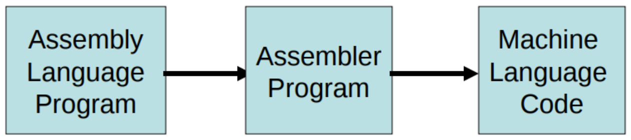
Unit II Introduction to Assembly Language Programming

Assembly Language Programming Basics

An assembly language is the most basic programming language available for any processor. With assembly language, a programmer works only with operations that are implemented directly on the physical CPU.

Assembly languages generally lack high-level conveniences such as variables and functions, and they are not portable between various families of processors. They have the same structures and set of commands as machine language, but allow a programmer to use names instead of numbers. This language is still useful for programmers when speed is necessary or when they need to carry out an operation that is not possible in high-level languages.
Assembly language is specific to a given processor. For e.g. assembly language of 8085 is different than that of Motorola 6800 microprocessor.

Microprocessor cannot understand a program written in Assembly language. A program known as Assembler is used to convert Assembly language program to machine language.



Assembly language program to add two numbers

```
MVI A, 2H  ;Copy value 2H in register A
MVI B, 4H  ;Copy value 4H in register B
ADD B           ;A = A + B
```

Advantages of Assembly Language

a) The symbolic programming of Assembly Language is easier to understand and saves a lot of time and effort of the programmer.
b) It is easier to correct errors and modify program instructions.
c) Assembly Language has the same efficiency of execution as the machine level language.

Disadvantages of Assembly Language

a) One of the major disadvantages is that assembly language is machine dependent. A program written for one computer might not run in other computers with different hardware configuration.
b) If you are programming in assembly language, you must have detailed knowledge of the particular microcomputer you are using.
c) Assembly language programs are not portable.

Classification of Instructions

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called **Instruction Set**. **8085** has **246** instructions. Each instruction is represented by an 8-bit binary value. These 8-bits of binary value is called Op-Code or Instruction Byte.

a) Data Transfer Instruction
b) Arithmetic Instructions
c) Logical Instructions
d) Branching Instructions
e) Control Instructions

Data Transfer Instruction

These instructions move data between registers, or between memory and registers. These instructions copy data from source to destination. While copying, the contents of source are not modified.
Example: MOV, MVI

Arithmetic Instructions

These instructions perform the operations like:
- Addition
- Subtract
- Increment
- Decrement

Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator. The result (sum) is stored in the accumulator. No two other 8-bit registers can be added directly.

Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator. The result is stored in the accumulator. No two other 8-bit registers can be subtracted directly.
The 8-bit contents of a register or a memory location can be incremented or decremented by 1. The 16-bit contents of a register pair can be incremented or decremented by 1. Increment or decrement can be performed on any register or a memory location.

Logical Instructions

These instructions perform logical operations on data stored in registers and memory.
The logical operations are:
- AND
- OR
- XOR
- Rotate
- Compare
- Complement

AND, OR, XOR
Any 8-bit data, or the contents of register, or memory location can logically have AND operation, OR operation, XOR operation with the contents of accumulator. The result is stored in accumulator.

Rotate
Each bit in the accumulator can be shifted either left or right to the next position.

Compare
Any 8-bit data, or the contents of register, or memory location can be compares for:

- Equality
- Greater Than
- Less Than

with the contents of accumulator.

Complement
The contents of accumulator can be complemented. Each 0 is replaced by 1 and each 1 is replaced by 0.

Branching Instructions

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.
The three types of branching instructions are:

- Jump
- Call
- Return

Jump Instructions
The jump instruction transfers the program sequence to the memory address given in the operand.
JMP 2050 (Jumps to the address 2050)
Call Instructions
The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack.
CALL 2050 (Transfers program sequence to address 2050)
Return Instructions
The return instruction transfers the program sequence from the subroutine to the calling program.
RET (Return from the subroutine)

Control Instructions

The control instructions control the operation of microprocessor.

