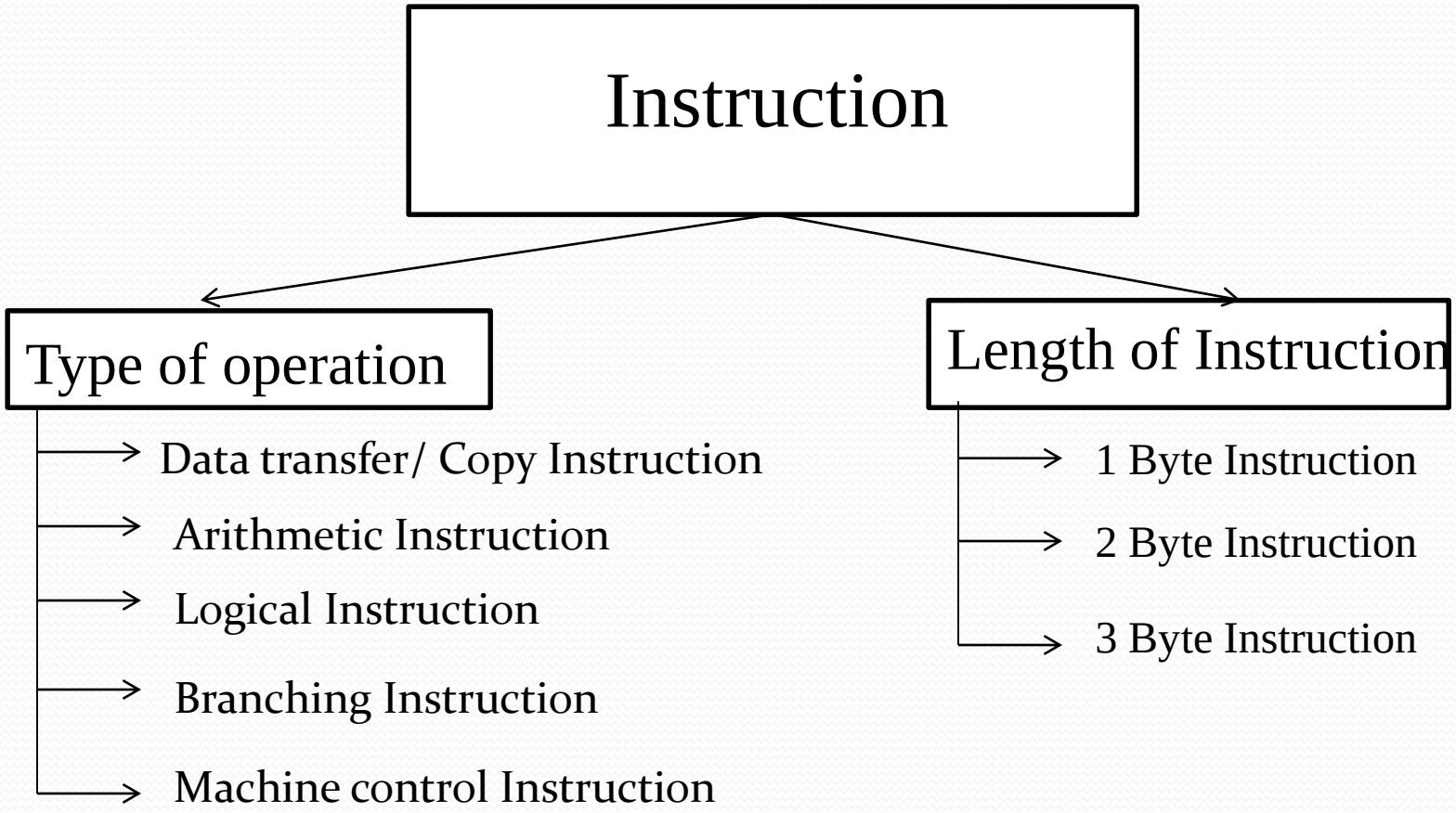
Instruction Set and Programming of Intel 8085 Microprocessor

UPES
UNIVERSITY WITH A PURPOSE

Instruction Set

- An instruction is a command given to the computer to perform a specified operation on a given data.
- The instruction set is the collection of instructions which the microprocessor is designed to execute.
- The assembly language program comprises of the collection of instructions to perform a certain operation.
- Different companies such as Intel, Motorola, etc. have different instruction set.


Instruction
Type of operation
Data transfer/ Copy Instruction
Arithmetic Instruction
Logical Instruction
Branching Instruction
Machine control Instruction
Length of Instruction
1 Byte Instruction
2 Byte Instruction
3 Byte Instruction

Data Transfer Instruction

- These instructions are used to transfer data from one location to another, i.e. source to destination. It may include:

✓ Register ⟶ Register (MOV A, B)

✓ Register ⟷ Memory (MOV M, r and MOV r, M)

✓ Data ⟶ Register (MVI r, data)

Arithmetic Instruction

- These instructions are used to perform arithmetic operations like addition, subtraction, increment, decrement. Example- ADD, SUB, INR, DAD, etc.

