Machine Language

Machine Language is the language written as strings of binary 1`s and 0`s. It is the only language which a computer understands without using a translation program.

A machine language instruction has two parts. The first part is the operation code which tells the computer what function to perform and the second part is the operand which tells the computer where to find or store the data which is to be manipulated. A programmer needs to write numeric codes for the instruction and storage location of data.

Disadvantages –

- It is machine dependant i.e. it differs from computer to computer.
- It is difficult to program and write
- It is prone to errors
- It is difficult to modify

Assembly Language

It is a low level programming language that allows a user to write a program using alphanumeric mnemonic codes, instead of numeric codes for a set of instructions.It requires a translator known as assembler to convert assembly language into machine language so that it can be understood by the computer. It is easier to remember and write than machine language.

Assembler – It is a computer program which converts or translates assembly language into machine language. It assembles the machine language program in the main memory of the computer and makes it ready for execution.

Advantages –

- It is easy to understand and use
- It is easy to locate and correct errors
- It is easier to modify

Disadvantages –

- It is machine dependant

High level Language

It is a machine independent language. It enables a user to write programs in a language which resembles English words and familiar mathematical symbols. COBOL was the first high level language developed for business.

Each statement in a high level language is a micro instruction which is translated into several machine language instructions.

A **compiler** is a translator program which translates a high level programming language into equivalent machine language programs. It compiles a set of machine language instructions for every high level language program.

Source code: It is the input or the programming instructor of a procedural language.

The compiler translates the source code into machine level language which is known as object code. Object code can be saved and executed as and when desired by the user.

Linker: A program used with a compiler to provide links to the libraries needed for an executable program. It takes one or more object code generated by a compiler and combines them into a single executable program.

Interpreter: It is a translator used for translating high level language into the desired output. It takes one statement, translates it into machine language instructions and then immediately executes the result. Its output is the result of program execution.

Advantages of High level Language –

- It is machine independent
- It is easier to learn and use
- It is easier to maintain and gives few errors

Disadvantages –

- It lowers efficiency
- It is less flexible

MNEMONIC: English word MNEMONIC means "A device such as a pattern of letters, ideas, or associations that assists in remembering something.". So, its usually used by assembly language programmers to remember the "OPERATIONS" a machine can do, like "ADD" and "MUL" and "MOV" etc. This is assembler specific.

INSTRUCTION FORMAT

An instruction (instruction format) is a command to the microprocessor to perform a given task on a particular data. Each instruction (instruction format) is of two parts. One is task to be performed, called the operation code or **opcode** and the second one is the data to be operated on, called the **operand**. The operands or data can be specified in different ways. It may include an 8-bit or 16-bit data, an internal register. a memory location, or 8-bit or 16-bit address. In some instructions, the operand is implicit.
Instruction Word Size

The 8085 instruction set is of three groups according to word size:

- **One-word or one-byte instructions.**
- **Two-word or two-byte instructions.**
- **Three-word or three-byte instructions.**
-

In the 8085 microprocessor, byte and words are synonymous because it is an 8-bit microprocessor. But, instructions are commonly referred to in terms of bytes rather than words.

One-byte instructions

A one-byte instruction includes a opcode and a operand in the same byte. Operand(s) are internal registers and are in the instruction in form of codes. If there is no numeral present in the instruction then that instruction will be of one-byte, for example, MOV C, A, RAL, and ADD B, etc. Table M.1 shows examples of one-byte instruction.

Task	Opcode	Operand
Copy the content of accumulator in the register C.	MOV	C, A
Add the contents of register B to the contents of the accumulator.	ADD	B
Invert each bit in the accumulator.	CMA	None

These instructions are of one-byte performing three different tasks. In the first instruction, operand and registers are specified. In the second instruction, the operand B is specific and the accumulator is not there. Similarly, in the third instruction, the accumulator is assume to be the implicit operand. These instructions are in 8-bit binary format in the memory and each requires one memory location.

