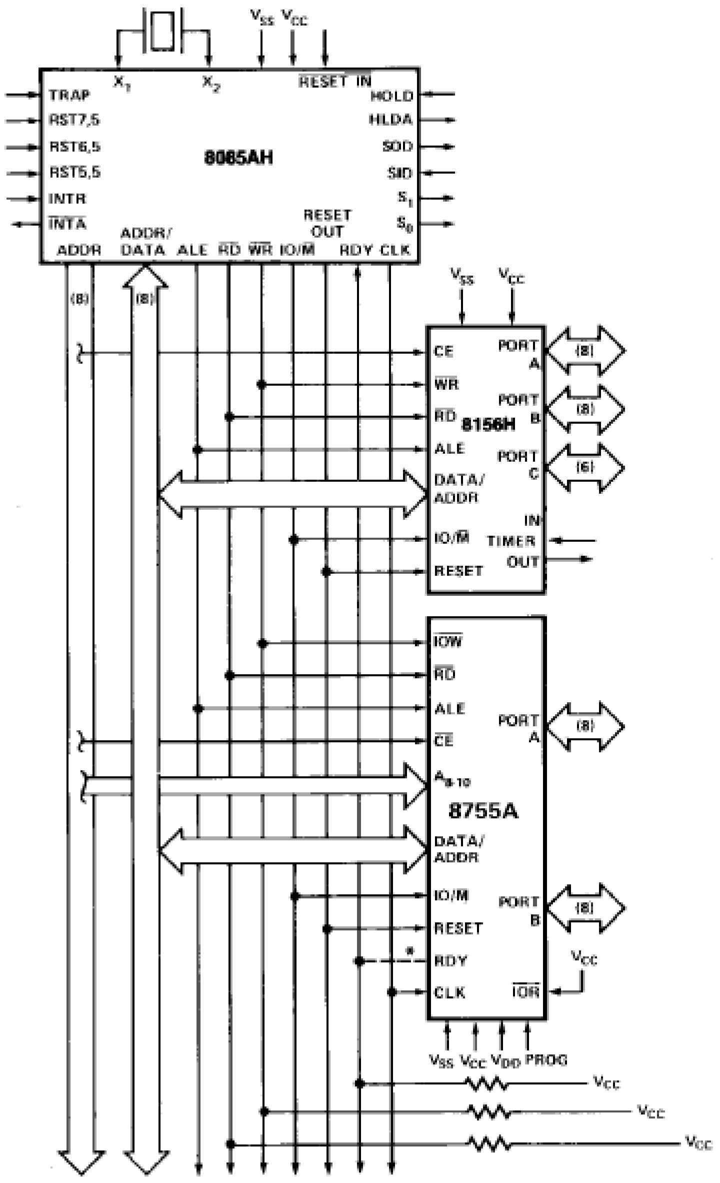


Fig. 2.7 – Minimal 8085 system with EEPROM and RAM

- **An input section**:

This section is responsible for getting input from a user. Input may be commands or the contents of a program. A matrix keypad/keyboard is usually the main component in this section. Input may also be through an asynchronous serial connection with a computer; programs may be loaded into the trainer memory through such a connection.

When a keypad is used with the 8085, an interfacing chip is usually dedicated to the task of scanning the keypad regularly and reporting key presses to the 8085. Often, an Intel 8279 keyboard/display controller is used in 8085 development boards for this purpose as shown in Fig. 2.8.

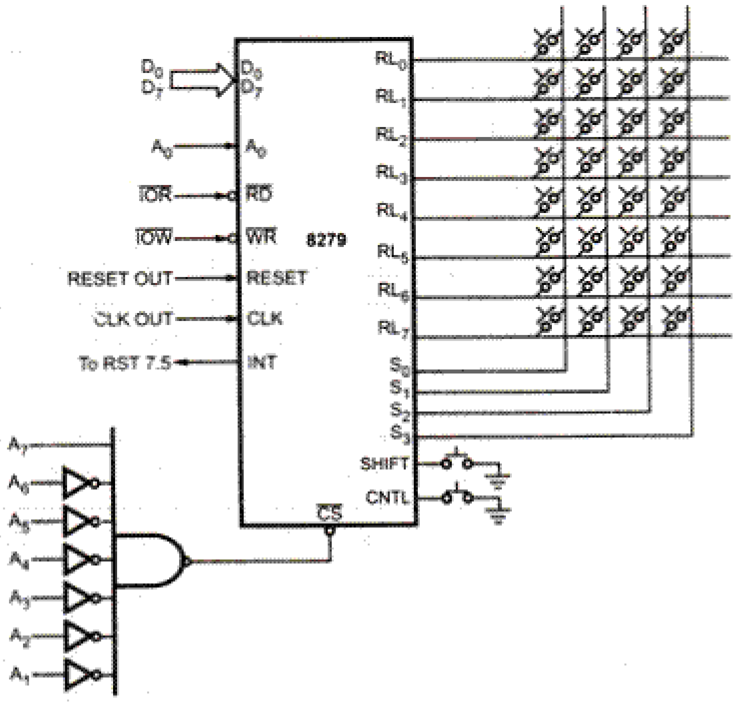


Fig. 2.8 – Interfacing an Intel 8279 with an 8x4 matrix keypad

An Intel 8255 I/O chip may also be programmed to scan a keypad. The buttons on a trainer keypad are typically labelled with commands that perform the following tasks:

- Display the contents of memory cell-by-cell and allow writing of new data to memory locations
- Display the contents of each CPU register and allow their alteration
- Allow a user to alter the contents of the program counter and initiate program execution
- Reset the main CPU

- Allow a user to single-step through a program in memory
- Trigger a CPU hardware interrupt

Fig. 2.9 shows a typical trainer keypad with the some buttons labelled with functions mentioned earlier.

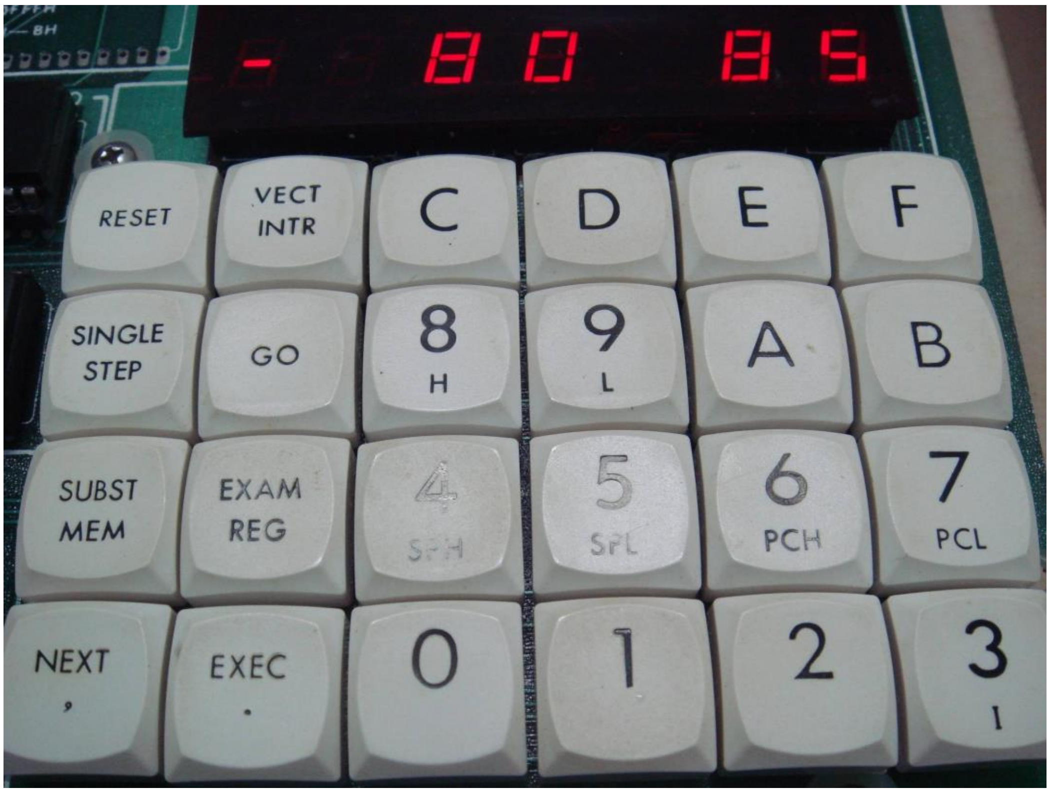


Fig. 2.9 – A typical 8085 trainer keypad

As can be seen, a typical keypad does not have a decimal digits but rather hexadecimal digits 0 – F, implying that addresses and data will be entered in hex format. This is because the convention is to enter assembly code literals in hexadecimal format rather than decimal format, due to the ease of conversion to binary format.

When asynchronous serial input is employed, either through an RS-232 connection or some other external driver, an Intel 8251 UART chip is typically used to interface the 8085 with the external driver, with additional level shifting provided as needed. A menu may be provided to users in the serial port monitoring program on the PC, allowing a user to perform all the actions that can be done with a keypad.

- **A display**:

As the name implies, this provides a means for a user to view the results of their interactions with the trainer. It displays the results of user commands or button presses. The display is usually one or more multiple-digit seven-segment LED displays; this usually suffices because the display for a trainer is not required to be particularly expressive. Fig. 2.10 shows a typical display for a microprocessor trainer.

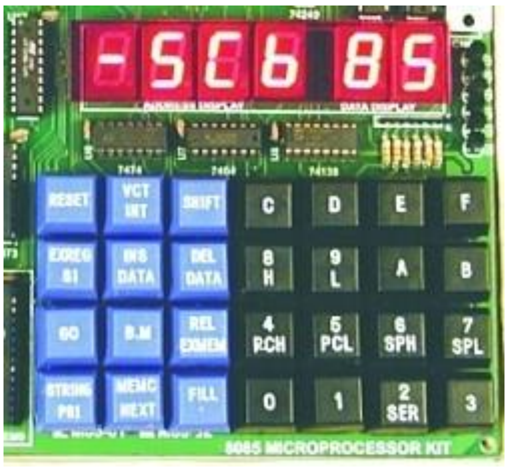


Fig. 2.10 – 6-digit seven-segment display for the TLA801 trainer

It comprises a 4-digit SSD and a 2-digit SSD. The 4-digit SSD usually shows the address and the 2-digit SSD shows the data, whenever the user accesses memory. This is convenient since the 8085 has an address space of 64K with addresses ranging from 0000h to FFFFh – no more than 4 digits are needed to display any address. The 8085 is also an 8-bit CPU, therefore the range of values for data is from 00h to FFh – no more than 2 digits are needed to display the contents of any memory location. Thus, the width of the display depends entirely on the range of values to be displayed. SSDs are used in many 8085 trainers because they are easily driven by the 8085 through the support chips that can act as display controllers such as the 8279. SSDs are limited though since they are naturally intended to display decimal digits and

Some trainers employ LCDs which provide more freedom in terms of what can be displayed but also require more complexity in driving them. Therefore, 8085-driven LCDs are typically character LCDs, as shown in Fig. 2.11, because of their relatively simple interface.