Logical Instruction

- These instructions performs logical operations such as AND, OR, compare, rotate. Example- ANA, XRA, CMP, ORA, RAL.

Branching Instruction

- The instruction under this group performs conditional and unconditional jump, subroutine call and return and restart. Example- JMP, CALL, RST, etc.

I/O and Machine control Instruction

- This group of instruction includes instruction for input/output ports, stack and machine control operations. Example- IN, OUT, PUSH, POP, HLT, etc.

DATA FORMAT

- There are various techniques of specifying data for an instruction. These can be defined as:

✓ Instruction itself includes 8 bit/16 bit data

✓ Instruction includes address of the memory location, I/O port.

✓ Instruction has only one register.

✓ Instruction has two register.

✓ In some instruction data is implied.

ADDRESSING MODES

- Instruction has opcode and operand. There are different techniques to specify operand in an instruction. These techniques are called as addressing modes.
- Intel 8085 has the following addressing modes:

a) Direct addressing mode
b) Register addressing mode
c) Register indirect addressing mode
d) Immediate addressing mode
e) Implicit/Implied addressing mode

a) **Direct addressing mode**

- Address of the operand (data) is given in the instruction itself.
- Example-

LDA 6000H	Load the accumulator with the contents of the memory location 6000H
STA 2400H	Store the contents of the accumulator in the memory location 2400H
IN 02	The data available on the 8 bit address of the input port is moved to the accumulator

b) Register addressing mode

- Operand is one of the GPR or the accumulator.
- Opcode specifies the operation to be performed and address of the registers.
- Operation takes place between registers.
- Example-

MOV A, B	Move the content of the register B to register A
ADD B	Add the content of the register B to the content of the register A

c) Register Indirect addressing mode

- Address of the data is present as the content of the another register pair.
- Example

LXI H, 2500H	Load H-L pair with 2500 H
MOV A,M	**Move the content of the memory location whose address is in H-L pair to the accumulator**
HLT	Halt

LXI H, 2500H	Load H-L pair with 2500 H
ADD M	**Add the content of the memory location whose address is in H-L pair to the content of the accumulator**
HLT	Halt

d) Immediate addressing mode

- In this addressing mode the operand (data) is specified within the instruction itself.
- Example

MVI A,05	Move the data 05 into the accumulator
ADI 06	Add 06 to the content of the accumulator
LXI H, 2500	2500 is 16 bit data and is loaded into H-L pair

e) Implicit/ Implied addressing mode

- In this type of instruction, operand is the content of the accumulator.
- No data, address of data, register are present in the instruction.
- Example

CMA	Complement the content of the accumulator
RAL	Rotate accumulator left through carry
RAR	Rotate accumulator right through carry
RLC	Rotate accumulator left

Symbols and Abbreviations

addr	16-bit address of the memory location
data	8-bit data
data 16	16-bit data
r,r_1,r_2	One of the register A, B, C, D, E, H, L
A, B, C, D, H, L	8-bit register
A	Accumulator
M	Memory whose address is in H-L pair
H	Appearing at the end of a group of digits specifies hexadecimal
rp	Represents one of the register pair B represents B-C pair D represents D-E pair H represents H-L pair
[]	The content of the register identified within the brackets
[[]]	The content of the memory location whose address is in the register pair identified within the brackets

Data Transfer Group

Instruction	Operation	Machine cycles
MOV A, B	Move the content of register B to register A	1
MOV r, M	Move the content of memory to register.	2
MOV M, r	Move the content of the register to the memory location addressed by H-L pair.	2
MVI r, data	Move immediate data to the register	2
MVI M, data	Move the immediate data to memory location addresses by H-L pair	3
LXI rp, data 16	Load the register pair with the 16 bit data	3
LDA addr	Load the accumulator with the content of the specified address in the instruction	4

STA addr	Store the content of the accumulator in the specified memory location	4
LHLD addr	Load the content of the specified memory location in the address to the L register and the content of the next memory location is loaded into H.	5
SHLD addr	Store the content of the L register in the specified address in the instruction and the content of the H register is stored in the next memory location.	5
LDAX rp	Load the accumulator with the content of the address stored in the rp	2
STAX rp	Store the content of the accumulator in the memory location whose address is in the register pair rp.	2
XCHG	Exchange the contents of H-L pair with D-E pair.	1

