Review of Assembly Language Programming

Kritish Pahi[#1], Kshitiz Shrestha[#2], Manish Chandra[#3], Manish Munikar[#4]

Department of Electronics and Computer Engineering, IOE, Central Campus, Pulchowk

Pulchowk, Lalitpur, Nepal

[1]kritishpahi@gmail.com

[2]xitiz.me78@gmail.com

[3]manishkarn02@gmail.com

[4]munikarmanish@gmail.com

***Abstract*—Basic introduction to assembly language programming, which the programmer must be familiar with, are presented. The report contains information regarding assembly language, assembler, assembly language format, instruction set of 8085, stack and subroutine. Moreover, the 8085 trainer kit and the way of writing and executing the programs in it, are also presented. The primary focus here is on the microprocessor because the microprocessor determines the machine language and the operations of a microprocessor-based system.**

I. Introduction

Before writing any assembly language program, a programmer must be familiar with some of the common terminologies, such as: machine language, assembly language, assembler, instruction set, etc.

A. Basic Terms

1) Microprocessor: A microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions, and provides results as output. The microprocessor can be embedded in a larger system, can be a stand-alone unit controlling processes, or it can function as the CPU of a computer called microcomputer. At a very elementary level, we can draw an analogy between microprocessor operations and the functions of a human brain that process information according to instructions (understanding) stored in its memory. The brain gets input from eyes and ears and sends processed information to output "devices" such as the face with its capacity to register expression, the hands or feet. However, there is no comparison between the complexity of a human brain and its memory and the relative simplicity of a microprocessor and its memory. A set of instructions written for the microprocessor to perform a certain task, is called a program. The microprocessor applications are classified primarily in two categories: reprogrammable systems and embedded systems. In reprogrammable systems, such as microcomputers, the microprocessor is used for computing and data processing. In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to the end user. Specifically speaking, here, we are concerned with the language of a widely used 8-bit microprocessor, the 8085, manufactured by Intel Corporation. The 8085 is a microprocessor with 8-bit word length: its instruction set is designed by using various combinations of these eight bits. A simple block diagram of a programmable machine is shown in Figure 1.

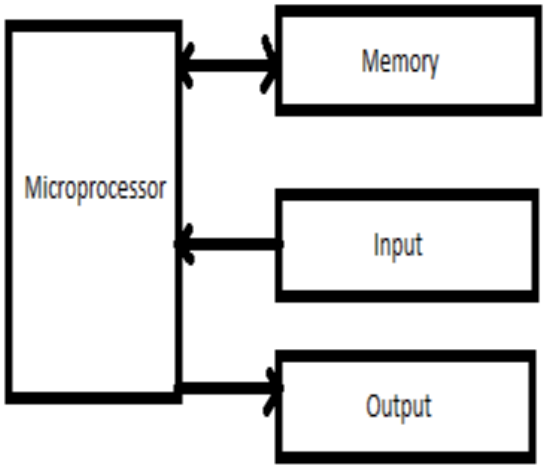

Fig. 1. A programmable machine

2) Machine Language: The microprocessor communicates and operates in the binary numbers 0 and 1, called bits. Each microprocessor has a fixed set of instructions in the form of binary patterns called a machine language. The number of bits in a word for a given machine is fixed, and words are formed through various combinations of these bits. For example, a machine with a word length of eight bits can have 256 (2^8) combinations of eight bits—thus a language of 256 words. However, not all of these words need to be used in the machine. The microprocessor design engineer selects combinations of bit patterns and gives a specific meaning to each combination by using electronic logic circuits; this is called an instruction. Instructions are made up of one word or several words. The set of instructions designed into the machine makes up its machine language- a binary language, composed of 0s and 1s- that is specific to each computer. The 8085 microprocessor has 246 such bit patterns, amounting to 74 different instructions for performing various operations. These 74 different instructions are called its instruction set. This binary language with a predetermined instruction set is called the 8085 machine language.

3) Assembly Language: It is difficult for humans to communicate in the language of 0s and 1s. Even though the instructions can be written in hexadecimal code, it is still difficult to understand a program written in hexadecimal numbers. Therefore, each manufacturer of a microprocessor has devised a symbolic code for each instruction, called a mnemonic. The complete set of 8085 mnemonics is called the 8085 assembly language, and a program written in these mnemonics is called an assembly language program. The mnemonic for a particular instruction consists of letters that suggest the operation to be performed by that instruction. Although these symbols do not specify the complete operations, they suggest its significant part. An assembly language program written for one microprocessor is not transferrable to a computer with another microprocessor unless the two microprocessors are compatible in their machine codes. For example, the binary code 0011 1100 ($3C_{16}$) of the 8085 microprocessor is represented by the mnemonic INR A. INR stands for increment, and A represents the accumulator. This symbol suggests the operation of incrementing the accumulator content by one.