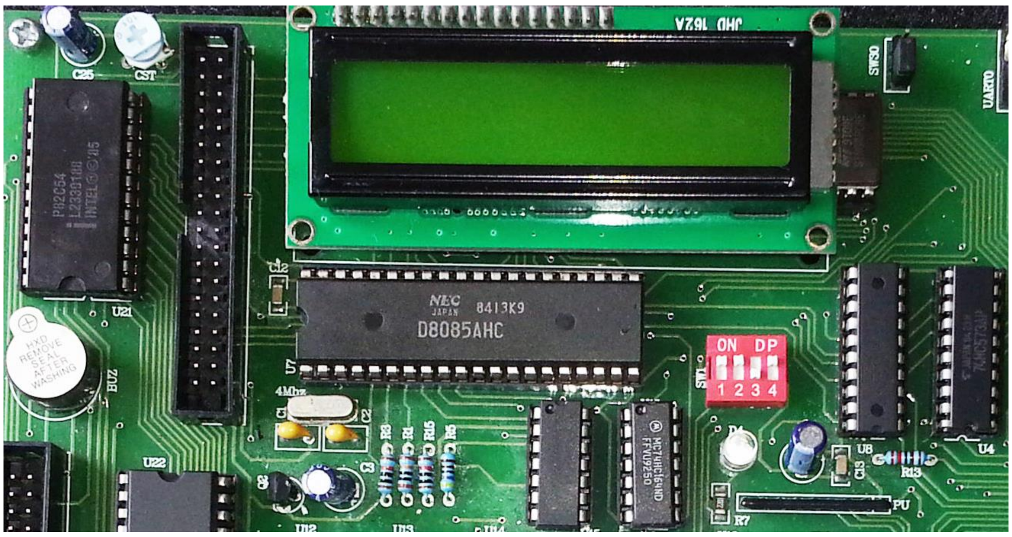


Fig. 2.11 - A 16x2 character LCD in the BT-85 trainer

- **I/O Peripherals**:

Peripheral devices such as LEDs and switches may be provided to the user as a means of interaction between the input 8085 programs and the outside world. These peripheral devices serve as general purpose input and output to provide physical input or visualize the results of user programs. I/O devices cannot be driven directly by microprocessors since they do not have latches to hold output data nor do they have the current sourcing and sinking capability. Instead I/O driver chips like the Intel 8255 serve as an interface between the microprocessor buses and the I/O devices.

I/O devices may be memory-mapped or I/O-mapped. Memory-mapped I/O devices share the same address space with RAM and ROM. They may be written to or read from just like any memory location with the same instructions (such as MOV, LDA, STA, etc.). I/O-mapped I/O devices have a separate address space and may only be accessed through special IN and OUT instructions.

The Intel 8255 is a programmable peripheral interface designed for use with Intel microprocessors [13]. It has three 8-bit programmable I/O ports – ports A, B and C. It has no real address bus but its data bus is connected to that of the 8085. The 8255 has a Chip Select pin which is used to enable communication between itself and the 8085. Address decode logic is required to decode the assigned base address of the peripheral device to a single CS signal. Any port may be selected based on the combination of signals on its A0 and A1 pins.

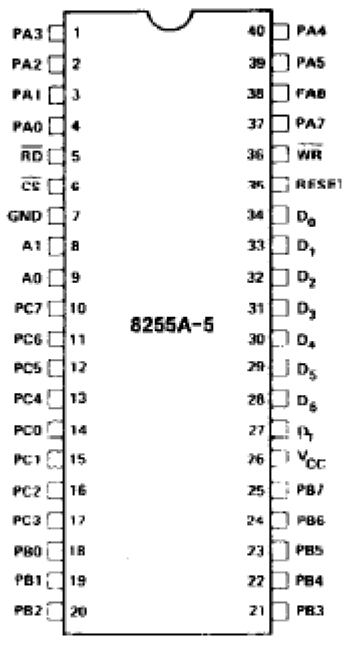


Fig. 2.12 – Intel 8255 pin-out diagram

Fig. 2.12 illustrates the pin-out configuration of the Intel 8255. LEDs may not be driven directly from its ports since they have very limited drive capability.

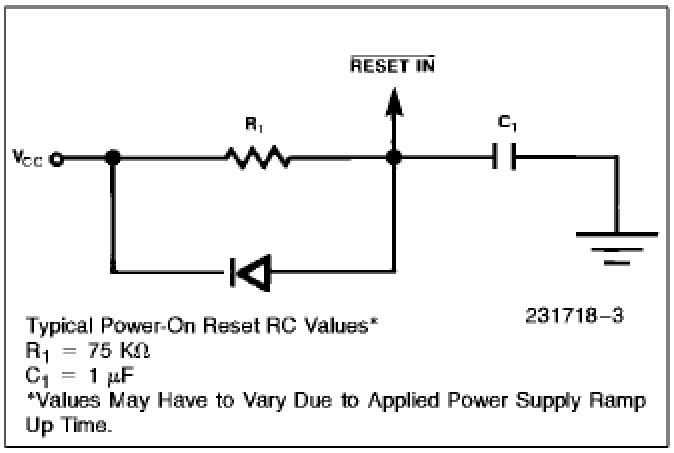
Instead, an I/O pin may drive a typical BJT, which will in turn drive an LED or some other output device.

2.3 – INTEL 8085 INTERFACE

This section describes the necessary interfacing needed for the Intel 8085 to function properly. It describes a minimal system comprising an 8085 and a generic compatible SRAM.

- **RESET CIRCUIT:**

The recommended reset circuit of an Intel 8085 typically consists of a diode, resistor and capacitor as shown in Fig. 2.13:



$\overline{\text{RESET IN}}$
R_1
C_1
V_{CC}
231718–3
Typical Power-On Reset RC Values*
R_1 = 75 KΩ
C_1 = 1 µF
*Values May Have to Vary Due to Applied Power Supply Ramp Up Time.

Fig. 2.13 – Intel 8085 Power-on Reset Circuit [7]

The capacitor acts to reduce the rise time of V_{CC} on the RESET_IN pin when the 8085 is powered up while the resistor controls the rate of capacitor charging and as a pull-up resistor as well. On power-on, the capacitor charges through the resistor and will be at 63% of V_{CC} (3.15 V) after one time constant, which depends on the values of R_1 and C_1. The minimum input-high voltage for a reset signal is 2.4 V [7]. This means the device will have left the reset state before $t = \tau$ after power-on.

If the recommended values are used:

$$\tau = RC = 75 \times 10^3 \times 10^{-6} = 75\,ms$$

The minimum time for the 8085 to exit reset state is given by:

$$\frac{2.4}{5} = 0.48 = 1 - e^{-\frac{t}{\tau}}$$

$$0.52 = e^{-t/\tau}$$

$$-\ln 0.52 = \frac{t}{\tau} = 0.6539$$

$$t = 0.6539 * \tau = 0.6539 * 75\,ms = 49.04\,ms$$

Thus, after the 8085 is powered with a V_{CC} of 5 V, it will take at least 49 milliseconds for the 8085 to begin program execution at the reset vector. The microprocessor may be reset again at any time by applying a LOW signal directly to the RESET_IN pin. This causes the capacitor to rapidly discharge through the low impedance of the applied GND signal. Therefore, placing the CPU into the reset state is a lot faster than releasing it from the reset state. The RESET_OUT pin of the 8085 is typically connected to the reset inputs of the other system components so that a reset of the 8085 causes a reset of the entire system.

A reset on the 8085 clears the Program Counter and tri-states the buses. The 8085 remains in reset state for as long as a LOW signal is applied to the RESET pin.

- **CLOCK CIRCUITRY**

A clock is necessary for any microprocessor to sequence its operations and synchronize the operations of the other components of the system. The typical clock sources are:

- Quartz crystal of at least 1 MHz as in Fig. 2.14

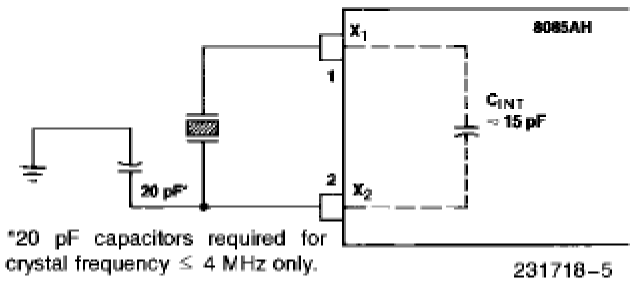

8085AH
X_1
1
C_{INT}
≈ 15 pF
2
X_2
20 pF*
*20 pF capacitors required for crystal frequency ≤ 4 MHz only.
231718–5

Fig. 2.14 – Quartz crystal clock circuit

Crystal frequency should be twice the desired internal clock frequency.

- An LC tuned circuit as in Fig. 2.15

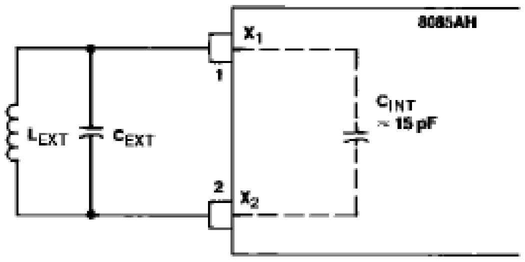


Fig. 2.15 – LC tuned circuit clock driver

- An RC circuit as in Fig. 2.16

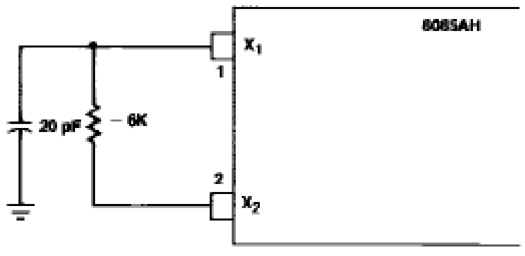


Fig. 2.16 – RC circuit clock driver

- An generic external clock signal as in Fig. 2.17

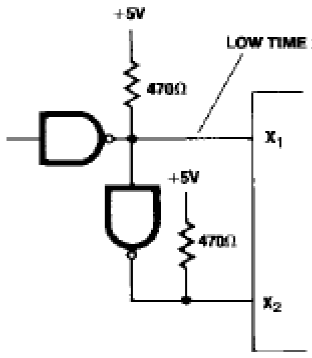


Fig. 2.17: 1 – 12 MHz external clock driver circuit

The 8085 has a CLK output pin which can be used to provide a clock signal to the system components; this may be considered as the system frequency. The frequency is the same as the internal clock frequency.

- **ADDRESS AND DATA BUS INTERFACE**

The address bus is 16 bits wide, with the lower 8 bits multiplexed with the data bus. When interfacing with memory or I/O devices:

- The high-order byte of the address bus is connected directly to the high-order byte of the address bus of the system components.
- The low-order byte of the address bus is typically connected to an octal latch, whose output is then connected to the low-order byte of the address bus. The latch is enabled through the ALE pin of the CPU. A falling edge on the ALE pin latches the inputs of the octal latch of its outputs. This occurs in the first clock cycle of a machine cycle. For the rest of the machine cycle, the low-order byte of the address bus functions as the data bus and is connected directly to the data bus of the system components. Fig. 2.18 illustrates the interfacing circuit between the A/D bus of an 8085 and memory using the typical 74LS373 octal latch.

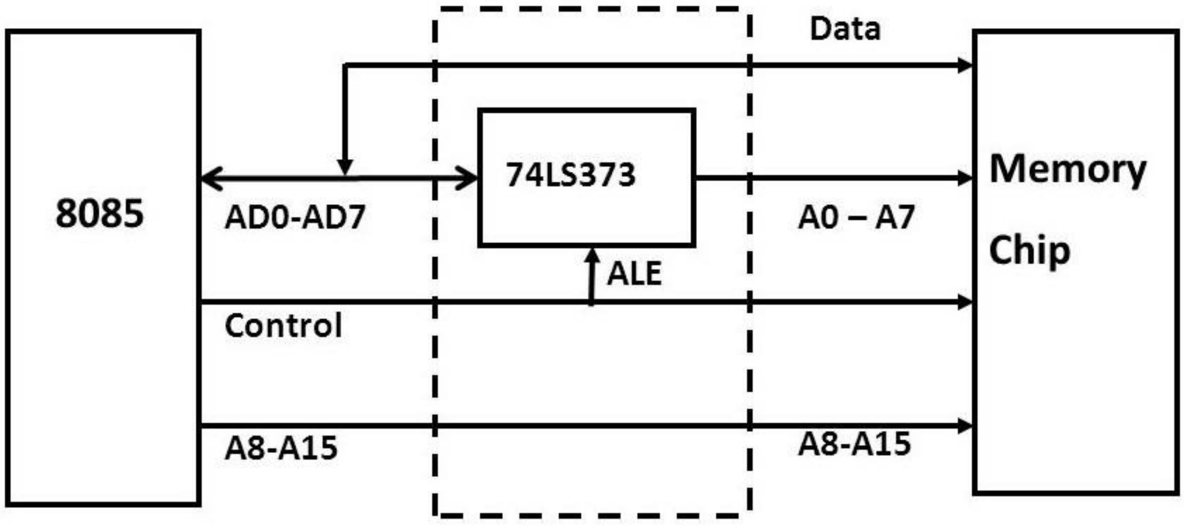


Fig. 2.18 – Intel 8085 address and data bus interface

- **CONTROL BUS INTERFACE**

The control bus comprises the signals used by the CPU to coordinate the operations of the system components. These include the following:

- $\overline{RD}, \overline{WR}, IO/\overline{M}$

The $IO/\overline{M}$ signal is used by the CPU to select a memory or I/O device; a LOW signal selects memory while a HIGH signal indicates an I/O operation is to be performed. This signal may be connected directly to memory and I/O devices if they have a provision for it, else the signal may pass through address-decode logic before ultimately selecting the right device. The $\overline{RD}$ and $\overline{WR}$ signals are active-LOW signals. They are connected to memory and I/O devices and are used by the CPU to indicate a read or write operation to the selected device.