Examples:

NOP (No operation is performed)
HLT (The CPU finishes executing the current instruction and halts any further execution)
DI (Disable Interrupts)
EI (Enable Interrupts)

Instruction Set of 8085

Data Transfer Group

Instruction	Addressing Modes	Operation	Description
LDA address	Direct	$[A] \leftarrow [M_{address}]$	**Loads the accumulator direct** **(**transfers contents stored in memory location to accumulator**)**
STA address	Direct	$[M_{address}] \leftarrow [A]$	**Stores the accumulator direct** **(**transfers contents stored in accumulator to memory address**)**
MOV A, A	Register	$[A] \leftarrow [A]$	tranfers the contents from one register to another
MOV A, B	Register	$[A] \leftarrow [B]$	
MOV A, C	Register	$[A] \leftarrow [C]$	
MOV A, D	Register	$[A] \leftarrow [D]$	
MOV A, E	Register	$[A] \leftarrow [E]$	
MOV A, H	Register	$[A] \leftarrow [H]$	
MOV A, L	Register	$[A] \leftarrow [L]$	
MOV B, A	Register	$[B] \leftarrow [A]$	
MOV B, B	Register	$[B] \leftarrow [B]$	
MOV B, C	Register	$[B] \leftarrow [C]$	
MOV B, D	Register	$[B] \leftarrow [D]$	

MOV B, E	Register	$[B] \leftarrow [E]$	
MOV B, H	Register	$[B] \leftarrow [H]$	
MOV B, L	Register	$[B] \leftarrow [L]$	
MOV C, A	Register	$[C] \leftarrow [A]$	
MOV C, B	Register	$[C] \leftarrow [B]$	
MOV C, C	Register	$[C] \leftarrow [C]$	
MOV C, D	Register	$[C] \leftarrow [D]$	
MOV C, E	Register	$[C] \leftarrow [E]$	
MOV C, H	Register	$[C] \leftarrow [H]$	
MOV C, L	Register	$[C] \leftarrow [L]$	
MOV D, A	Register	$[D] \leftarrow [A]$	
MOV D, B	Register	$[D] \leftarrow [B]$	
MOV D, C	Register	$[D] \leftarrow [C]$	
MOV D, D	Register	$[D] \leftarrow [D]$	
MOV D, E	Register	$[D] \leftarrow [E]$	
MOV D, H	Register	$[D] \leftarrow [H]$	
MOV D, L	Register	$[D] \leftarrow [L]$	
MOV E, A	Register	$[E] \leftarrow [A]$	
MOV E, B	Register	$[E] \leftarrow [B]$	
MOV E, C	Register	$[E] \leftarrow [C]$	
MOV E, D	Register	$[E] \leftarrow [D]$	
MOV E, E	Register	$[E] \leftarrow [E]$	
MOV E, H	Register	$[E] \leftarrow [H]$	

MOV E, L	Register	$[E] \leftarrow [L]$	
MOV H, A	Register	$[H] \leftarrow [A]$	
MOV H, B	Register	$[H] \leftarrow [B]$	
MOV H, C	Register	$[H] \leftarrow [C]$	
MOV H, D	Register	$[H] \leftarrow [D]$	
MOV H, E	Register	$[H] \leftarrow [E]$	
MOV H, H	Register	$[H] \leftarrow [H]$	
MOV H, L	Register	$[H] \leftarrow [L]$	
MOV L, A	Register	$[L] \leftarrow [A]$	
MOV L, B	Register	$[L] \leftarrow [B]$	
MOV L, C	Register	$[L] \leftarrow [C]$	
MOV L, D	Register	$[L] \leftarrow [D]$	
MOV L, E	Register	$[L] \leftarrow [E]$	
MOV L, H	Register	$[L] \leftarrow [H]$	
MOV L, L	Register	$[L] \leftarrow [L]$	
MOV A, M	Register indirect	$[A] \leftarrow [M_{H-L}]$	It moves / copies the data stored in memory location whose address is given in H-L register pair, to the given register.
MOV B, M	Register indirect	$[B] \leftarrow [M_{H-L}]$	
MOV C, M	Register indirect	$[C] \leftarrow [M_{H-L}]$	
MOV D, M	Register indirect	$[D] \leftarrow [M_{H-L}]$	
MOV E, M	Register indirect	$[E] \leftarrow [M_{H-L}]$	
MOV H, M	Register indirect	$[H] \leftarrow [M_{H-L}]$	