Two-byte instructions

In a two-byte instruction, the first byte specifies the operation code and second byte specifies the operand. Source operand is a data byte and immediately following the opcode. If an 8-bit numeral is present in the instruction then that instruction will be of two-byte. Here, the numeral may be a data or an address. For example, in MVI A, 35H and IN 29H, etc. In a two-byte instruction, the first byte will be the opcode and the second byte will be for the numeral present in the instruction.

Task	Opcode	Operand
Load an 8-bit data byte in the accumulator.	MVI	A 35H

Three-byte instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit operand. The second byte is the low-order operand and the third byte is the high-order operand. If a 16-bit numeral is present in the instruction then that instruction will be of three-byte. Here, the numeral may be a data or an address, for example, in LXI H,3500H and STA 2500H, etc.

Task	**Opcode**
Transfer the program sequence to the memory location 2085h	JMP

Addressing modes in 8085 microprocessor

The way of specifying data to be operated by an instruction is called addressing mode.

Types of addressing modes –
In 8085 microprocessor there are 5 types of addressing modes:

1. **Immediate Addressing Mode –**
   In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:
MVI B 45 (move the data 45H immediately to register B)
LXI H 3050 (load the H-L pair with the operand 3050H immediately)
JMP address (jump to the operand address immediately)

2. **Register Addressing Mode –**
   In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore the operation is performed within various registers of the microprocessor.
   Examples:
   MOV A, B (move the contents of register B to register A)
   ADD B (add contents of registers A and B and store the result in

register A)
INR A (increment the contents of register A by one)

3. **Direct Addressing Mode –**
   In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.
   Examples:
   LDA 2050 (load the contents of memory location into accumulator A)
   LHLD address (load contents of 16-bit memory location into H-L register pair)
   IN 35 (read the data from port whose address is 01)

4. **Register Indirect Addressing Mode –**
   In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.
   Examples:
   MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)
   LDAX B (move contains of B-C register to the accumulator)
   LXIH 9570 (load immediate the H-L pair with the address of the location 9570)

5. **Implied/Implicit Addressing Mode –**
   In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.
   Examples:
   CMA (finds and stores the 1's complement of the contains of accumultor A in A)
   RRC (rotate accumulator A right by one bit)
   RLC (rotate accumulator A left by one bit)

Instruction Set of Intel 8085 Microprocessor

An Instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

1. **Data Transfer Group**
2. **Arithmetic Group**
3. **Logical Group**
4. **Branch Control Group**
5. **I/O and Machine Control Group**

Data Transfer Group

Instructions, which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group.

Examples are: MOV, MVI, LXI, LDA, STA etc. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source.

For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

1. MOV r1, r2 (Move Data; Move the content of the one register to another). [r1] <-- [r2]
2. MOV r, m (Move the content of memory register). r <-- [M]
3. MOV M, r. (Move the content of register to memory). M <-- [r]
4. MVI r, data. (Move immediate data to register). [r] <-- data.
5. MVI M, data. (Move immediate data to memory). M <-- data.
6. LXI rp, data 16. (Load register pair immediate). [rp] <-- data 16 bits, [rh] <-- 8 LSBs of data.
7. LDA addr. (Load Accumulator direct). [A] <-- [addr].
8. STA addr. (Store accumulator direct). [addr] <-- [A].
9. LHLD addr. (Load H-L pair direct). [L] <-- [addr], [H] <-- [addr+1].
10. SHLD addr. (Store H-L pair direct) [addr] <-- [L], [addr+1] <-- [H].
11. LDAX rp. (LOAD accumulator indirect) [A] <-- [[rp]]
12. STAX rp. (Store accumulator indirect) [[rp]] <-- [A].

13.XCHG. (Exchange the contents of H-L with D-E pair) [H-L] <--> [D-E].

Arithmetic Group

The instructions of this group perform arithmetic operations such as addition, subtraction; increment or decrement of the content of a register or memory. Examples are: ADD, SUB, INR, DAD etc.