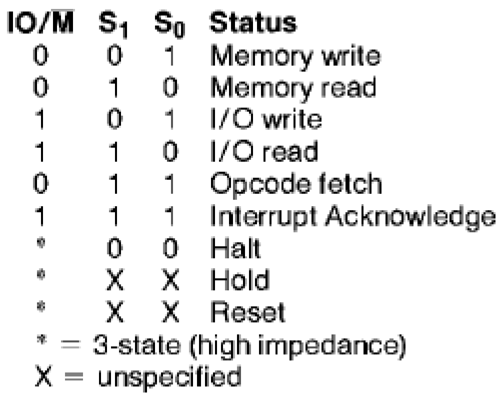
- READY, HOLD, $\overline{HLDA}$

The READY signal is an active-HIGH input signal used by slower system components to introduce wait-states in the program execution flow of the 8085. When unused, it should be connected to V_{CC}. HOLD is an active-HIGH input signal is used by other system components to request the control of the system buses from the CPU. $\overline{HLDA}$ is an active-LOW output signal which is asserted by the 8085 when it is ready to relinquish the bus to a requestor.

- **STATUS SIGNALS**

The S0 and S1 signals of the 8085 may be used, in combination with $IO/\overline{M}$, to determine the current machine cycle status of the 8085. Table 2.1 interprets the possible status signals to indicate the current operation being performed by the CPU.

Table 2.1 – Interpretation of the status signals of the Intel 8085



$IO/\overline{M}$	S_1	S_0	Status
0	0	1	Memory write
0	1	0	Memory read
1	0	1	I/O write
1	1	0	I/O read
0	1	1	Opcode fetch
1	1	1	Interrupt Acknowledge
*	0	0	Halt
*	X	X	Hold
*	X	X	Reset

* = 3-state (high impedance)
X = unspecified

- **INTERRUPT SIGNALS**

The 8085 has 5 hardware interrupt sources:

- INTR – A general-purpose interrupt
- RST5.5, RST6.5, RST7.5 – These are maskable RESTART interrupts, arranged in the order of increasing priority. They cause the CPU to branch to some pre-set address.
- TRAP – This is a non-maskable RESTART interrupt and has the highest interrupt priority

When unused, the interrupt pins should be tied to GND since they are all active-HIGH inputs.

2.4 – CHAPTER SUMMARY

This chapter reviewed past work on microprocessor trainers, especially those based on the Intel 8085. It also detailed the different components of a microprocessor trainer, as well as the necessary interfacing for an Intel 8085 in a microprocessor-based system.

CHAPTER 3

DESIGN AND METHODOLOGY

This chapter details the rationale behind the design choices made in this project as well as the methods employed to achieve the goal of the project.

3.1 – DESIGN OBJECTIVES

To design and construct an Intel 8085 microprocessor trainer:

- With the minimal capabilities of accepting user programs in machine code, executing them and displaying the results in a modern intuitive manner
- That can be manufactured relatively cheaply, compared to existing trainers
- That can be powered by common AA batteries or AC mains
- With at least one I/O peripheral

3.2 – DESIGN CONSIDERATIONS

The following factors were considered in the design of the microprocessor trainer:

- **Cost**

The cost of manufacturing the trainers was one of the primary factors considered during the design process. Manufacturing a trainer, from PCB fabrication to assembling, has to be inexpensive enough for the department to seriously consider doing so in large quantities.

- **Ease of use**

A microprocessor trainer must be easy to use and intuitive considering that it is intended for students who have most likely had no prior experience with programming. The traditional trainer interface does not offer much in terms of help to inexperienced students. This is an important factor that must be considered especially in a country like Nigeria, with a high rate of ICT-illiteracy.

- **Power dissipation**

For the trainer to be battery-powered, it must have a low power dissipation. This means the components of the system must draw little current, not only to conserve battery power but also because excessive current draw reduces the terminal voltage of the batteries and increases heat dissipation within the regulators in the circuit.

- **Compatibility**

As much this project is intended to be an improvement on existing trainers in terms of user interface, it must also offer some backward compatibility to existing trainers so that students and technologists can easily adapt to using them.

3.3 – DESIGN SPECIFICATIONS

The desired specifications for the resultant microprocessor trainer of this project are as follows:

- An internal system frequency of 500 kHz for the Intel 8085
- 2K x 8 static RAM, of which at least 1k will be user RAM
- One switch and one LED as I/O-mapped I/O devices
- A hexadecimal keypad that also contains all the relevant commands for operating the trainer
- An intuitive LCD display
- An independent system monitor that will act as the system front-end and also control all trainer operations

- A power supply circuit that can accept a maximum of 12 V DC from any source, whether rectified AC or batteries, and output the voltages needed for the trainer to function
- A maximum current draw of 100 mA at any instant

3.4 – HARDWARE DESIGN

This section details the design choices and procedure in the hardware design process of the microprocessor trainer. Fig. 3.1 illustrates a block diagram of the microprocessor trainer.

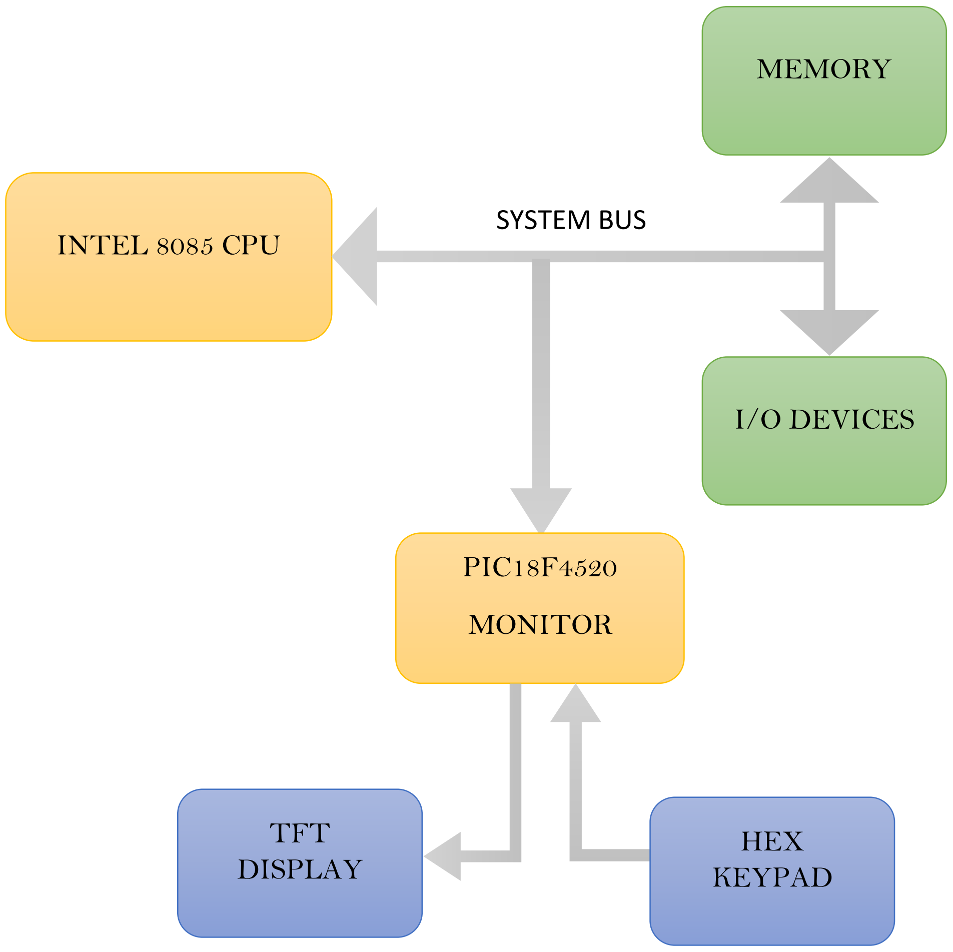


Fig. 3.1 – Block diagram of the Intel 8085 trainer

3.4.1 – BILL OF MATERIALS

Below, in Table 3.1, is the list of electronic components used in the construction of the microprocessor trainer:

Table 3.1 – Bill of materials for the Intel 8085 trainer

COMPONENT NAME	QUANTITY	PACKAGE
Intel 8085AH-2 microprocessor	1	DIL-40
HM6116ALP-3 SRAM	1	DIL-24
Intel 8255A PPI	1	DIL-40
74HC373 octal latch	1	DIL-20
74LS138 3-to-8 line decoder	1	DIL-16
PIC18F4520 microcontroller	1	DIL-40
MCP23S17 GPIO expander	1	DIL-28
1.8” 128x160 TFT display with ST7735 driver	1	SIL-16
4x5 membrane matrix keypad	1	SIL-9
LM2940 LDO regulator	1	TO-220
LM1117 LDO regulator	1	TO-220
BC547 NPN transistor	1	TO-92
IN4148 diode	1	
0.1uF ceramic capacitor	10	2mm pitch
22pF ceramic capacitor	1	
1nF ceramic capacitor	1	
0.47uF electrolytic capacitor	1	
1uF electrolytic capacitor	2	
22uF electrolytic capacitor	1	
1 MHz quartz crystal	1	

68kΩ resistor	1	
10kΩ resistor	5	
1.8kΩ resistor	1	
33kΩ resistor	1	
1kΩ resistor	6	
330Ω resistor	2	
Red LED	2	
SPST push-button	1	
5.5x2.1mm barrel jack	1	
TO-220 heat sink	2	
DIL-40 socket	3	
Wide DIL-24 socket	1	
DIL-16 socket	1	
DIL-28 socket	1	
DIL-20 socket	1	
SIL-16 header	1	
SIL-9 header	1	
SIL-4 header	1	

3.4.2 – DESIGN CHOICES

- **CPU**

The Intel 8085 is the CPU under study in this project. The trainer is built around it. Its internal clock frequency was chosen to be 500 kHz so that the 8-MIPS monitor can easily track the 8085's operations on the buses. A quartz crystal was chosen to provide the clock signal due to the simplicity of the clock circuit as well as the inherent stability of quartz crystals. The 8085's clock and reset circuitry is very similar to what is recommended in its datasheet [7]. Since 8085 interrupts are outside the scope of this project, all the interrupt pins but RST7.5 are tied to GND to prevent any spurious interrupts if left floating. RST7.5 is connected to the monitor and used to exit the HLT state. The HOLD and READY pins are both tied to GND and V_{CC} respectively since they are unused and are best not left floating. The address, data and control buses are interfaced as described in earlier chapters to memory, I/O and the system monitor. The status signals are also connected to the monitor.

- **Memory**

The HM6116ALP-3 was chosen as a memory chip due to its compatibility with the 8085 buses. Its capacity is 2 KB which is sufficient for a standard student trainer. The allocated user RAM is 1 KB, which is an improvement on the 256 bytes offered by the existing lab trainers. The 6116 also has fast access time and easily accommodates the 500 kHz operation of the 8085.

Fig. 3.2 shows the microprocessor, SRAM and octal latch purchased for this project.

Fig. 3.2 – Intel P8085AH (left), HM6116ALP-3 (top right), 74HC373 (bottom right)

- **I/O peripheral**

The Intel 8255A is the chosen I/O peripheral due to its availability and 8085-compatible data and control interface. It offers 3 programmable 8-bit I/O ports. A 74LS138 decoder was used to decode the bits AD7-AD3 of the 8085 address bus and the $IO/\overline{M}$ pin to yield a single Chip Select signal for the 8255. The use of the $IO/\overline{M}$ pin ensures that the 8255 is an I/O-mapped I/O peripheral. Bits AD1-AD0 are connected to the A1 and A0 pins respectively of the 8255 and are used to select one of its three I/O ports for read/write operations. Fully decoded, the I/O base addresses of the 8255 are 30H and 34H. It has 2 addresses because the address bits are only partially decoded; bit AD2 does not play a part in selecting the 8255.