Arithmetic Group

ADD rl,	Add register to accumulator. [A]⟵[A]+[r]	1
ADD M	Add the content of the memory to accumulator [A]⟵ [A]+[[H-L]]	2
ADI data	Add immediate data to accumulator [A]⟵ [A]+data	2
ADC r	The content of register r and carry status are added with the content of the accumulator	1
DAD rp	Add the content of register pair rp to the content of H-L pair [H-L]⟵ [H-L]+[rp]	3
SUB r	Subtract register from memory [A]⟵ [A]-[r]	1
SUB M	Subtract memory from accumulator [A]⟵ [A]-[[H-L]]	2
SUI data	Subtract immediate data from accumulator [A]⟵ [A]-data	2

SBB r	The content of register r and carry status are subtracted from the content of the accumulator.	1
INR r	Content of register r are incremented by one [r]←[r]+1	1
INR M	Increment the content of memory location in H-L pair by one [[H-L]]← [[H-L]]+1	3
DCR r	Decrement the content of register by one [r] ←[r]-1	1
DCR M	Decrement the content of memory location in H-L pair by one [[H-L]]← [[H-L]]-1	3
INX rp	Increment the content of register pair rp by one [rp] ← [rp]+1	1
DCX rp	Decrement the content of register pair rp by one [rp] ← [rp]-1	1
DAA	Converts the content of accumulator from Hexadecimal to Decimal	1

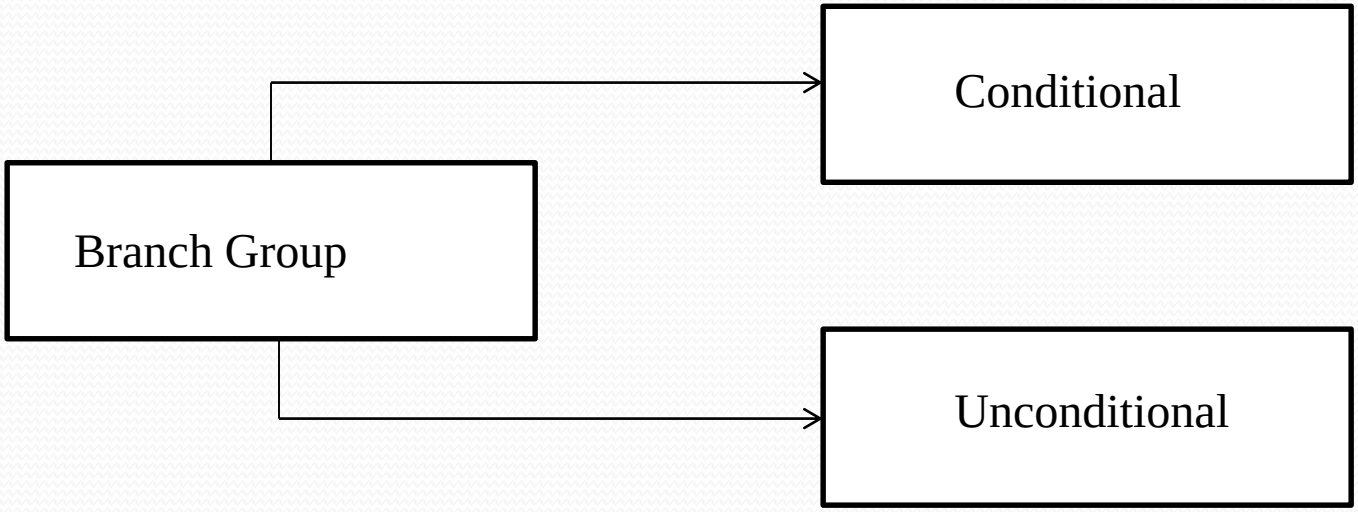
Logical Group

ANA r	The content of register r is ANDed with the content of accumulator	1
ANA M	The content of the memory location addressed by H-L pair is ANDed with content of accumulator	2
ANI data	The data is ANDed with the content of the accumulator	2
ORA r	The content of the register is ORed with the content of the accumulator	1
ORA M	The content of the memory location addressed by H-L pair is ORed with content of accumulator	2
ORA data	The data is ORed with the content of the accumulator	2
XRA r	The content of the register r is EXCLUSIVE-ORed with the content of the accumulator	1

XRA M	The content of the memory location addressed by H-L pair is EXCLUSIVE-ORed with content of accumulator	2
XRI data	The data is EXCLUSIVE-ORed with the content of the accumulator	2
CMA	Complement the content of the accumulator	1
CMP r	The content of the register r is subtracted with the content of the accumulator and the status flag are set according to the result	1
CMP M	The content of the memory location addressed by H-L pair is subtracted with the content of the accumulator and the status flag are set according to the result. The result of the accumulator remains unchanged.	2
RLC	The content of the accumulator is rotated left by one bit without carry`	1
RRC	The content of the accumulator is rotated right by one bit without carry	1
RAL	The content of the accumulator is rotated left by one bit with carry	1
RAR	The content of the accumulator is rotated right by one bit with carry	1

Branch Group

- The instruction in this group change the normal sequence of the program.