1. **Arithmetic Group**
    1. ADD r. (Add register to accumulator) [A] <-- [A] + [r].
    2. ADD M. (Add memory to accumulator) [A] <-- [A] + [[H-L]].
    3. ADC r. (Add register with carry to accumulator). [A] <-- [A] + [r] + [CS].
    4. ADC M. (Add memory with carry to accumulator) [A] <-- [A] + [[H-L]] [CS].
    5. ADI data (Add immediate data to accumulator) [A] <-- [A] + data.
    6. ACI data (Add with carry immediate data to accumulator). [A] <-- [A] + data + [CS].
    7. DAD rp. (Add register paid to H-L pair). [H-L] <-- [H-L] + [rp].
    8. SUB r. (Subtract register from accumulator). [A] <-- [A] – [r].
    9. SUB M. (Subtract memory from accumulator). [A] <-- [A] – [[H-L]].
    10. SBB r. (Subtract register from accumulator with borrow). [A] <-- [A] – [r] – [CS].
    11. SBB M. (Subtract memory from accumulator with borrow). [A] <-- [A] – [[H-L]] – [CS].
    12. SUI data. (Subtract immediate data from accumulator) [A] <-- [A] – data.
    13. SBI data. (Subtract immediate data from accumulator with borrow). [A] <-- [A] – data – [CS].
    14. INR r (Increment register content) [r] <-- [r] +1.
    15. INR M. (Increment memory content) [[H-L]] <-- [[H-L]] + 1.
    16. DCR r. (Decrement register content). [r] <-- [r] – 1.
    17. DCR M. (Decrement memory content) [[H-L]] <-- [[H-L]] – 1.
    18. INX rp. (Increment register pair) [rp] <-- [rp] – 1.
    19. DCX rp (Decrement register pair) [rp] <-- [rp] -1.
    20. DAA (Decimal adjust accumulator) .

The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in the decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

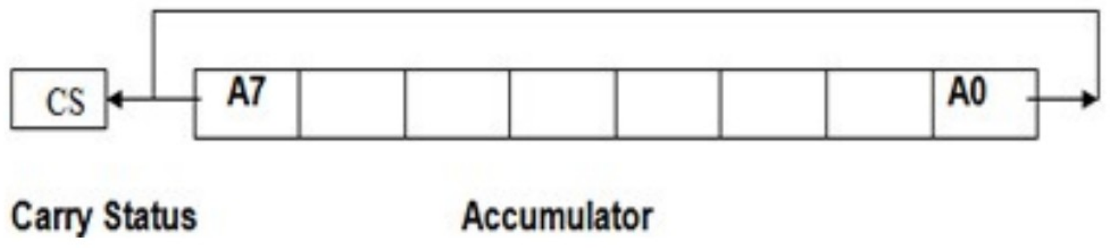
Logical Group

The Instructions under this group perform logical operation such as AND, OR, compare, rotate etc. Examples are: ANA, XRA, ORA, CMP, and RAL etc.

1. ANA r. (AND register with accumulator) [A] <-- [A] ^ [r].
2. ANA M. (AND memory with accumulator). [A] <-- [A] ^ [[H-L]].
3. ANI data. (AND immediate data with accumulator) [A] <-- [A] ^ data.
4. ORA r. (OR register with accumulator) [A] <-- [A] v [r].
5. ORA M. (OR memory with accumulator) [A] <-- [A] v [[H-L]]
6. ORI data. (OR immediate data with accumulator) [A] <-- [A] v data.
7. XRA r. (EXCLUSIVE – OR register with accumulator) [A] <-- [A] v [r]
8. XRA M. (EXCLUSIVE-OR memory with accumulator) [A] <-- [A] v [[H-L]]
9. XRI data. (EXCLUSIVE-OR immediate data with accumulator) [A] <-- [A]
10. CMA. (Complement the accumulator) [A] <-- [A]
11. CMC. (Complement the carry status) [CS] <-- [CS]
12. STC. (Set carry status) [CS] <-- 1.
13. CMP r. (Compare register with accumulator) [A] – [r]
14. CMP M. (Compare memory with accumulator) [A] – [[H-L]]
15. CPI data. (Compare immediate data with accumulator) [A] – data.