MOV L, M	Register indirect	$[L] \leftarrow [M_{H-L}]$	
MOV M, A	Register indirect	$[M_{H-L}] \leftarrow [A]$	This instruction moves / copies the data in the given register to the memory location addressed by H-L register pair.
MOV M, B	Register Indirect	$[M_{H-L}] \leftarrow [B]$	
MOV M, C	Register indirect	$[M_{H-L}] \leftarrow [C]$	
MOV M, D	Register indirect	$[M_{H-L}] \leftarrow [D]$	
MOV M, E	Register indirect	$[M_{H-L}] \leftarrow [E]$	
MOV M, H	Register indirect	$[M_{H-L}] \leftarrow [H]$	
MOV M, L	Register indirect	$[M_{H-L}] \leftarrow [L]$	
MVI A, data	Immediate	$[A] \leftarrow data$	This instruction transfers the given data to the register.
MVI B, data	Immediate	$[B] \leftarrow data$	
MVI C, data	Immediate	$[C] \leftarrow data$	
MVI D, data	Immediate	$[D] \leftarrow data$	
MVI E, data	Immediate	$[E] \leftarrow data$	
MVI H, data	Immediate	$[H] \leftarrow data$	
MVI L, data	Immediate	$[L] \leftarrow data$	
MVI M, data	Immediate	$[M_{H-L}] \leftarrow data$	

Arithmetic Group of Instructions

Instruction	Addressing Modes	Operation	Description
ADD A	Register	$[A] \leftarrow [A] + [A]$	It adds the content stored in given register with the accumulator. The result of this addition is stored in accumulator.
ADD B	Register	$[A] \leftarrow [A] + [B]$	
ADD C	Register	$[A] \leftarrow [A] + [C]$	
ADD D	Register	$[A] \leftarrow [A] + [D]$	
ADD E	Register	$[A] \leftarrow [A] + [E]$	
ADD H	Register	$[A] \leftarrow [A] + [H]$	
ADD L	Register	$[A] \leftarrow [A] + [L]$	
ADD M	Register indirect	$[A] \leftarrow [A] + [M_{H-L}]$	adds the content of memory location whose address is given in H-L register pair with the accumulator and the answer is stored in accumulator.
ADI data	Immediate	$[A] \leftarrow [A] + data$	It immediately adds the given data with the accumulator and the answer will be stored in Accumulator.
ADC A	Register	$[A] \leftarrow [A] + [A] + CY$	It adds the content stored in given register and content of CY flag with the content of accumulator. The result of this addition is stored in accumulator.
ADC B	Register	$[A] \leftarrow [A] + [B] + CY$	
ADC C	Register	$[A] \leftarrow [A] + [C] + CY$	

ADC D	Register	$[A] \leftarrow [A] + [D] + CY$	
ADC E	Register	$[A] \leftarrow [A] + [E] + CY$	
ADC H	Register	$[A] \leftarrow [A] + [H] + CY$	
ADC L	Register	$[A] \leftarrow [A] + [L] + CY$	
ADC M	Register indirect	$[A] \leftarrow [A] + [M_{H-L}] + CY$	adds the content of memory location whose address is given in H-L register pair to the accumulator with carry and the answer is stored in accumulator.
ACI data	Immediate	$[A] \leftarrow [A] + data + CY$	It immediately adds the given data to the accumulator with carry and the answer will be stored in Accumulator.
DAD B	Register	$BC \leftarrow BC + BC$	Adds the contents of pair register and store in pair
DAD D	Register	$DE \leftarrow DE + DE$	
DAD H	Register	$HL \leftarrow HL + HL$	
SUB A	Register	$[A] \leftarrow [A] - [A]$	It subtracts the content stored in given register with the accumulator. The result of this subtraction is stored in accumulator.
SUB B	Register	$[A] \leftarrow [A] - [B]$	
SUB C	Register	$[A] \leftarrow [A] - [C]$	
SUB D	Register	$[A] \leftarrow [A] - [D]$	
SUB E	Register	$[A] \leftarrow [A] - [E]$	
SUB H	Register	$[A] \leftarrow [A] - [H]$	
SUB L	Register	$[A] \leftarrow [A] - [L]$	