- Unconditional- The program is transferred to a specified memory location unconditionally.
- Conditional- The program is transferred to a specified memory location when certain condition is satisfied.

Unconditional instructions

- JMP addr- Program jump to the instruction specified by the address (label) unconditionally.

Conditional instructions

JZ addr	Jump if the result is zero
JNZ addr	Jump if the result is non-zero
JC addr	Jump if there is a carry
JNC addr	Jump if there is no carry
JP addr	Jump if the result is plus
JM addr	Jump if the result is minus
JPE addr	Jump if even parity
JPO addr	Jump if odd parity

Machine control Group

IN port-address	The data available on the port is moved to the accumulator	03
OUT port-address	The content of the accumulator is moved to the port specified by its address	03
HLT	The program execution is stopped	01

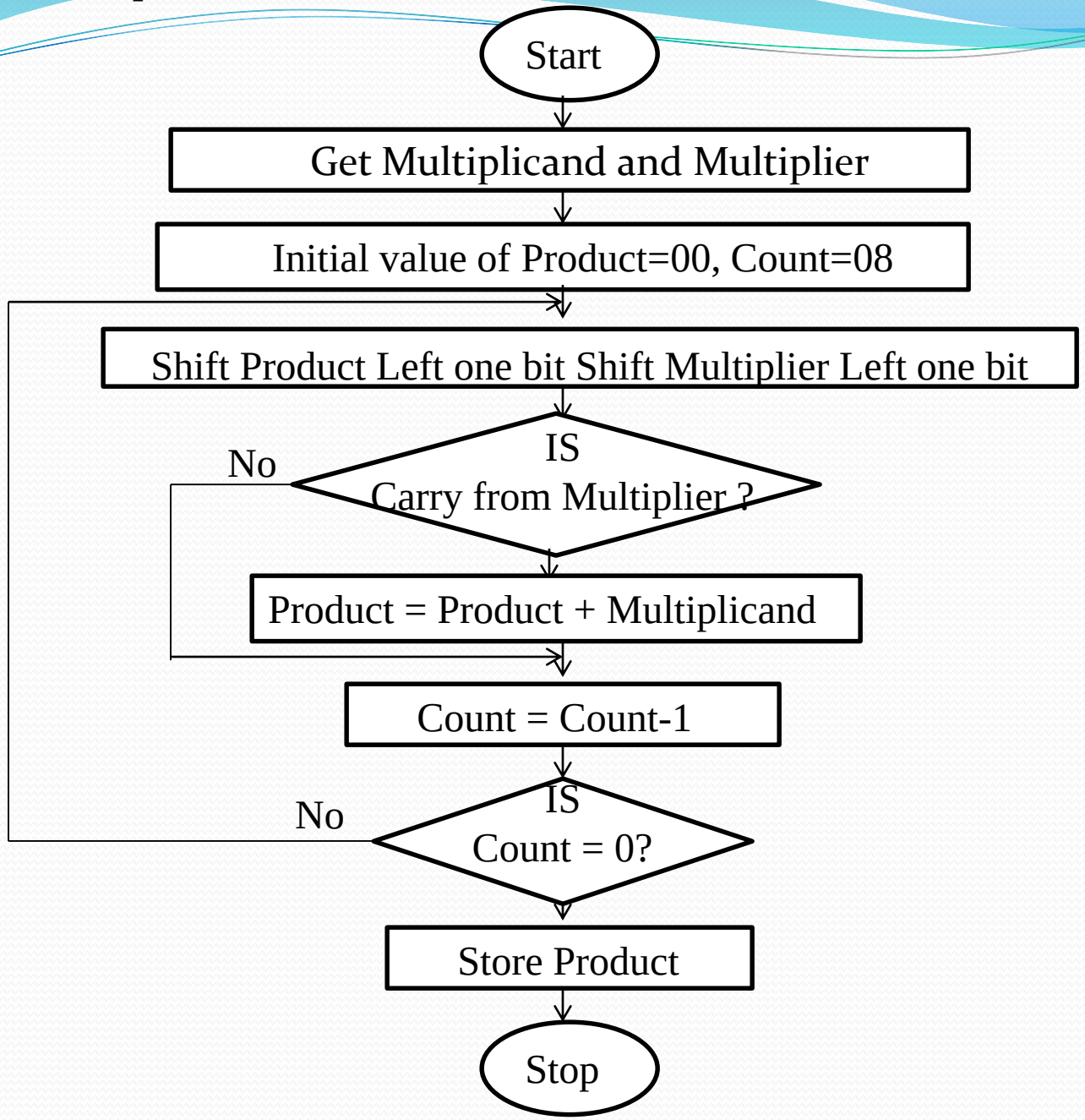
Problems

- WAP to place the content of the memory location FC50H in register B and that of FC51 in register C. The content of FC50 and FC51 are 11H and 12H.
- WAP to place 05 in the accumulator. Increment it by one and store the result in the memory location FC50H.
- WAP to add two numbers 49H and 56H which are stored in memory location 2501H and 2502H and store the result in 2503H.
- WAP to subtract two numbers 48 and 32 which are stored in 2501H and 2502H and store the result in 2503H.

- WAP to find one’s complement of 96H which is stored in 2501H and store the result in 2502H.
- WAP to find the one’s complement of 5485H. The number is stored in memory location 2501H, 2502H and the result is to be stored in memory location 2503H and 2504H.
- WAP to find two’s complement of 96H which is stored in 2501H and store the result in 2502H.
- WAP to find the two’s complement of 5B8CH. The number is stored in memory location 2501H, 2502H and the result is to be stored in memory location 2503H and 2504H.