An LED and one push-button were connected to the bit 0 of ports A and B respectively. The LED is not driven directly by the 8255 but rather through a BJT, since the 8255 lacks the necessary drive current. The button is connected with RC debounce circuitry. Both LED and pushbutton can be read or written to by IN and OUT instructions. They were chosen so that students can see some visual results or provide physical inputs to the programs they write. The reset pin of the 8255 was connected to the reset pin of the 8085 RESET_OUT pin to ensure simultaneous reset of both devices every time.

Fig. 3.3 shows the Intel 8255 purchased for this project.

Fig. 3.3 – Intel 8255A

- **System monitor**

This is perhaps the most significant difference between the microprocessor trainer developed in this project and most existing trainers. The monitor used in this project is not merely a program residing in EEPROM to be executed by the 8085. It is actually a separate piece of hardware entirely – a PIC18F4520 microcontroller. Employing a separate device as a monitor has a huge advantage in terms of fast and relatively easy program development. It is a lot easier to write C programs for an MCU like the PIC18F4520, which has an established and supported tool-chain, than it is to write assembly programs for the 8085. This is even more significant when one considers that a microprocessor trainer monitor program is reasonably complex and would require significant experience in assembly programming, not to mention considerable time, to design it in that language.

The PIC18F4520 was chosen for several reasons. The first of these is its 16K program memory and 1536 bytes of data memory [14], which are sufficient for this project. It also has an 8 MHz internal clock which may be multiplied by a 4x internal PLL to yield a clock frequency of 32 MHz. Due to the nature of the PIC microcontrollers, a 32 MHz clock actually translates to 8 MIPS. Therefore, the PIC is expected to have no trouble keeping up with the bus activities of the relatively slow (500 kHz ≈ 0.05 MIPS) 8085 in the former's duties as system monitor. The PIC also has 35 I/O pins as well as SPI and UART peripherals, which are of significant importance in interfacing it with the other components in the system such as memory and display. The PIC18F4520 is supported by Proteus thus ensuring easy simulation. The MPLABX IDE, together with the XC8 compiler, provide a conducive environment for developing the monitor software.

- **Keypad**

The trainer keypad is supposed to have a button for each hex digit as well as all the common commands needed to operate the trainer. Based on this factor, at least 24 buttons are needed for the keypad. However, the common keypads in online stores are typically 3x4 and 4x4 matrix keypads. Attempts to locally build a 6x4 button keypad were only partially successful; the assembled PCB was of poor quality and had switch debouncing issues that were not easily rectified. Therefore, the closest alternative found on Aliexpress was a 4x5 membrane matrix keypad, as shown in Fig. 3.4.

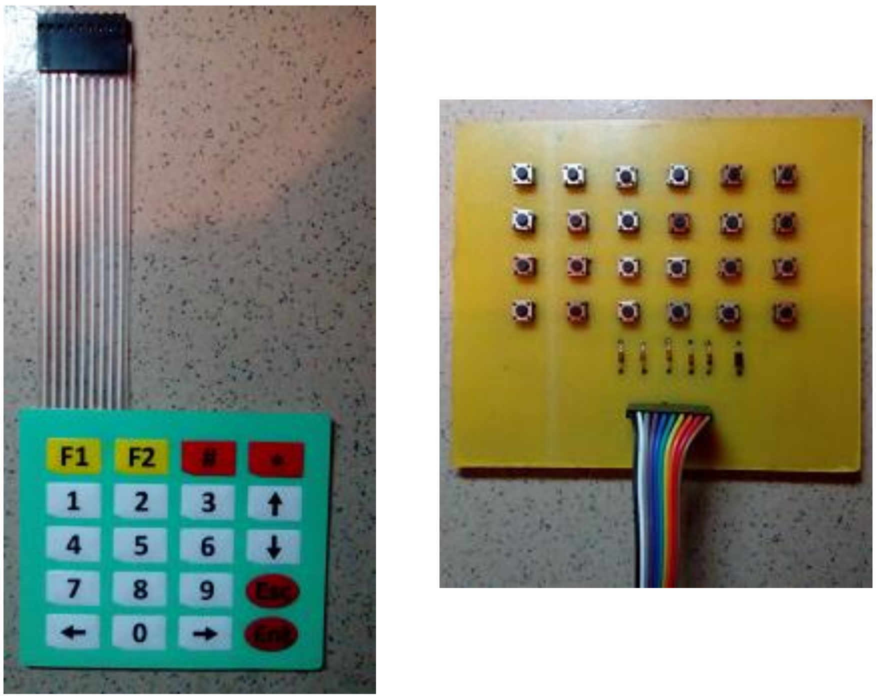


Fig. 3.4 – 4x5 membrane keypad from Aliexpress (left), locally assembled 6x4 keypad (right)

A membrane keypad was advantageous because switch bounce is close to negligible in such keypads. To get around the problem of not enough buttons, a few buttons were assigned dual functions in order to make up for the deficit. A button was programmed to act as the SHIFT button in order to select the alternative function of the dual-function buttons.

The keypad was interfaced with the PIC18F4520 monitor, through the MCP23S17 GPIO expander chip. Since a 4x5 matrix keypad would demand 9 digital pins on the monitor, a GPIO expander helps relieve some of this burden. Only 4 pins are required to control this particular expander chip as opposed to the 9 pins that would have been used by the keypad if interfaced directly.

The MCP23S17 was chosen for several reasons. It offers two 8-bit configurable ports which are more than sufficient for the keypad. It also has a straightforward SPI control interface; indeed, “software SPI” was implemented with ease to communicate with the I/O expander. This is in contrast with the counterpart MCP23017 which has an I2C interface – a protocol that is not easily set up on PIC MCUs when using the hardware peripheral and is not easily implemented in software either. The MCP23S17 also offers built-in pull-up resistors for its port pins, thus eliminating the need to wire up resistors externally for the keypad.

- **Display**

Traditionally, seven segment displays are used as the display device for microprocessor trainers. They offer easy control for the monitor, especially when combined with Intel 8279 as a display controller. One must also consider that the monitor code is typically written in 8085 assembly and it is not trivial to write a driver for more sophisticated displays in such a language. Trainers also do not need much in the way of display.

However, since a separate MCU, whose code is written in C instead, takes the burden of a monitor in this system, it is significantly easier to write code for more complex devices such as is required for the driver of TFT displays. Therefore, a 1.8” TFT LCD with a resolution of 128x160 pixels was chosen for this project, as illustrated in Fig. 3.5. It is an attempt to provide a more modern UI in contrast with the vintage-looking SSDs. A TFT display also offers a more expressive display

– error messages can be more detailed, help messages may be provided, etc. 1.8 inches is large enough for the trainer's purposes and yet small enough that it will not consume too much power. The driver for this particular display is the ST7735 which has an existing library that was easily ported to XC8 C.

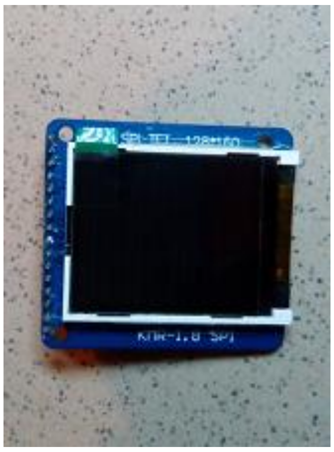


Fig. 3.5 – 1.8" 128x160 TFT display

- **Power**

Power supply is an important consideration in any electronic system. In keeping with the goals of the project to create a battery-powered trainer, the input to the trainer is DC from any source up to 12 V through a common barrel jack connector. There is no AC-DC conversion circuit within the trainer, thus placing the burden on conversion, if needed, on the user. This allows a user to connect their own power source, whether 12 VDC from a wall adapter or 6 V from 4xAA batteries. A slide switch was used as the power switch for the trainer.

An LM2940 LDO regulator is used to step down the input DC voltage to 5 V for the system components. Another LDO regulator – an LM1117T – further steps down the 5V to 3.3 V for the TFT display. Linear regulators were chosen as opposed to switching regulators due to the clean relatively noise-less output yielded by the former. The output of switching regulators is inherently noisy and full of high-frequency harmonics which are undesirable in a trainer which routes high-frequency signals that are vulnerable to EMI. The low efficiency of linear regulators is acceptable as well as the heat losses, considering the trainer is not expected to draw any significant current.

The trainer is comprised mostly of ICs driving the high-impedance inputs of other ICs. The biggest single power consumer is the TFT display which may draw as much as 50 mA. Therefore, the current drawn by the entire trainer is not expected to exceed 200 mA at any instant. 0.1uF decoupling capacitors are connected as close as possible to the power pins of each IC to deal with dips in V_{CC} during high frequency operation. Decoupling capacitors are also connected close to the input and output of the regulators to further stabilize their output and guard against the effects of noise.

3.4.3 – SCHEMATIC DIAGRAM AND PCB DESIGN

A schematic diagram of the entire trainer circuit was drawn up in Eagle CAD 7.3.0 using a combination of the default parts libraries and the Sparkfun libraries. Net connections were used to make connections between devices in order to avoid the clutter of wired connections.

A 2-layer PCB was designed in Eagle CAD for the trainer, based on the drawn schematic. Through-hole packages were used for every component to allow easy assembly and soldering. Two mounting holes were included for fastening the PCB to a platform. The trace width was 24 mils for power traces and 10 mils for other signals. A ground pour was used on the top and bottom layers to minimize the path to GND for any particular connection and also to aid decoupling. The dimensions of the final PCB were 7.60 inches x 4.85 inches.

3.5 – SOFTWARE DESIGN

The monitor oversees the operations of the system. Therefore, the monitor program is the only program of note to be developed for the trainer. This section details the design process of the monitor software.

3.5.1 – DEVELOPMENT ENVIRONMENT

The monitor for this trainer is a PIC18F4520 MCU. Its code was developed in C partly within the Proteus environment and partly within the MPLABX IDE. The XC8 compiler was used for this project. XC8 is intended for 8-bit Baseline and Mid-range PIC MCUs. A PICkit 2 programmer was used to burn programs into the monitor flash memory via ICSP.

3.5.2 – HARDWARE DRIVERS

Drivers were written for each of the devices which are linked with the monitor. Existing libraries were ported as in the case of the TFT display and MCP23S17. Code snippet 3.1 shows the functions that were implemented in order to drive the TFT display.

```
void init_screen(void);
void set_window(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1);
void fill_screen(uint16_t color);
void draw_pixel(int16_t x, int16_t y, uint16_t color);
void draw_vline(int16_t x, int16_t y, int16_t h, uint16_t color);
void draw_hline(int16_t x, int16_t y, int16_t w, uint16_t color);
void draw_rect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
void fill_rect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
void draw_round_rect(int16_t x, int16_t y, int16_t w, int16_t h, int16_t r, uint16_t
color);
void fill_triangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2,
uint16_t color);
void writech(uint8_t c);
void draw_char(int16_t x, int16_t y, unsigned char c, uint16_t color, uint8_t size);
void set_rotation(uint8_t r);
void set_cursor(int16_t x, int16_t y);
void set_text_color(uint16_t c);
uint16_t color_565(uint8_t r, uint8_t g, uint8_t b);
void set_text_size(uint8_t s);
```

Snippet 3.1 – Functions in the TFT driver header file

New libraries were developed for I/O operations on the HM6116, as shown in Snippet 3.2, and for interacting with the 8085 itself.

```
void init_mem_interface(void);
void mem_config(char state);
unsigned char read_mem(unsigned int addr);
void write_mem(unsigned int addr, unsigned char dat);
void write_mem_cont(unsigned int start, unsigned char data[], unsigned char len);
void read_mem_cont(unsigned int start, unsigned char buf[], unsigned char len);
```

Snippet 3.2 – Functions in the SRAM driver header file

A few utility functions were also written for UART debugging, millisecond and microsecond delays, etc.