SUB M	Register indirect	$[A] \leftarrow [A] - [M_{H-L}]$	subracts the content of memory location whose address is given in H-L register pair with the accumulator and the answer is stored in accumulator.
SUI data	Immediate	$[A] \leftarrow [A] - data$	It immediately subtracts the given data with the accumulator and the answer will be stored in Accumulator.
SBB A	Register	$[A] \leftarrow [A] - [A] - CY$	Subtract with borrow
SBB B	Register	$[A] \leftarrow [A] - [B] - CY$	
SBB C	Register	$[A] \leftarrow [A] - [C] - CY$	
SBB D	Register	$[A] \leftarrow [A] - [D] - CY$	
SBB E	Register	$[A] \leftarrow [A] - [E] - CY$	
SBB H	Register	$[A] \leftarrow [A] - [H] - CY$	
SBB L	Register	$[A] \leftarrow [A] - [L] - CY$	
SBB M	Register indirect	$[A] \leftarrow [A] - [M_{H-L}] - CY$	Subtracts the contents of memory address given by H-L pair with accumulator with borrow
SBI data	Immediate	$[A] \leftarrow [A] - data - CY$	Subtract immediate with borrow

INR A	Register	$[A] \leftarrow [A] + 1$	Increment contents of register by 1
INR B	Register	$[B] \leftarrow [B] + 1$	
INR C	Register	$[C] \leftarrow [C] + 1$	

INR D	Register	$[D] \leftarrow [D] + 1$	
INR E	Register	$[E] \leftarrow [E] + 1$	
INR H	Register	$[H] \leftarrow [H] + 1$	
INR L	Register	$[L] \leftarrow [L] + 1$	
INR M	Register indirect	$[M_{H-L}] \leftarrow [M_{H-L}] + 1$	Increments contents of memory address given by H-L pair by 1
INX B	Register	$BC \leftarrow BC + 1$	**Increment register pair by 1**
INX D	Register	$DE \leftarrow DE + 1$	
INX H	Register	$HL \leftarrow HL + 1$	
DCR A	Register	$[A] \leftarrow [A] - 1$	Decrement contents of register by 1
DCR B	Register	$[B] \leftarrow [B] - 1$	
DCR C	Register	$[C] \leftarrow [C] - 1$	
DCR D	Register	$[D] \leftarrow [D] - 1$	
DCR E	Register	$[E] \leftarrow [E] - 1$	
DCR H	Register	$[H] \leftarrow [H] - 1$	
DCR L	Register	$[L] \leftarrow [L] - 1$	

DCR M	Register indirect	$[M_{H-L}] \leftarrow [M_{H-L}] - 1$	Decrements contents of memory address given by H-L pair by 1
DCX B	Register	$BC \leftarrow BC - 1$	**Decrement register pair by 1**
DCX D	Register	$DE \leftarrow DE - 1$	
DCX H	Register	$HL \leftarrow HL - 1$	
RLC	Implicit/implied	Rotate left through carry	
RAL	Implicit/implied	Rotate all left	
RRC	Implicit/implied	Rotate right through carry	
RAR	Implicit/implied	Rotate all right	

Logic Transfer Group

Instructions	Addressing Modes	Operation	Description
ANA A	Register	$[A] \leftarrow [A].AND.[A]$	In this instruction each bit of the given register contents are ANDed with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator. It does not affect the contents of the given register.
ANA B	Register	$[A] \leftarrow [A].AND.[B]$	
ANA C	Register	$[A] \leftarrow [A].AND.[C]$	

ANA D	Register	$[A] \leftarrow [A].AND.[D]$	
ANA E ANA H	Register Register	$[A] \leftarrow [A].AND.[E]$ $[A] \leftarrow [A].AND.[H]$	
ANA L	Register	$[A] \leftarrow [A].AND.[L]$	
ANA M	Register Indirect	$[A] \leftarrow [A].AND.[M_{H-L}]$	In this instruction each bit of the data stored in the memory location addressed by H-L register pair are ANDed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator like *ANA*.
ANI data	Immediate	$[A] \leftarrow [A].AND.data$	In this instruction each bit of the given data is immediately ANDed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.