- WAP to find smaller of two numbers 56H and 32H. The first number is stored in 2501H and second number is stored in 2502H. The result is to be stored in 2503H.

- WAP to access data 33H and CCH from 01H and 08H. Do the OR operation and transfer the resultant value to port address 03H.
- WAP to access data 32H and 45H from 2501H and 2502H, add them and rotate the result right 3 times and store the result in memory location 2503H.
- WAP to access data 33H and CCH from 2501H and 2502H. Do the AND operation and store the result in memory location 2503H.

- WAP to find product of two 8 bit numbers and the numbers are 84H and 56H.



Microprocessor Programming

- **Program** contains a sequence of instructions.
- **Software** contains a set of programs written for a particular computer.
- A computer understands information composed of zeros and ones.
- **Machine language program** contains instruction in the form of zero and ones.

10000000	Add content of registers A and B
01111000	Move content of register B to register A
01110110	Stop

Demerits of machine language program

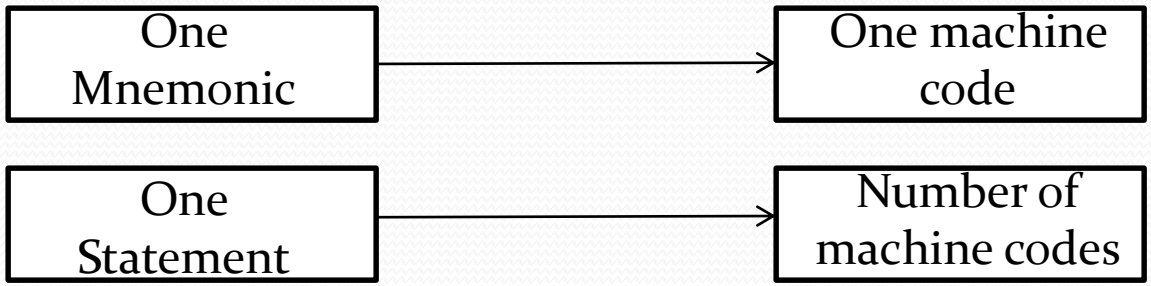
- Difficult to understand or debug a program
- Entry of a program is slow and programs are long
- Program writing is difficult and tiresome
- Chances of errors in writing the program

- To facilitate programmer machine codes are written in **hexadecimal system**. The mistakes can easily be detected.

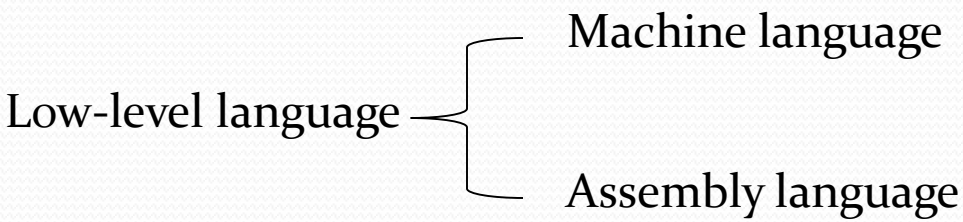
- **Source language -** Language in which programmer writes a program.
- **Object/Machine language** – Language in which a computer works.

Assembly language program

- It is one of the easily understandable language.
- Program is written in alphanumeric symbols instead of zeros and ones.
- Meaningful symbols are chosen for the purpose, ex- ADD, SUB, ORA, CMA.
- Such symbols are called **mnemonics**.
- Programs written in assembly language is easier and faster than machine language.