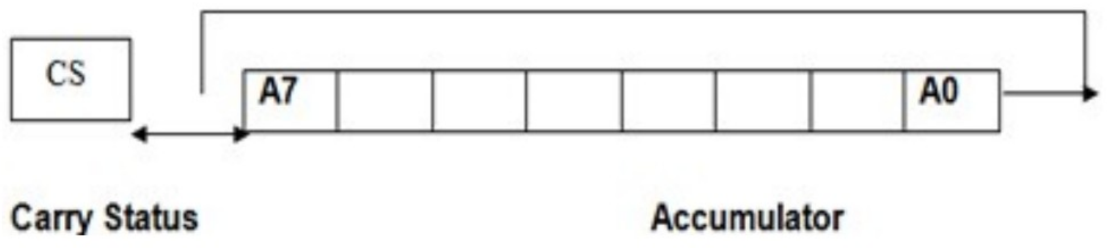
The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

16. RLC (Rotate accumulator left) [An+1] <-- [An], [A0] <-- [A7],[CS] <-- [A7].



The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected.

17. RRC. (Rotate accumulator right) [A7] <-- [A0], [CS] <-- [A0], [An] <-- [An+1].



The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected.

18. RAL. (Rotate accumulator left through carry) [An+1] <-- [An], [CS] <-- [A7], [A0] <-- [CS].
19. RAR. (Rotate accumulator right through carry) [An] <-- [An+1], [CS] <-- [A0], [A7] <-- [CS]

Branch Control Group

This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart. Examples are: JMP, JC, JZ, CALL, CZ, RST etc.

1. JMP addr (label). (Unconditional jump: jump to the instruction specified by the address). [PC] <-- Label.
2. Conditional Jump addr (label): After the execution of the conditional jump instruction the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled. If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 machine cycles: 10 states. If condition is not

true, only 2 machine cycles; 7 states are required for the execution of the instruction.

    1. **JZ** addr (label). (Jump if the result is zero)
    2. **JNZ** addr (label) (Jump if the result is not zero)
    3. **JC** addr (label). (Jump if there is a carry)
    4. **JNC** addr (label). (Jump if there is no carry)
    5. **JP** addr (label). (Jump if the result is plus)
    6. **JM** addr (label). (Jump if the result is minus)
    7. **JPE** addr (label) (Jump if even parity)
    8. **JPO** addr (label) (Jump if odd parity)

3. CALL addr (label) (Unconditional CALL: call the subroutine identified by the operand)

   CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stack top. Then the program jumps to subroutine starting at address specified by the label.

4. RET (Return from subroutine)
5. RST n (Restart) Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location.

I/O and Machine Control Group

This group includes the instructions for input/output ports, stack and machine control. Examples are: IN, OUT, PUSH, POP, and HLT etc.

Stack, I/O and Machine Control Group

1. IN port-address. (Input to accumulator from I/O port) [A] <-- [Port]
2. OUT port-address (Output from accumulator to I/O port) [Port] <-- [A]
3. PUSH rp (Push the content of register pair to stack)
4. PUSH PSW (PUSH Processor Status Word)
5. POP rp (Pop the content of register pair, which was saved, from the stack)
6. POP PSW (Pop Processor Status Word)
7. HLT (Halt)
8. XTHL (Exchange stack-top with H-L)

9. SPHL (Move the contents of H-L pair to stack pointer)
10. EI (Enable Interrupts)
11. DI (Disable Interrupts)
12. SIM (Set Interrupt Masks)
13. RIM (Read Interrupt Masks)
14. NOP (No Operation)