Machine language and assembly language are microprocessor-specific and are both considered low-level languages. The machine language is in binary, and the assembly language is in English-like words; however the microprocessor understands only the binary.

4) Assembly Language Format: A typical assembly language programming statement is divided into four parts called fields: label, operation code (opcode), operand and comments. These fields are separated by delimiters as shown in table I.

TABLE I
TYPICAL DELIMITERS USED IN ASSEMBLER STATEMENTS

Delimiter	**Placement**
Colon	After label (optional)
Space	Between an opcode and an operand
Comma	Between two operands
Semicolon	Before the beginning of a comment

The assembler statements have a free-field format, which means that any number of blanks can be left between the fields. Comments are optional but are generally included for good documentation. Similarly, a label for an instruction is also optional, but its use greatly facilitates specifying jump locations. As an example, a typical assembly language statement is written as follows:

```
Label    Opcode    Operand     Comments
START:   LXI       SP, 20FFH   ;Initialize stack pointer
```

5) Assembler: The mnemonics can be written by hand on paper and translated manually in hexadecimal code, called hand assembly. Similarly, the mnemonics can be written electronically on a computer using a program called an Editor in the ASCII code and translated into binary code by using the program called an assembler. Hence, the assembler is a program that translates the mnemonics entered by the ASCII keyboard into the corresponding binary machine codes of the microprocessor. Each microprocessor has its own assembler because the mnemonics and machine codes are specific to the microprocessor being used, and each assembler has rules that must be followed by the programmer.

A computer is a binary machine; to communicate with the computer in alphabetic letters and decimal numbers, translation codes are necessary. The commonly used code is known as ASCII—American Standard Code for Information Interchange. It is a 7-bit code with 128 (2^7) combinations, and each combination from 00H to 7FH is assigned to either a letter, a decimal number, a symbol or a machine command.

6) Cross-Assembler: A cross-assembler is a program that can be used to translate 8085 mnemonics by a computer that has a microprocessor other than the 8085. The format of a cross-assembler is almost identical to that of an assembler with a few variations. Personal Computers are commonly available these days. These computers are based on 16- or 32-bit microprocessors with different mnemonics than the 8085 microprocessor. However, cross-assemblers can be used to translate the 8085 mnemonics into appropriate machine codes.

B. The 8085 Instruction Set

The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. The set includes all the 8080A instructions plus two additional instructions (SIM and RIM, related to serial I/O). The 8085 instruction set is classified into five different groups: data transfer, arithmetic, logic, branch and machine control.

1) Data Transfer (Copy) Instructions: These instructions perform the following six operations:

- Load an 8-bit number in a register
- Load 16-bit number in a register pair
- Copy from register to register
- Copy between register and memory
- Copy between I/O and accumulator
- Copy between registers and stack memory

Here, the contents of the source are not transferred, but are copied into the destination register without modifying the contents of the source.

2) Arithmetic Instructions: The frequently used arithmetic operations are:

- Addition
- Subtraction
- Increment
- Decrement

Here, addition and subtraction are performed in relation to the contents of the accumulator.

3) Logic and Bit Manipulation Instructions: These instructions include the following operations:

- AND
- OR

- X-OR (Exclusive OR)
- Compare
- Rotate bits

All logic operations are performed in relation to the contents of the accumulator.

4) Branch Instructions: These instructions allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions. These instructions are the key to the flexibility and versatility of a computer.

5) Machine Control Instructions: These instructions affect the operation of the processor.

- HLT
- NOP

C. Stack and Subroutines

1) Stack: The stack is a group of memory locations in the R/W memory that is used for temporary storage of binary information during the execution of a program. The starting memory location of the stack is defined in the main program, and space is reserved, usually at the high end of the memory map. The stack is shared by the programmer and the microprocessor. Specifically speaking, the stack in an 8085 microcomputer system can be described as a set of memory locations in the R/W memory, specified by a programmer in a main program. These memory locations are used to store binary information (bytes) temporarily during the execution of a program. As a general practice, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The programmer can store and retrieve the contents of a register pair by using PUSH and POP instructions. Similarly, the microprocessor automatically stores the contents of the program counter when a subroutine is called. Hence, read/write memory generally is used for three different purposes:

- to store programs or instructions
- to store data
- to store information temporarily in defined memory locations called the stack, during the execution of the program