Low-level language- A microprocessor specific language is known as a low level language.



Assembler- A program which translates an assembly language program into a machine language program is called an assembler.

Self-assembler- It is an assembler which runs on the microcomputer for which it produces object codes.

Cross-assembler- It is an assembler that runs on a computer other than that for which it produces object codes.

- **Disassembler-** Its role is complementary to that of an assembler and converts a machine language program to an assembly language program.
- **Hand assembly-** In an ordinary kit assembler is not available. Programmer writes program in assembly language and converts it into the hexadecimal codes.
- **One-Pass assembler-** An assembler which goes through an assembly language program only once.
- **Two-Pass assembler-** An assembler which goes through an assembly language program twice is called a two-pass assembler.

✓ **First pass-** It collects all the labels

✓ **Second pass-** It produces machine codes for all instructions and assign addresses.

Advantages	Disadvantages
The computation time of assembly language is less.	1. Programming is difficult and time consuming
	2. Program is computer oriented
	3. Program is not portable
	4. Program is long

High level language

- Instructions in high level language are called **statements**.
- Statements more precisely resembles English and Mathematics than mnemonics.
- This language is procedure oriented than computer oriented.
- The programs are portable.
- One statement of high level language corresponds to many instructions of assembly language.
- Examples are FORTRAN, COBOL, BASIC, JAVA, C, etc.
- A **compiler** is needed to translate high-level language to machine codes.

- **COMPILER**
- It translates a high-level language program into a machine language.
- It is more powerful than assembler.
- It occupies more memory space and longer time to produce results.
- **Self compiler-** Compiler runs on the computer for which it produces object codes.
- **Cross compiler-** Compiler runs on the computer other than that for which it produces object codes.

- **INTERPRETER**
- It translates a high-level language into object codes, statementwise.
- It took one statement at a time, translates it and executes it.
- It is slower than a compiler.
- It occupies less memory space, it is cheaper and suitable for a smaller system.

- **FORTRAN**
- FORTRAN stands for Formula Translation.
- It is the first high level language developed in 1950s by IBM.
- It is an ideal and most popular language for scientific and engineering computation.
- Initially FORTRAN IV was popular.
- In 1977 the American National Standard Institute (ANSI) published a standard for FORTRAN.
- The standard form of language is FORTRAN 77, FORTRAN 90, FORTRAN 95.

- **BASIC**
- It stands for Beginners All-purpose Symbolic Instruction Code.
- It is simple and easy language and was introduced in 1965.
- It is suitable for scientific and engineering applications and is not as powerful as FORTRAN.
- **QBASIC** is a recent version of BASIC and was developed by Microsoft. It uses GUI (Graphical User Interface) and Interpreter.
- **VISUAL BASIC** combine small programs written in BASIC language. It is a powerful tool for developing windows applications (GUI).

- **C language**
- It was developed at AT and T's Bell laboratories in 1972 by Dennis Ritchie.
- Programs are small and concise.
- It supports library functions which are equivalent to subroutines.
- C was first used to write UNIX operating system.
- It is used to write system and commercial software packages such as operating system, compilers, spreadsheet, etc.
- **C++** is an extension of C language. It was developed in 1980 at Bell laboratories. It is an object oriented language.

- **JAVA**
- It is an object-oriented programming language derived from C++ and is developed by Sun Microsystems.
- Applications written in JAVA are capable of running on any platform.
- It is suitable for Internet, desktop, computers, servers, microcontrollers, etc.
- **Java Script** is a scripting language based on JAVA.
- **Java chips** are dedicated microprocessor optimized to run Java based business and embedded applications.

Advantages & Disadvantages of High Level Language

Advantages	Disadvantages
1. Instructions are clear	1. One has to learn special rules for writing a program in high level language
2. Writing a program is easier and faster	2. The computation time is more
3. Program is portable	3. Low efficiency of memory utilization
4. Easier documentation	4. Extensive hardware and software support is required (Cost of peripherals is high).
5. Problem oriented rather than computer oriented.	5. A compiler is required which is costly.
6. Standard syntax	

Application areas of various languages

Machine language	Assembly language	High level language
1. Small and simple programs	1. Small to moderate size programs	1. For large programs
2. Simple control applications where less computation is required	2. Real-time control applications	2. Large volume of data is to be processed
3. Applications where prototype is the final product.	3. Small volume of data to be processed	3. Large computations are required
	4. Cost of memory is a consideration	4. Applications having large memories
	5. In training kits and industrial applications.	5. Complex mathematical computations
		6. Applications where high cost is justified

In microcomputer future trends favor's high level language due to following reasons:

- Hardware and memory becoming less expensive.
- Large size of memory chips are available at low cost.
- Compilers are easily available
- Efficient high level language are developed