ORA A	Register	$[A] \neg [A].OR.[A]$	In this instruction each bit of the given register contents are ORed with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator.
ORA B	Register	$[A] \neg [A].OR.[B]$	
ORA C	Register	$[A] \neg [A].OR.[C]$	
ORA D	Register	$[A] \neg [A].OR.[D]$	

ORA E	Register	$[A] \leftarrow [A].OR.[E]$	
ORA H	Register	$[A] \neg [A].OR.[H]$	
ORA L	Register	$[A] \neg [A].OR.[L]$	
ORA M	Register Indirect	$[A] \neg [A].OR.[M_{H-L}]$	In this instruction each bit of the data stored in the memory location addressed by H-L register pair are ORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.
ORI data	Immediate	$[A] \neg [A].OR.data$	In this instruction each bit of the given data is immediately ORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.
XRA A	Register	$[A] \neg [A]$XOR $[A]$	In this instruction each bit of the given register contents are *XORed* with each bit of the accumulator contents (bit by bit). The result is saved in the accumulator.
XRA B	Register	$[A] \neg [A]$XOR $[B]$	
XRA C	Register	$[A] \neg [A]$ XOR $[C]$	
XRA D	Register	$[A] \neg [A]$ XOR $[D]$	
XRA E	Register	$[A] \neg [A]$ XOR $[E]$	
XRA H	Register	$[A] \neg [A]$ XOR.$[H]$	

XRA L	Register	$[A] \neg [A]$ XOR $[L]$	
XRA M	Register Indirect	$[A] \neg [A]$ XOR $[M_{H-L}]$	In this instruction each bit of the data stored in the memory location addressed by H-L register pair are XORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.
XRI data	Immediate	$[A] \neg [A]$ XOR *data*	In this instruction each bit of the given data is immediately XORed with each bit of the accumulator contents (bit by bit). The result is stored in the accumulator.
CMP A	Register		The contents of the given register are compared with the accumulator contents. In fact the contents of the register are subtracted from the contents of accumulator and the accumulator contents remain unchanged.
CMP B	Register		
CMP C	Register		
CMP D	Register		
CMP E	Register		
CMP H	Register		
CMP L	Register		
CMP M	Register Indirect		the contents of the addressed memory location will be compared by the accumulator contents.
CPI data	Immediate		In this instruction the given data is compared with the accumulator contents.

CMA	---	$[A] \leftarrow [A]^{COM}$	The execution of this instruction inverts each bit of the accumulator contents and the result is saved in the accumulator.
CMC	---	$CY \leftarrow \overline{CY}$	Complements the carry flag.
STC	---	$CY \leftarrow 1$	It sets the carry flag. Only carry flag gets affected on this instruction.

Branching Group

Instructions	**Addressing Modes**	**Operation**	**Description**
JMP addr	Immediate	$[PC] \leftarrow [addr]$	Unconditional Jump
JNZ addr	Immediate	$[PC] \leftarrow [addr]$ if Z=0.	Conditional Jump
JZ addr	Immediate	$[PC] \leftarrow [addr]$ if Z=1.	
JNC addr	Immediate	$[PC] \leftarrow [addr]$ if CY=0.	
JC addr	Immediate	$[PC] \leftarrow [addr]$ if CY=1.	
JM addr	Immediate	$[PC] \leftarrow [addr]$ if S=1.	
JP addr	Immediate	$[PC] \leftarrow [addr]$ if S=0.	
JPO addr	Immediate	$[PC] \leftarrow [addr]$ if P=0.	
JPE addr	Immediate	$[PC] \leftarrow [addr]$ if P=1.	
CALL addr	Immediate	Calls subroutine program.	Unconditional Call
CNZ addr	Immediate	Calls subroutine program if Z=1.	Conditional Call
CZ addr	Immediate	Calls subroutine program if Z=0.	

CNC addr	Immediate	Calls subroutine program if CY=1.	
CC addr	Immediate	Calls subroutine program if CY=0.	
CM addr	Immediate	Calls subroutine program if S=1.	
CP addr	Immediate	Calls subroutine program if S=0.	
CPO addr	Immediate	Calls subroutine program if P=0.	
CPE addr	Immediate	Calls subroutine program if P=1.	
RNZ	Register indirect	Returns to main program if Z=1.	
RZ	Register indirect	Returns to main program if Z=0.	
RNC	Register indirect	Returns to main program if CY=0.	
RC	Register indirect	Returns to main program if CY=1.	
RM	Register indirect	Returns to main program if S=1.	

RP	Register indirect	Returns to main program if S=0.	
RPO	Register indirect	Returns to main program if P=0.	
RPE	Register indirect	Returns to main program if P=1.	

Control Instructions Group

Instruction	Meaning	Explanation
NOP	No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT	Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI	Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM	Read interrupt mask	This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
SIM	Set interrupt mask	This instruction is used to implement the interrupts 7.5, 6.5, 5.5, and serial data output.