3.5.3 – MICROPROCESSOR TRAINER COMMANDS

There are 4 commands supported by this microprocessor trainer. They are the minimum required functions needed to achieve the project's goals. They are as follows:

- **SUBST MEM**

This command is used to access RAM, view its contents and change them if needed. The general sequence of events for this command is:

SUBST MEM → Enter an address → NEXT → Enter data (optional) → NEXT → Enter data (optional) → NEXT → … → EXEC

NEXT and EXEC are buttons on the keypad. NEXT is used to move to the next address or the next stage of a command whereas EXEC is used to complete a command and return to the IDLE state of the system. The ellipsis (…) indicates one can keep moving through memory, altering its contents as needed, till one gets to the boundary and is returned to the beginning. This command is typically used to enter programs in machine code into the RAM, byte-by-byte. The hex keypad is used to enter all literals.

- **GO**

This command is used to set the contents of the program counter and instruct the 8085 to begin execution from the specified address. The general sequence for this command is:

GO → Enter address → EXEC

After EXEC is pressed, the 8085 begins executing the instructions in RAM beginning from the specified address. It does not stop till it encounters a HLT instruction.

- **EXAM REG**

This command is used to instruct the trainer to display the contents of the 8085 registers. Each register's contents are displayed on demand with the NEXT button used to cycle through the registers. The general sequence of events is:

EXAM REG → (Register A contents are displayed) → NEXT → (Register B contents are displayed) → NEXT → … → (Stack pointer contents are displayed) → NEXT/EXEC

Registers A, B, C, D, E, H, L, flags, Program Counter and Stack Pointer are all displayed in sequence.

- **RESET**

This command serves to reset the 8085 and return the microprocessor trainer to the IDLE state.

3.5.4 – SOFTWARE ARCHITECTURE

The entire monitor program was designed as a Mealy finite state machine. The trainer's operation is very much dependent on user input. The keypad is scanned constantly to determine the next task to be performed. The tasks themselves are already known; the user's inputs only determine which ones are performed and when they are performed. Also, some commands share some processes in common – for instance, SUBST MEM and GO commands both collect an address from the user. A state machine was chosen because it is an effective and efficient way to define the system, as opposed to lengthy IF-ELSE structures. A state-machine approach also makes the system modular since the designer is forced to break down the system into manageable states and their transition functions.

There are 7 possible states as defined in the struct below:

```
typedef enum {
    ENTERING_IDLE,
    IDLE,
    SUBST_MEM,
    ENTER_DATA,
    GOTO_ADDR,
    WAITING_MPU,
    EXAM_REG,
} State_Type;
```

Snippet 3.3 – The possible states of the monitor

There are also 7 functions, one for each state:

```
void (*state_table[])() = {entering_idle,
                           idle_state,
                           subst_mem,
                           enter_data,
                           goto_addr,
                           waiting_mpu,
                           exam_reg};
```

Snippet 3.4 – State machine functions

On start-up, the monitor performs the following tasks in the *main()* function:

- Set the PIC clock source and enable the 4xPLL
- Places the 8085 in reset state
- Initialize UART debugging

- Initialize the keypad interface
- Initialize the TFT display
- Initialize the memory interface and configure it for R/W operations
- Initialize interrupts needed for capturing ALE events

In the *main()* function, a *while* loop runs indefinitely, executing the transition function for the current state. A variable is defined to keep track of the current state of the monitor. The default state on system start-up is the ENTERING_IDLE state which just clears the display and prints "READY" on the screen before proceeding to the IDLE state. The ENTERING_IDLE state is executed only once at any given time and serves merely as a precursor to prepare the system before immediately proceeding to the IDLE state.

While in any state, each time the state transition function is called, the keypad is scanned once and if there are any pressed buttons, a decision is made to proceed to another state or perform some action if the pressed button is one of the currently allowed inputs. If the pressed button is not one of the allowed inputs, it is merely ignored or in a few cases, an error message is raised and the system returns to ENTERING_IDLE state.

A summary of the states and their allowable inputs is presented below:

ENTERING_IDLE: There are no allowed inputs. Its function is executed only once before it updates the current state to IDLE.

IDLE_STATE: In this state, the keypad is polled to obtain the user's command. The allowed inputs are the SHIFT, SUBST_MEM, EXAM_REG, GO, RESET buttons.

SUBST_MEM: In this state, the RAM address, whose contents are to be examined or altered, is entered by the user. The next state is typically the ENTER_DATA state. The allowed inputs are the HEX_KEY, NEXT, RESET buttons.

ENTER_DATA: In this state, the user enters the data to be filled into the current memory location or simply advances to the next location without changing anything. The allowed inputs are the SHIFT, HEX_KEY, NEXT, EXEC, RESET buttons. NEXT advances to the next address while EXEC returns the monitor to the ENTERING_IDLE state.

GOTO_ADDR: In this state, the address intended for the Program Counter is entered and execution begins when the NEXT button is pressed. The monitor preloads a JMP instruction to the specified address at the 8085 reset vector in RAM and releases the 8085 from the reset state. The allowable inputs are the SHIFT, HEX_KEY, EXEC, RESET buttons.

WAITING_MPU: In this state, the monitor regularly polls the status pins S1 and S0 of the 8085 to know when a HLT instruction has been encountered, signalling the end of execution and the tri-stating of the 8085's A/D and control pins. When execution is done, this monitor proceeds to the IDLE_STATE from this state. The only allowable input is the RESET button which is useful if the user wishes to terminate the 8085's execution process.

EXAM_REG: In this state, the current contents of the CPU registers are obtained from the 8085 and stored in an array in the monitor memory.

- **Managing user program execution**

Executing a program is only half the task. A user must be able to inspect memory and the CPU registers, to determine the results of their program. Inspecting RAM is no problem at all since it is entirely separate from the CPU and easily accessible by the monitor through the shared bus. It is imperative that the monitor checks that the CPU bus pins are tri-stated, in the HALT or RESET state, before attempting to access memory. However, examining the CPU registers is not as straightforward. Extracting the contents of the 8-bit registers A – L is easy enough using '*MOV M, R*' instructions to export the register contents to RAM. But there are no such direct instructions to obtain the contents of the flag register or the 16-bit registers. The initial approach involved monitoring the 8085 ALE and status signals during user program execution. The falling edge of the ALE signal served as an interrupt source to the PIC to determine when to sample the address bus during the op-code fetch machine cycle and so constantly capture the address in the Program Counter, right until the program HLTs.

To get the contents of the other registers, an 8085 program was designed to copy the 8085 register contents to RAM and also push some data onto the stack in order to also capture the Stack Pointer contents. This program is executed immediately after the PIC monitor detects that the user's program had HLTed. A *PUSH PSW* instruction followed by a *POP PSW*, while monitoring the address lines, was used to obtain to Stack Pointer contents and, subsequently, the flag register contents. The contents of the other registers are easily obtained with '*MOV M, R*' instructions.

This method was not successful however. With an 8085 cycle period of 2 microseconds, according to the datasheet [7], the lower address byte (AD0 – AD7) must be held on the bus for at least 0.95 microseconds after the trailing edge of ALE. 0.95 microseconds is equivalent to 7.6 PIC instruction cycles at 8 MIPS. Assuming the 8085 adhered this minimum HOLD time, this means that the monitor, upon detection of falling ALE, must proceed to its interrupt routine and sample the bus all within this time window. The monitor must also check that the current machine cycle is the op-code fetch cycle, if the PC contents are of interest, or if the current cycle is the memory-write cycle after a PUSH, if the SP contents are desired. Considering that interrupt latency is between 3 – 4 cycles for the monitor and the monitor program was written in C, compiled by the free XC8 compiler with few optimizations, in contrast with tightly optimized assembly, the monitor was simply unable to keep up with the 8085. Too many instructions are executed in the ISR causing ALE events to be missed. Well-written inline assembly may be more effective but this approach was discarded in favour of a less demanding and more elegant solution.

The final solution relied on three vectors to function: the RST 0 vector, the RST 1 vector and the RST 7.5 vector. The RST 0 vector is the starting point of the CPU after it is released from RESET. The RST 7.5 vector is the address to which the CPU branches after an RST 7.5 interrupt. The RST 1 vector is the restart address for an RST 1 software interrupt. Fig. 3.6 shows the system memory map.

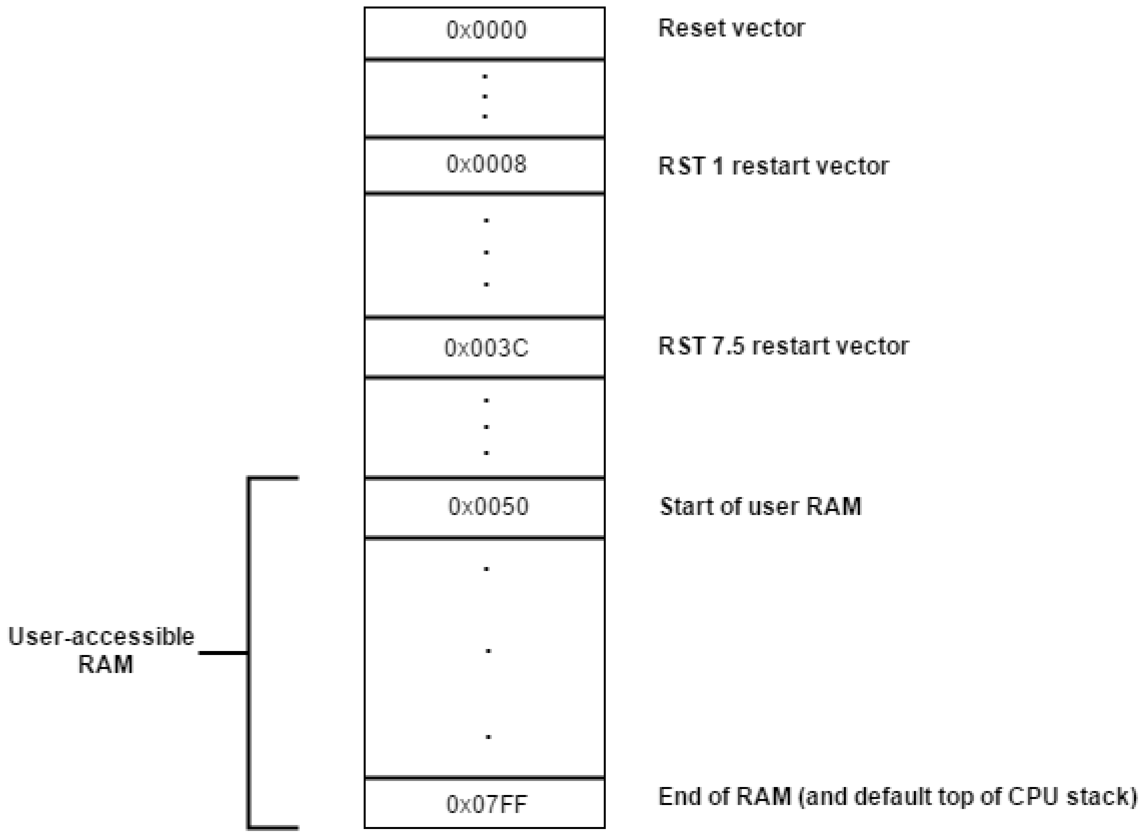


Fig. 3.6 – System memory map

As can be seen in Fig. 3.6, while entering data or programs into memory, users are only permitted to access addresses from 0x0050 to the end of available RAM at 0x07ff. The section of memory preceding 0x0050 (80 bytes) is used by the monitor as a sort of scratchpad to manage the CPU, leaving 1968 bytes for user RAM. However, these bounds are not strictly enforced because a user program could still access these addresses through the usual data transfer instructions; the intent here is to prevent accidental interference of the monitor's operations and not a determined user.