STACK

- Stack is 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 the content temporarily during the execution of the program.
- The beginning of the stack is defined using:

  LXI SP,……. ⟵ 16 bit address

- Once the stack location is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register.
- **PUSH** instruction is used to store the content of register pair on the stack

- **POP** instruction is used to transfer the contents from the stack to respective registers.
- The stack pointer register tracks the storage and retrieval of the information.
- The contents of the program counter can be stored when a subroutine is called.
- **Instructions**

PUSH Rp	Copy the content of the specified register pair on the stack. Rp can be B, D, H which represents the register pair BC, DE, HL.
PUSH PSW	Copy the contents of accumulator and flags on the stack
POP Rp	Copy the content of the memory locations of the stack into the specified register pair.
POP PSW	Copy the content of memory location into the registers

- The stack space grows upward in the numerically decreasing order of memory addresses.
- The storage and retrieval of data bytes on the stacks should follow the LIFO sequence.
- Information in stack locations is not destroyed until new information is stored in those locations.

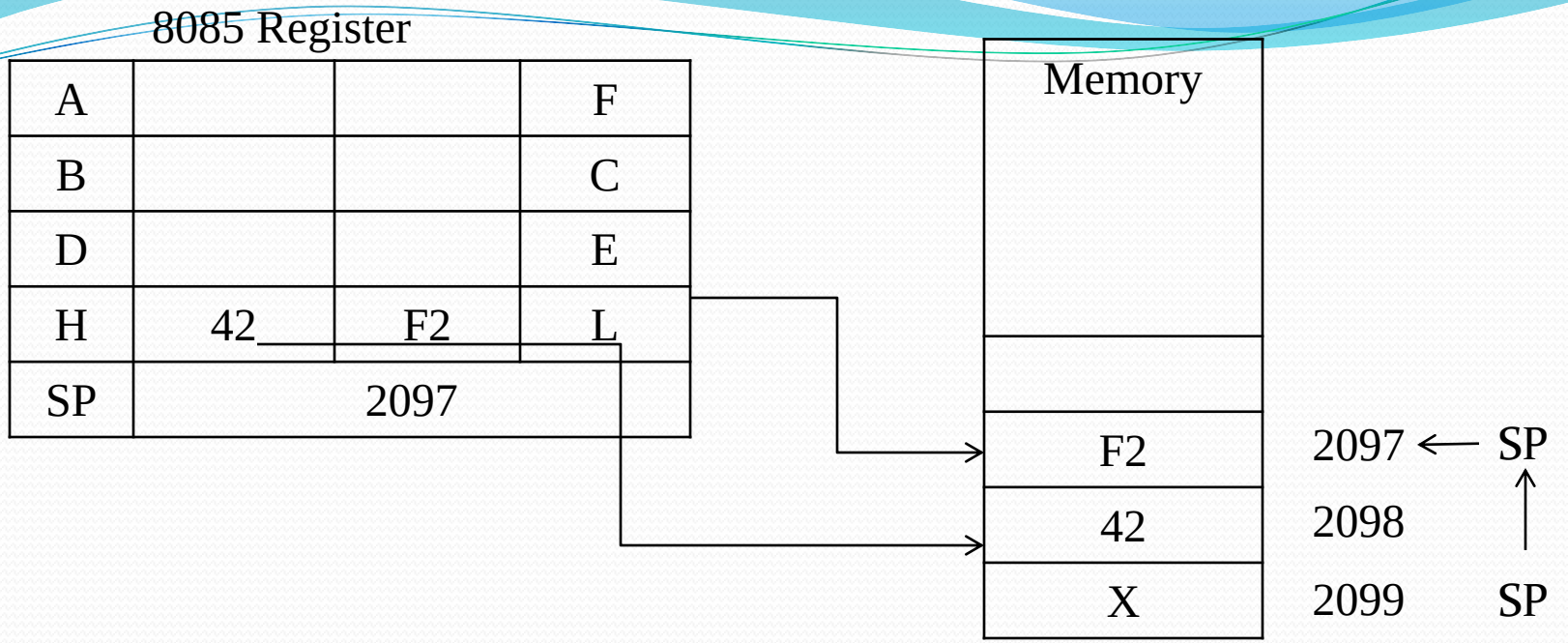
- **Example 1**

The stack pointer has to be initialized at 2099H and the content of the register pair HL (42F2H) are stored on the stack and after certain delay the content of the HL pair are retrieved from the memory location.