2) Subroutine: A subroutine is a group of instructions that performs a subtask (e.g., time delay) of repeated occurrence. For example, if a time delay is required between three successive events, three delays can be written in the main program. To avoid repetition of the same delay instructions, the subroutine technique is used. Delay instructions are written once, separately from the main program, and are called by the main program when needed. Specifically speaking, the 8085 microprocessor has two instructions to implement subroutines: CALL and RET. The CALL instruction is used in the main program to call a subroutine, and the RET instruction is used at the end of the subroutine to return to the main program. When a subroutine is called, the contents of the program counter, which is the address of the instruction following the CALL instruction, is stored on the stack and the program execution is transferred to the subroutine address. When the RET instruction is executed at the end of the subroutine, the memory address stored on the stack is retrieved, and the sequence of execution is resumed in the main program. The subroutine is written as a separate unit, apart from the main program, and the microprocessor transfers the program execution from the main memory to the subroutine whenever it is called to perform the task. After the completion of the subroutine task, the microprocessor returns to the main program. Before implementing the subroutine technique, the stack must be defined; the stack is used to store the memory address of the instruction in the main program that follows the subroutine call.

II. THE 8085 TRAINER KIT

A. Basic Introduction

The 8085 trainer kit is a board, where one can enter machine code by using hex keys and can execute it. The kit has EPROM, 8085 microprocessor, RAM, battery, hex keyboard, seven-segment display (optional), output LED ports, registers, capacitors, etc. Because it is tedious and error-inducive for people to recognize and write instructions in binary language, these instructions are, for convenience, written in hexadecimal code and entered in a single-board microcomputer by using Hex keys. For example, the binary instruction 0011 1100 is equivalent to 3C in hexadecimal. This instruction can be entered in a single-board microcomputer system with a Hex keyboard by pressing two keys: 3 and C. The monitor program of the system translates these keys into their equivalent binary pattern. The 8085 trainer kit is shown in figure 2.

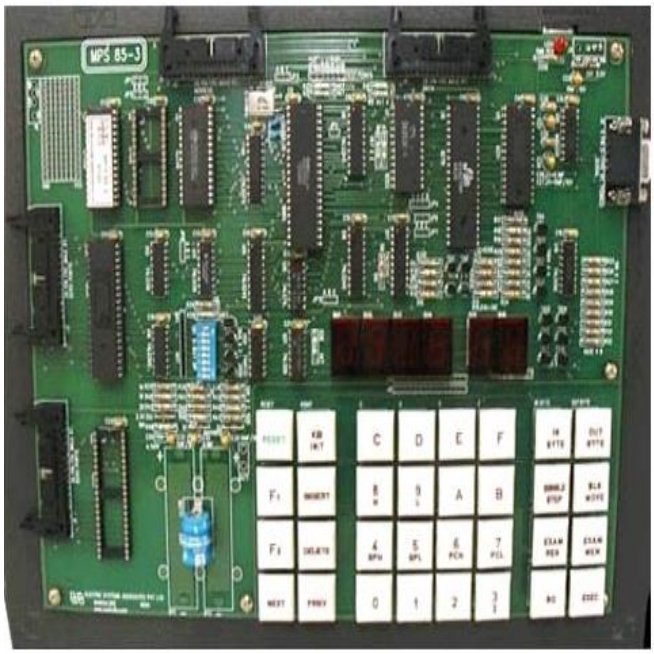

Fig. 2. The 8085 trainer kit

B. Storing the Program in Memory

To store the program in R/W memory of a single-board microprocessor and display the output, we need to know the memory addresses and the output port address. To enter the program:

- Reset the system by using the RESET key.
- Press the EXAM MEM key and enter the first memory address using Hex keys, where the program should be stored.
- Enter each machine code by pushing Hex keys and then press NEXT key to enter the code in next memory location.
- Repeat it until the last machine code.
- Reset the system.

The Hex code get converted into binary code by the monitor program stored in Read-Only memory (or EPROM). An important function of the monitor program is to check the keys and convert Hex code into binary code.

C. Executing the Program

To execute the program, we press the GO key and then we need to tell the microprocessor where the program begins by entering the memory address. Now, we can push the Execute key to begin the execution. As soon as the Execute function key is pushed, the microprocessor loads the starting address of program in the program counter, and the program control is transferred from the Monitor program to our program.

The microprocessor begins to read one machine code at a time, and when it fetches the complete instruction, it executes that instruction. The critical concept that needs to be emphasized here is that the microprocessor can understand and execute only the binary instructions (or data); everything else (mnemonics, Hex code, comments) is for the convenience of human beings.