Users enter programs into memory with the SUBST MEM command and execute the programs with the GO command by indicating the start address for their program. When the EXEC button is pressed, the monitor loads two programs into memory for the CPU. The first program is intended to initialize the 8085 Stack Pointer to its last known value; if the CPU is about to be released from RESET, the Stack Pointer is set to 0x07ff which is the end of user RAM, else it is set to the value from the previous execution. The program also includes an unconditional JMP that makes the 8085 proceed to the start of the user's program and begin executing it.

```
LXI SP, [addr]
JMP [user_prog]
```

Snippet 3.5 – Initialization program

The memory address where this initialization program is loaded depends on the CPU's state. If the 8085 is still in RESET, the program is loaded at the reset vector whereas if the 8085 is in the HALT state, the program is loaded at the RST 7.5 vector. As mentioned earlier, an RST 7.5 hardware interrupt, in the form of a rising edge, is used to 'wake' the 8085 from the HALT state; it pushes the Program Counter contents onto the stack and proceeds to the RST 7.5 restart address. This is the reason why any initialization program must be placed at the RST 7.5 vector, if the CPU is to be released from the HALT state.

The second program handles the post-execution process. It is responsible for extracting the contents of the CPU registers, enabling the RST 7.5 interrupt in preparation for the next user program execution, and eventually HALTing the CPU.

An effort is made to ensure that all register contents presented to the user are exactly as expected, assuming the CPU executed the user program alone. When the next program executes, the contents of the registers must also be the same as they were immediately after the previous user program execution. The only exception is the PC whose contents are irrelevant to the next user program since the user will provide a new start address for the GO command anyways.

In the second program, the contents of the HL register pair are first saved to RAM with the *SHLD* instruction and then a memory address in the monitor RAM is loaded into the register pair. The contents of the general-purpose registers are obtained with successive *'MOV M, R'* instructions that transfer them to RAM; the address in the HL pair is incremented after each transfer.

```
SHLD 30H
PUSH PSW
LXI HL, 32H
MOV M, E;    INX HL
MOV M, D;    INX HL
MOV M, C;    INX HL
MOV M, B;    INX HL
MOV M, A
LXI H, 00H
DAD SP
SHLD 37H
LHLD 30H
POP PSW
EI;      MVI 0BH;     SIM
HLT
```

Snippet 3.6 – Post-execution program

The process of obtaining the Stack Pointer's contents provide access to the contents of the remaining flag register and Program Counter. To obtain the SP's value, the HL register pair is loaded with 0x00 and a '*DAD SP'* instruction is executed, which adds the Stack Pointer contents to the HL pair contents and stores the result in the HL pair. This has the effect of storing the SP's contents in the HL register pair, from where it can be easily exported to RAM with the *SHLD* instruction. A *PUSH PSW* is used to extract the flag register contents and a *POP PSW* is then executed to counter the PUSH. The *LHLD* instruction is used to restore the contents of the HL register pair from their saved location. Finally, interrupts are enabled with *EI* and the interrupt masks are set with *SIM* and the 8085 is HALTed to await the next user program or a RESET.

This post-execution program is loaded at the RST 1 address; this address is fixed and independent of the CPU's state. Therefore, in order for this program to be executed immediately after the user program as desired, the user must terminate their program with the RST 1 instruction i.e. 0xCF must be the last byte in any user program so that the CPU branches to the RST 1 address without any interference. Before branching to this vector, the CPU places the PC contents on the stack. This means that when the SP contents are deduced in the post-execution program, the PC contents can be easily gotten since they lie just below the flag register contents in memory, after the latter have been obtained with the post-execution PUSH/POP.

A C function in the monitor handles the aggregation of all the register contents into two arrays – one for the 8-bit registers and another for the 16-bit registers. When the user presses the EXAM_REG button, the monitor presents the contents of each register from the array, advancing with each press of the NEXT button. The allowable inputs here are the SHIFT, NEXT, EXEC, RESET buttons.

EXEC terminates the examination of the register contents and returns the monitor to the ENTERING_IDLE state.

Note that HEX_KEY above refers to any of the hexadecimal digits 0 – F on the keypad. Fig. 3.7 on the next page illustrates the entire state machine of the monitor. The labels along the transition lines represent the state inputs, whereas the label within each circle represent the name of that state. For instance, while in the IDLE state, an input of *exam_reg* causes a transition from the IDLE state to the EXAM_REG state. In almost all cases, the state inputs are actually button presses from the user. The only exceptions are: the transition from ENTERING_IDLE to IDLE_STATE which occurs automatically and the transition from WAITING_MPU to ENTERING_IDLE which is dependent on when the 8085 encounters a HLT instruction or is reset by the user. The double-circle around ENTERING_IDLE indicate it is the reset state of the system; all button presses of RESET from any other state lead back to ENTERING_IDLE.

3.6 – CHAPTER SUMMARY

This chapter detailed the factors considered in designing the hardware and software of the Intel 8085 microprocessor trainer. It presented the hardware design process all the way to the printed circuit board design and also explained the thought process in producing the software and its overall architecture.

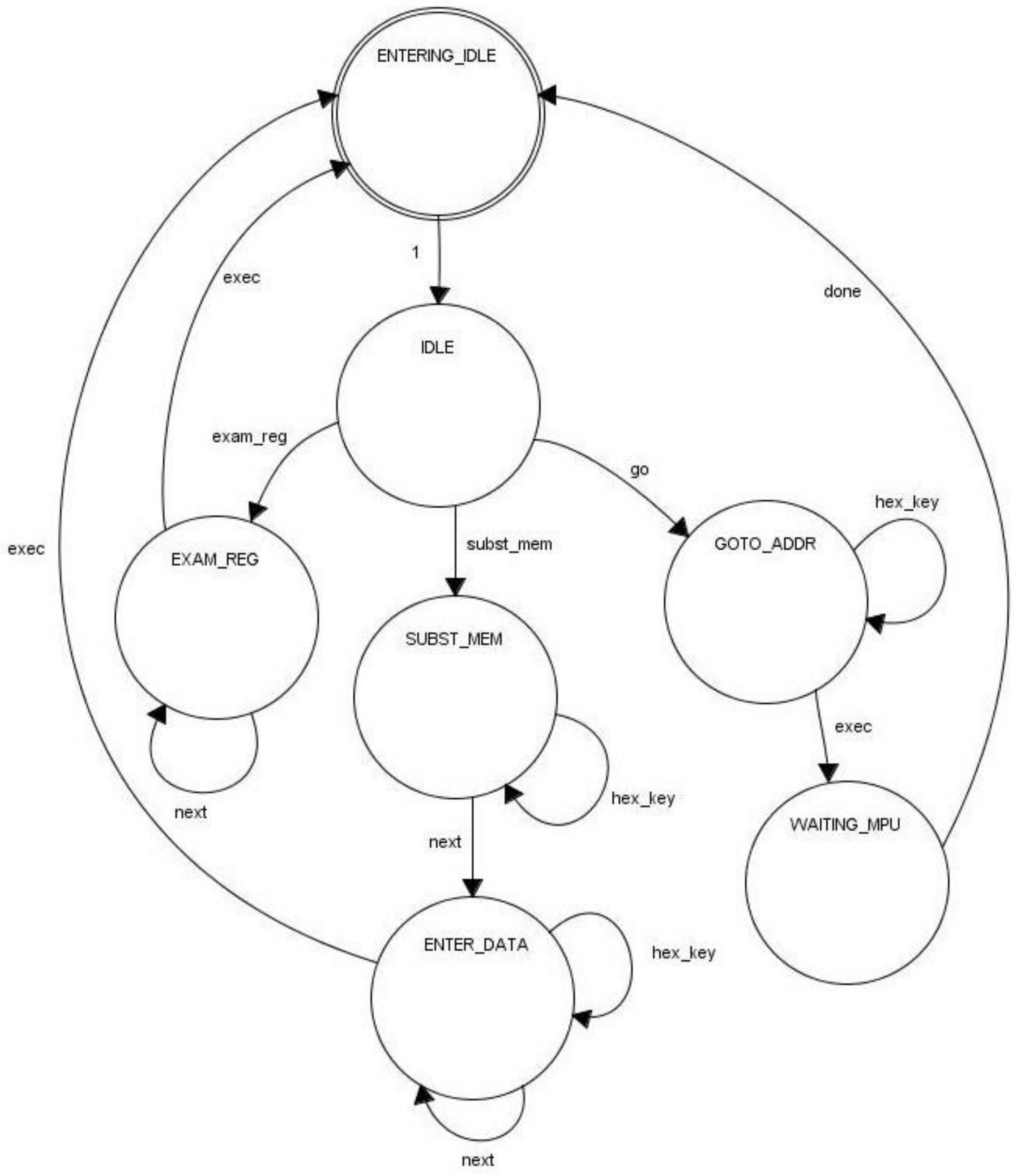


Fig. 3.7 – Overall FSM of the microprocessor trainer monitor

CHAPTER 4

RESULTS AND DISCUSSION

4.1 – UNIT TESTING

This section presents the unit and integration testing performed on the constructed prototypes as well as the results obtained from the tests.

Most of the tests performed were centred on the PIC18F4520 monitor. The hardware UART debug port of the PIC was invaluable in displaying the results of tests. A PICkit 2 served not only as a programmer but also to supply power to the test circuits and as a USB-UART converter through the 'UART tool' in the PICkit 2 application. This enabled easy programming of the monitor and subsequent testing.

The tests in this section were all performed on a large platform which held 4 breadboards arranged side by side.

- **Keypad test:**

The PIC18F4520 monitor was interfaced with the GPIO expander and 4x5 keypad on a breadboard to test input capability. This was easily achieved especially since the keypad is a membrane keypad and so did not require much in the way of debouncing.

- **Display test:**

The monitor was also interfaced successfully with the TFT LCD via 'bit-banged' SPI. The ported Adafruit drivers made this task considerably easier. Therefore, at this stage, input and display for the monitor were functional.

- **SRAM test:**

The monitor was then interfaced with the SRAM and a program was written to test the memory. The test program basically wrote a unique byte to each of the first 256 addresses and then read back the data from the memory. Both the written and read bytes for each address were printed in two columns to the debug port and displayed in the 'UART Tool' of the PICkit 2 application. The bytes were then compared visually to ensure that they were the same. These tests were successful; the same values that were written to the SRAM were also read from it.

- **8085 free run test:**

Next, a free run test was performed with the Intel 8085. This test is intended to find out if the 8085 is functional; one is expected to observe the 8085 as it runs through its address space. The interrupt pins are connected to GND. The HOLD and READY pins are connected to GND and V_{CC} respectively. The data bus was tied to GND through 10k pull-up resistors so that the 8085 seems to fetch only NOP instructions from memory. A 4 MHz crystal was used to generate the 8085 clock signal for this test. The A15 and A14 pins were connected to the PGD and PGC pins of the PICkit 2 programmer, now serving as a logic analyzer. The signals on both pins were viewed in the PICkit 2 application and a rectangular pulse train was observed on each pin with the frequency of A14's signal twice that of A15. This is sufficient proof that the 8085 was functional.

4.2 – INTEGRATION TESTING

- **Integration on a breadboard**:

Then the 8085 was connected to the existing A/D bus of the monitor and the SRAM test was executed again. This time, the test was not successful. Wrong values were either written to or read from the SRAM, resulting in different values in the byte columns printed through the serial port.

Since an oscilloscope was unavailable, the cause of this issue could not be determined with exactitude. However, it is most likely due to a combination of different factors such as:

- Long DuPont cables and breadboard conductors which introduce parasitic capacitance that distorts the parallel bus signals,
- Crosstalk between so many wires with fast-moving signals clustered together. Other possibilities include weak connections or wiring mistakes.

Whatever the cause, at this point, it was decided that it would be best to design a PCB which was expected to resolve these problems.

- **Integration tests on a Vero board**

The monitor portion of the circuit was assembled on a Vero board in order to provide proof-of-concept for the software design. This test involved the PIC18F4520, the keypad and I/O expander and the TFT LCD. The state machine was implemented and uploaded to the monitor. The test was successful. The state machine was implemented perfectly and the monitor stepped through each state based on user input. A dummy 8085 program was entered into the system and was 'executed'. The CPU registers could be examined and RAM could be modified. These are all emulated by the monitor test program.

Fig. 4.1 shows the test circuit running on 5 V supplied by a USB-connected Arduino.

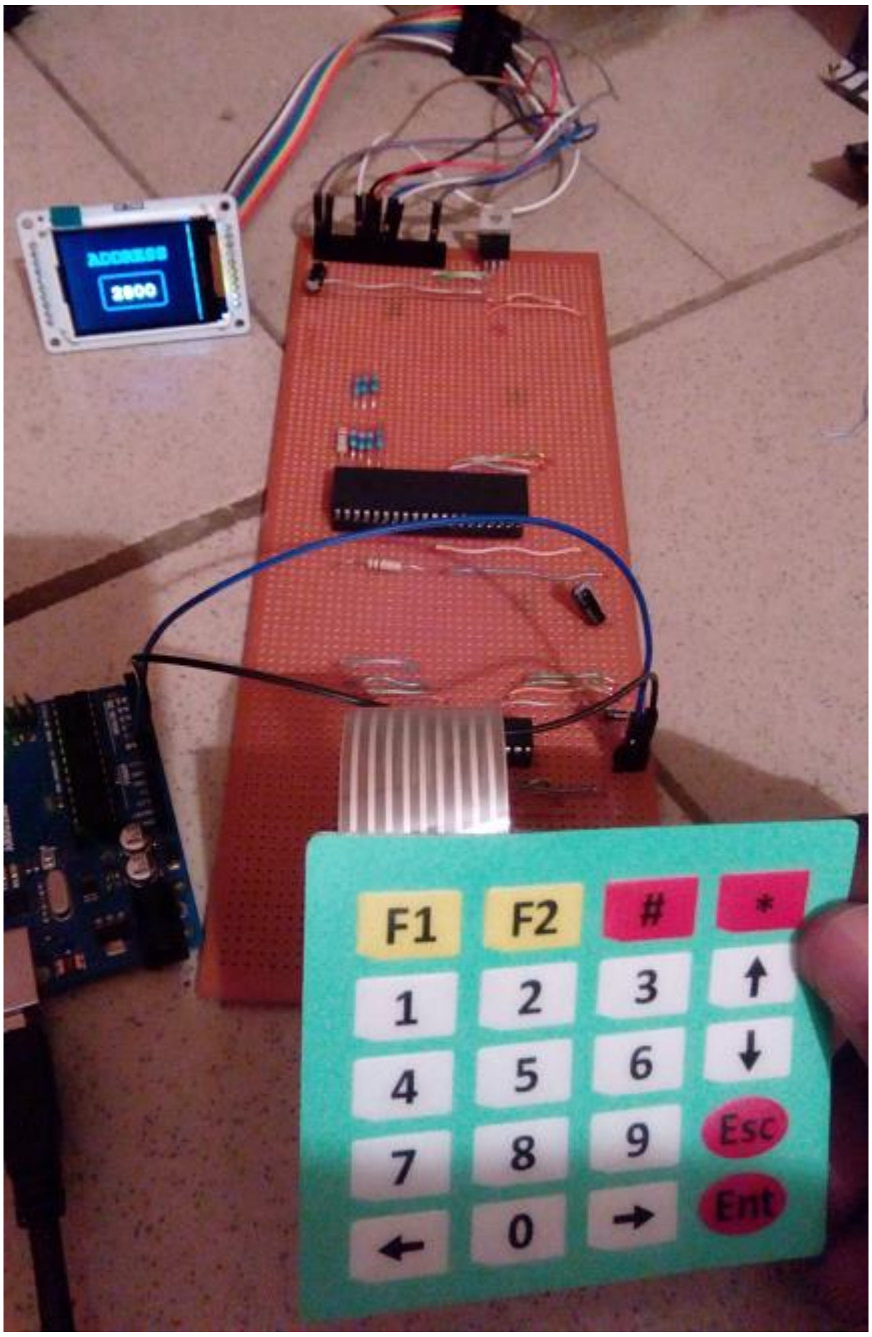


Fig. 4.1 – Monitor FSM running on a Vero board

The circuit was also tested with a battery pack of 3 AA batteries and worked perfectly.

- **Integration tests on a PCB**

The fabricated but unassembled PCB of the entire microprocessor trainer is shown in Fig. 4.2.

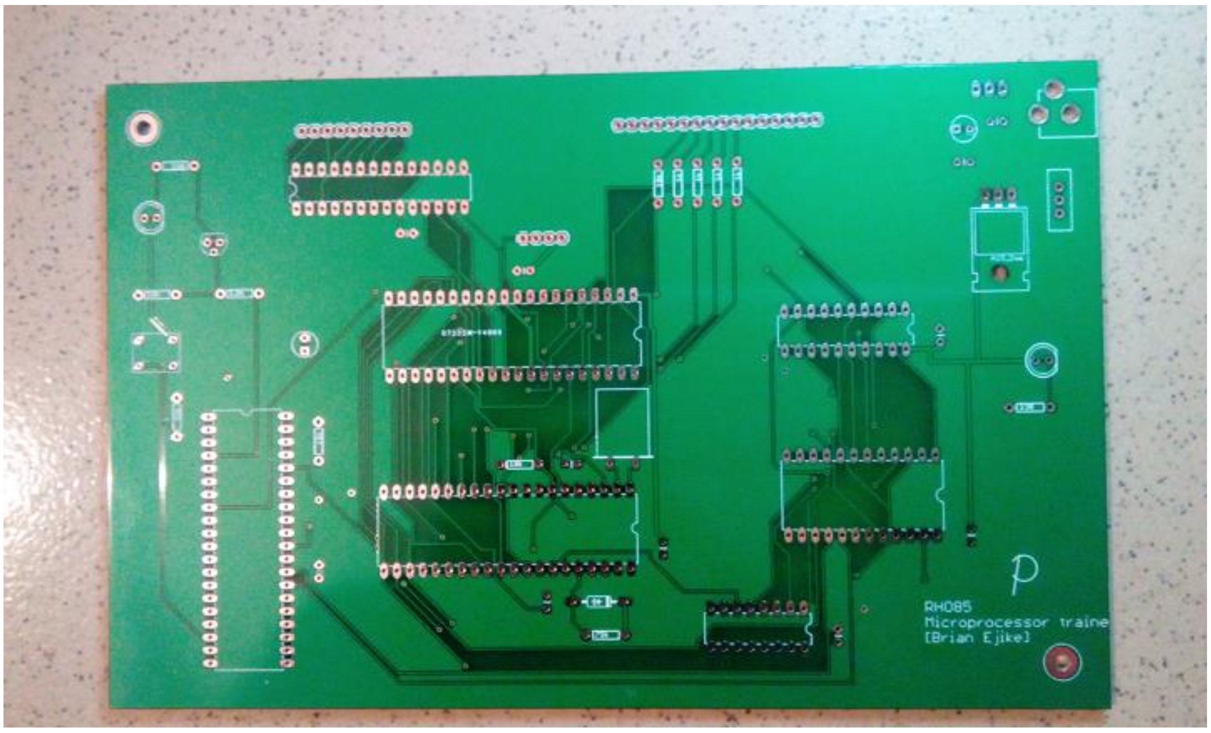


Fig. 4.2 – The fabricated microprocessor trainer PCB

The PCB was assembled and soldered without issues. Most of the iterative process of firmware development involving the 8085 occurred at this stage. Because an ICSP header was omitted in the PCB design, the PIC monitor had to be unplugged from the PCB, programmed on a breadboard and plugged back into the PCB.

The fabricated PCB was slightly flawed. When designing the schematic, two ICs were inadvertently excluded from the data bus shared by the rest of the system. The chips, SRAM and PPI, were accidentally omitted in a change of net names for the data bus and this was reflected in the PCB design. This mistake was only discovered after the PCB had been assembled and attempts to perform R/W operations on the SRAM failed inexplicably. To fix the problem, jumpers were soldered beneath the board, from the SRAM to the octal latch, to link the omitted chips with the rest of the system data bus. The schematic in the Appendices reflects this correction, though the PCB design is still the same one that was fabricated.

This workaround presented significant problems however. The wires were long and springy. Insecure solder joints had to be held in place with hot glue, marring the surface of the board further. Crosstalk was the main obstacle though. Bit D5 on the data bus was particularly susceptible, going HIGH if enough bits around it are also HIGH. Eventually, the D5 wire was soldered along a separate path, from the PPI to the 8085, in order to fix the crosstalk issue. However, a twisted-pair approach, coupled with short wiring over continuous ground pours, would have prevented these issues, if there had been sufficient foresight or PCB design experience beforehand.

The original TFT display for which the board was designed got damaged before the PCB arrived. The only readily available TFT LCD had a different pin-out and so did not fit into the existing SIL-16 header on the PCB. It had to be connected to the header with jumpers. Also, this display required a 5 V supply, as there was already a 3.3 V regulator built-in with the LCD. This means the LM1117T on the designed PCB was no longer needed and so its output was shorted to its input to supply the LCD with 5 V. Some of the resistor values on the PCB were changed during assembly, as well.

Eventually, with the system working as expected, an enclosure was designed for the trainer. Acrylic was chosen because it is easily workable and generally aesthetically pleasing. Figs. 4.3 and 4.4 show the completely assembled trainer in its enclosure, powered by 4 AA batteries in a battery pack.

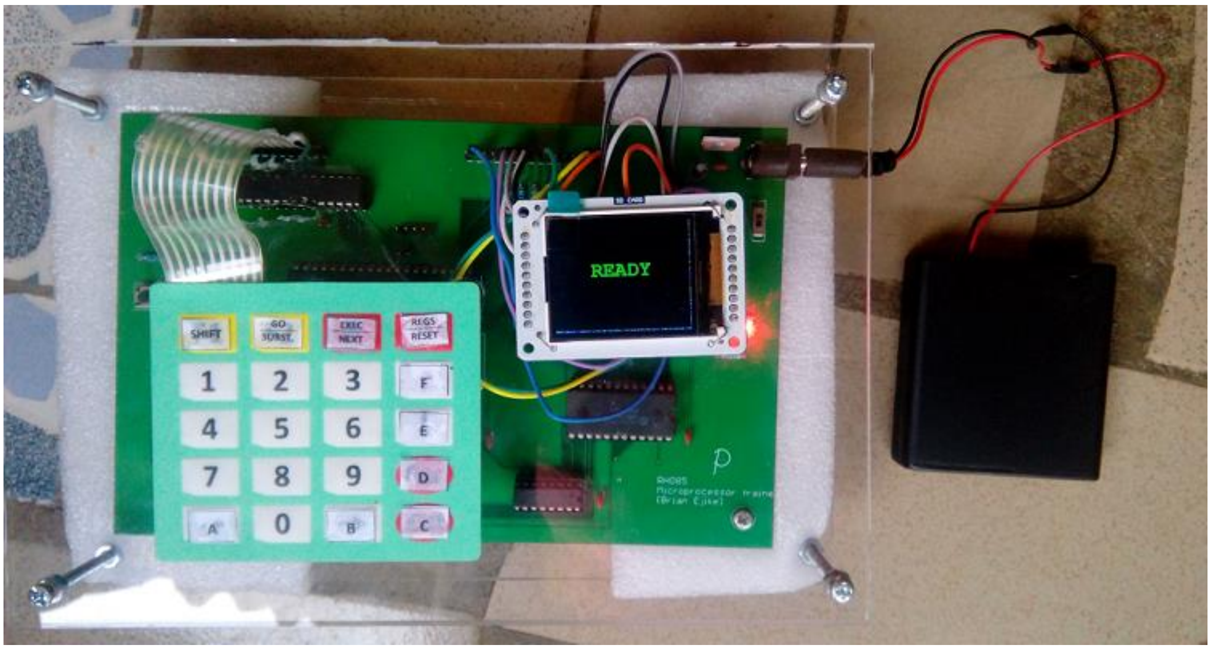


Fig. 4.3 – Completely assembled Intel 8085 trainer (top)