III. Conclusion

This report has focused pretty much on 8085 assembly language programming. All the basics of assembly language programming have been covered here. Best efforts have been made to cover all the basic requirements for writing and executing assembly language programs, more specifically, on the 8085 trainer kit. This review lab can be handy for the upcoming labs where being familiar with the concepts used in this particular lab, is a must. The solutions to the questions of this part of the lab are given in appendices I and II in order.

Appendix I

A table consists of ten 8-bit data starting at 8050H. Write an 8085 program to store the sum of odd numbers at 8060H and store sum of even numbers at 8070H. Also display the sum at output ports.

TABLE II
Solution to Question no.1

Address	Label	Mnemonics	Hexcode
8000		MVI A,80	3E, 80
8002		OUT 43	D3, 43
8004		MVI A,00	3E, 00
8006		STA 8060	32, 60, 80
8009		STA 8070	32, 70, 80
800C		MVI C,0A	OE, OA
800E		LXI H, 8050	21, 50, 80
8011	START	MOV A,M	7E
8012		RRC	0F
8013		JC ODD	DA, 24, 00
8016		LDA 8070	3A, 70 80
8019		ADD M	86
801A		STA 8070	32, 70, 80
801D		JNC SKIP	D2, 2F, 00
8020		INR B	04
8021		JMP SKIP	C3, 2F, 00
8024	ODD	LDA 8060	3A, 60, 80
8027		ADD M	86
8028		STA 8060	32, 60, 80
802B		JNC SKIP	D2, 2F, 00
802E		INR D	14
802F	SKIP	INX H	23
8030		DCR C	0D
8031		JNZ START	C2, 11, 00
8034		MOV A,B	78
8035		STA 8071	32, 71, 80
8038		OUT 40	D3, 40
803A		LDA 8070	3A, 70, 80
803D		OUT 41	D3, 41
803F		MOV A,D	7A
8040		STA 8061	32, 61, 80
8043		OUT 42	D3, 42
8045		LDA 8060	3A, 60, 80
8048		OUT 43	D3, 43
804A		RST 5	EF

Appendix II

Write a program to calculate the sum of following sequence 1×2+2×3+3×4+4×5+… up to n terms, where n is an 8-bit number stored at memory location 9000H. Display the 16-bit result in output ports. Use subroutine for multiplication.

TABLE III
Solution to Question no.2

Address	Label	Mnemonics	Hexcode
8000		MVI A,80	3E, 80
8002		OUT 43	D3, 43
8004		MVI A,00	3E, 00
8006		STA 9050	32, 50, 90
8009		STA 9051	32, 51, 90
800C		LHLD 9050	2A, 50, 90
800F		LDA 9000	3A, 00, 90
8012		LXI B,0102	01, 02, 01
8015	START	CALL MUL	CD, 26, 80
8018		DAD D	19
8019		INR B	04
801A		INR C	0C
801B		DCR A	3D
801C		JNZ START	C2, 15, 80
801F		MOV A,L	7D

8020		OUT 40	D3, 40
8022		MOV A,H	7C
8023		OUT 41	D3, 41
8025		RST 5	EF
8026	MUL	PUSH PSW	F5
8027		PUSH B	C5
8028		XRA A	AF
8029	REP	ADD B	80
802A		JNC SKIP	D2, 2E, 80
802D		INR D	14
802E	SKIP	DCR C	0D
802F		JNZ REP	C2, 29, 80
8032		MOV E,A	5F
8033		POP B	C1
8034		POP PSW	F1
8035		RET	C9

ACKNOWLEDGEMENT

We, the students of department of Electronics and Computer Engineering, wish to acknowledge Daya Sagar Baral Sir, Dinesh Baniya Kshatri Sir and other contributors for helping complete this part of the lab with accuracy, either in a direct or an indirect way. Their class lectures proved to be quite handy in solving the exercises of this part of the lab. We would also like to acknowledge Kamal Nepal Sir for his support and assistance with the trainer kit, in the lab. We also wish to acknowledge our classmates, discussions with whom, made the concepts on certain topics even more clear.

REFERENCES

[1] R.S. Gaonkar, "Microprocessor Architecture, Programming and Applications with the 8085", 5th ed., Pearson Education, 1996.

[2] The website for electronics study materials. [Online]. Available: http://www.daenotes.com/

[3] A webpage on 8085 trainer kit. [Online]. Available: http://www.pantechsolutions.net/microcontroller-boards/8085-trainer-kit-user-and-technical-reference-manual/