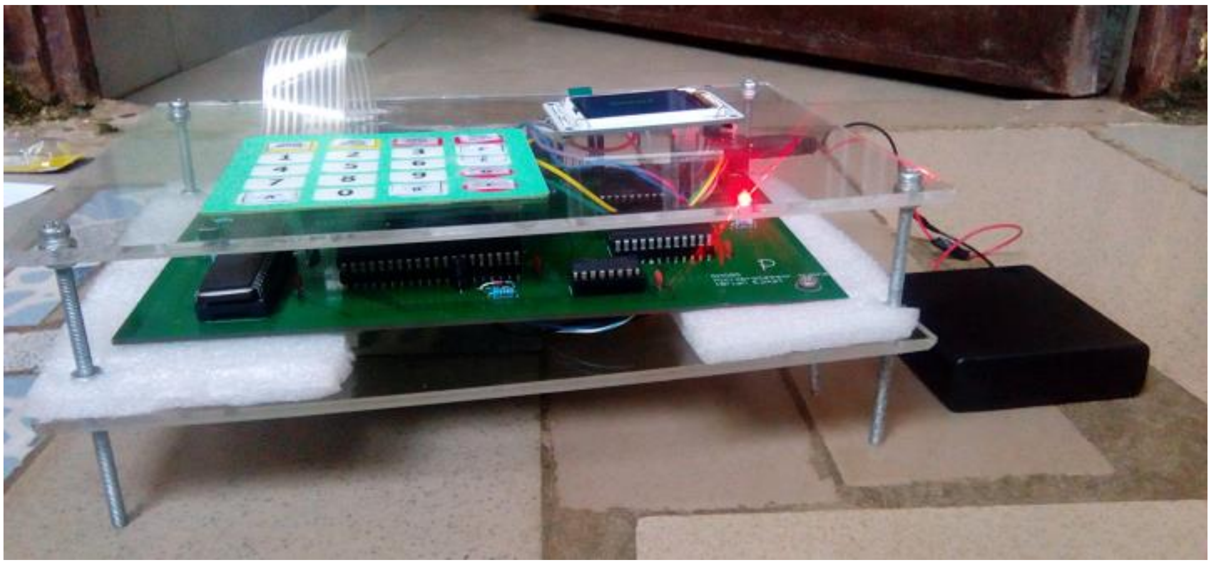


Fig. 4.4 – Completely assembled Intel 8085 trainer (side)

Current measurements were taken from start-up through user program execution till the trainer was turned off. The current drawn by the trainer fluctuated between 190 mA and 220 mA. With the LED on, the current rises by 8 mA. The LDO regulator kept the voltage output reasonably stable at 5V.

4.3 – SUMMARY OF THE PROECT RESULTS

Below is a summary of the results of the project in terms of its objectives:

- **Cost Minimization**

The foremost goal for this project was the minimization of the cost of obtaining an Intel 8085 microprocessor trainer. On the one hand, the price for a common Intel 8085 trainer manufactured in India is about 4400 INR [15] which is roughly equivalent to 21000 NGN, using the current exchange rate. On the other hand, the cost of the fabricated PCBs for this project was 22 USD; 5 boards were fabricated, equivalent to a unit cost of 4.4 USD. Many of the components used for this project could not be sourced locally within Nigeria. However, they may be bought cheaply on Aliexpress for under 30 USD.

Table 5.1 – Cost of the electronic components used for this project

Item	**Unit cost in USD**
P8085AH	0.78
HM6116	0.89
PIC18F4520	2.80
Intel 8255A	3.50
1.8” TFT LCD	4.65
4x5 matrix keypad	5
Misc.	10 USD

Table 5.1 gives the cost of the components used in this project, based on Aliexpress.com prices. Of course, some of these components cannot be bought in single quantities but must be bought in bulk. Yet, the total cost of the

components and the PCB seem to be about 35 USD which is roughly equivalent to 11200 NGN – almost half the price of the trainer from India.

One could argue that the trainer from India has a lot more features and is better designed than the trainer designed in this project and one would be right. Yet, the aim of the project is to develop a trainer with the minimal capabilities needed for students to learn microprocessor architecture and programming. Even with the existing trainers in the lab, only their basic features are employed in practical sessions, so it is more sensible to manufacture trainers with only the features that will definitely be used. In future, perhaps these additional capabilities may be added to further educate students. But it is clear that this project accomplished its primary goal of constructing a relatively cheap Intel 8085 trainer.

- **Functionality**

The trainer faithfully accepts user programs, executes them and allows inspection of memory and the CPU registers. Test programs to add 2 numbers and store the result in RAM and also to blink the on-board LED were written to verify that the 8085 is functional and the trainer works as expected. Thus, the fundamental features of a trainer were implemented. A switch and an LED were also included to act as physical I/O peripherals for users.

- **User interface and compatibility**

The trainer constructed in this project further improved on the expressiveness of existing trainers, provided a more modern-looking display and was completely compatible, in terms of commands, with the existing trainers in the lab.

- **Power supply and consumption**

An AA battery pack or the DC output of an AC adapter may be used to power the constructed trainer. This liberates students from the yoke of being subject to the whims of the national power distribution system and from having to buy gasoline for every practical session. Anyone can come with their batteries, use the trainer and, when they are done, depart with the batteries. A 4-pin header was also provided so that power may be supplied to the circuit without plugging into the jack.

At worst, the trainer consumes power of 0.23A × 5V = 1.15 W and energy in one hour equivalent to 1.15Wh. The amount of power dissipated within the regulator at any point depends on the voltage at that point and reduces with input voltage. A typical combination of 4 fresh AA batteries each with a capacity of 0.8Ah and voltage 1.7 V would have stored energy approximately equivalent to 1.7 × 4 × 0.8 = 5.44Wh. However, these batteries would only remain useful so long as their combined voltage is above the sum of 5 V and the dropout voltage of the on-board 5V regulator. The LM2940-N has a drop-out of about 0.15 V [16] when current output is 0.23A. Therefore, neglecting cell internal resistance, the usable energy is ((1.7 × 4) – 5.15) × 0.8 = 1.32Wh. Considering regulator power dissipation, if the worst input-output voltage difference and current are assumed, the dissipated power is ((1.7 × 4) – 5) × 0.23 = 0.414 W and if this is assumed constant for an hour, the wasted energy would be 0.414Wh. Subtracting this from the usable battery energy, the result is 1.32 – 0.414 = 0.906Wh which may be used by the trainer. Therefore, the trainer may be used continuously for at least 0.906Wh ÷ 1.15W ≈ 0.79 hours ≈ 47 minutes before the batteries are drained. This is long enough for any single practical session, especially if the trainer is turned off when not in use.

4.4 – PROBLEMS AND DISCUSSION

The 8085 CPU gradually heats up as soon as power is applied to the circuit. No short-circuits were discovered and the trainer performs as expected most times, though a certain program involving an endless loop returned not long after its execution began. Due to time constraints, the cause of this problem could not be narrowed down. Any further tests in future research will require new PCBs wherein the current PCB issues have been fixed in order to eliminate them as potential problem sources.

Since the current monitor RAM block encompasses the hardware and software interrupt vectors, users are deprived of interrupt capabilities in the programs they execute. Students should have practical knowledge of how CPUs handle interrupts so a future improvement could involve the re-allocation of monitor RAM to some block perhaps near the end of memory, rather than at the start. Access to a hardware interrupt for users is another possible improvement, in keeping with the features already offered by current trainers.

The major drawback of using a separate hardware monitor is the additional connections that must be made between the monitor and the 8085 buses in order for it to succeed as a monitor. If single-stepping is to be eventually implemented, 8085 bus operations have to be intercepted every instruction cycle in order for the PIC to follow the execution process. This requires delicate timing to, for instance, track each falling edge of the ALE signal to obtain the addresses placed by the 8085 on the bus. If external interrupts are used for the PIC to register events, the interrupt latency must be low enough and the ISR must be as brief as possible, else events will be missed.

Other issues include the extra power consumed by the separate monitor, as well as the additional line capacitance and inductance introduced by connecting one more device to the system buses. However, these demerits are surmountable and well balanced by the ease of program development and maintenance for an MCU monitor.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 – CONSTRAINTS AND CONCLUSION

The major constraint in this project was the time wasted in waiting for electronic components to arrive from China. Due to this unavoidable delay, a lot of time, perhaps too much time, was spent on carefully selecting the components that would be needed in order to prevent any mistakes. This is coupled with the time taken to negotiate with sellers, place an order and then wait for the arrival of the items. In all, at least a month, and usually more, is spent from concept to arrival of the items. This greatly delayed firmware development. Despite all this, the project was completed with moderate success and it achieved its objectives with few issues. The trainer's current draw is not as low as hoped but is still within bounds and sustainable for a reasonable amount of time on cheap batteries, as intended.

5.2 – RECOMMENDATIONS

If the fabrication of these trainers is taken up by the department, it is recommended that all PCB assembly and firmware development be done by students under the supervision of lecturers and technologists. Thus, not only would cheap trainers be manufactured for the use of students but also the students will learn a great deal about electronics in general during development process. Hopefully, this should improve the quality of ECE engineers produced by this university and help strengthen the nation's technological stance in the long run.

REFERENCES

[1] A. K. Maini, Digital Electronics, West Sussex: John Wiley & Sons Ltd, 2007.

[2] Future Electronics, "Microcontroller and Microprocessor," Future Electronics, [Online]. Available: https://www.futureelectronics.com/en/microcontroller-microprocessor/microcontroller-microprocessor.aspx. [Accessed 1 October 2016].

[3] "KIM-1 Computer," 2002. [Online]. Available: http://oldcomputers.net/kim1.html. [Accessed 1 October 2016].

[4] Intel Corporation, *MCS-85 System Design Kit manual,* California: Intel Corporation, 1978.

[5] The Glitch Works, "Building an 8085 Single Board Computer," 2 September 2010. [Online]. Available: http://www.glitchwrks.com/2010/09/02/8085-sbc. [Accessed 9 October 2016].

[6] M. Graybill, "MAG-85: An 8085 CPU Microprocessor Project," 2010. [Online]. Available: http://saundby.com/electronics/8085/. [Accessed 11 October 2016].

[7] Intel Corporation, *Intel 8085AH Datasheet,* 1993.

[8] M. Horvath, "Tutorial on Introduction to 8085 Architecture and Programming," [Online]. Available: uni-obuda.hu/users/horvath.mark/kando/interfeszek/8085/8085general.pdf. [Accessed 15 October 2016].

[9] Grade Stack, "Classification of Interrupts," [Online]. Available: http://gradestack.com/Microprocessors-and/Interrupts-of-8085/Classification-Of/19314-3912-38133-study-wtw. [Accessed 16 October 2016].

[10] EDITS WORLD, "RISC and CISC," 2014. [Online]. Available: http://www.editsworld.com/articles/risc-and-cisc/. [Accessed 16 October 2016].

[11] "Instruction Set and Format," [Online]. Available: http://vle.du.ac.in/mod/book/print.php?id=13334&chapterid=29357. [Accessed 16 October 2016].

[12] I. Poole, "EEPROM Technology," Adrio Communications Ltd, [Online]. Available: http://www.radio-electronics.com/info/data/semicond/memory/eeprom-basics-tutorial.php. [Accessed 16 October 2016].

[13] Intel Corporation, *Intel 8255A Programmable Peripheral Interface,* 1993.

[14] Microchip Technology Inc., *PIC18F2420/2520/4420/4520 Datasheet,* Microchip Technology Inc., 2004.

[15] RBA Engineering, "Price List," [Online]. Available: http://rbaengineering.com/eprice.asp. [Accessed 20 October 2016].

[16] Texas Instruments, "LM2940x 1-A Low Dropout Regulator (Rev. J)," 2016. [Online]. Available: www.ti.com/lit/ds/symlink/lm2940c.pdf. [Accessed 11 November 2016].

APPENDIX A – TECHNICAL DATA

The schematic (corrected), PCB design (flawed version), monitor code and drivers and all relevant documentation are hosted online at https://github.com/brianrho/Rho85

APPENDIX B – SAMPLE 8085 PROGRAM

This sample program loads two numbers (0x6E and 0x64) into registers A and B, adds them and stores the result in register A as well as at an address in RAM.

```
MVI A, 6Eh
MVI B, 64h
ADD B
STA 0150h
RST 1
```

In assembly

```
3E
6E
06
64
80
32
50
01
CF
```

In machine code

The machine code can be entered directly into the trainer starting from any address in user RAM and executed. Address 0x150 and the registers can then be examined to see if they contain the expected values:

```
0150h: D2

A: D2
B: 64
C: unknown
D: unknown
E: unknown
H: unknown
L: unknown
Sign, Parity and Auxiliary Carry flags are set
PC: [Starting address + 8]
SP: 07ff
```