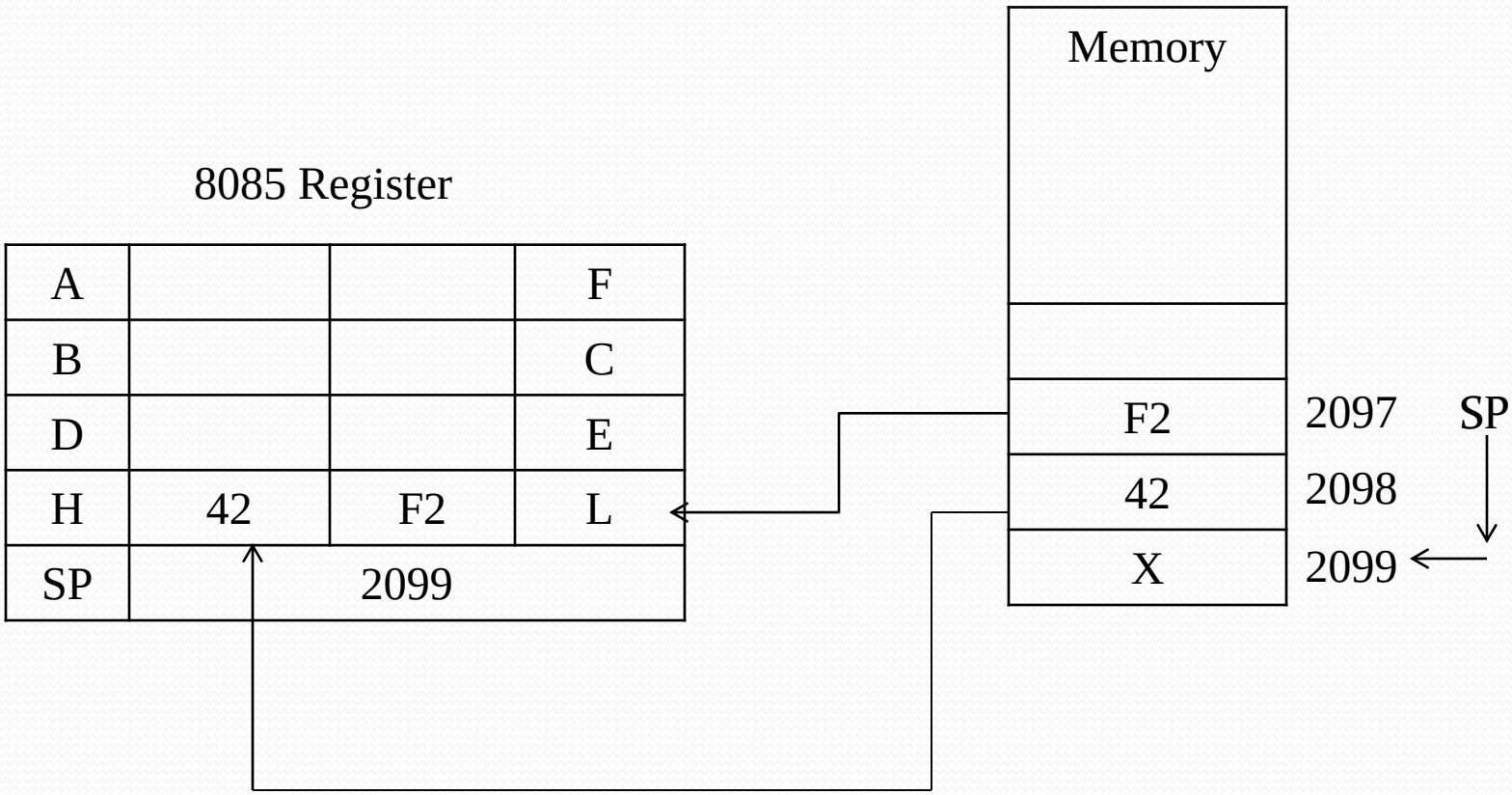
Memory Location	**Mnemonics**
2000	LXI SP, 2099H
2003	LXI H, 42F2H
2006	PUSH H
2007	Delay Counter
200F	
2010	POP H

Register Content

A			**F**
B			**C**
D			**E**
H	42	F2	**L**
SP 2099			



Content on the stack and in the Registers After the PUSH Instruction



Content on the stack and in the Registers After the POP Instruction

- **Example 2**

The stack pointer has to be initialized at 2489H and the content of the register pair HL (2150H), BC (2280H), PSW are stored on the stack and after certain delay the content of the HL, BC, PSW are retrieved from the memory location.

- WAP to add two 16 bit numbers 5B98H and 8E4CH which are stored at 2501H to 2504H. Store the results in the memory locations 2505H to 2507H.
- WAP to divide two 8 bit numbers.

SUBROUTINE

- Certain programs have operations that occur several times but they are not available as individual instructions.
- Example-**Multiplication, Square-root, etc.**
- A **subroutine** is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program.
- It avoids repetition of the smaller programs.
- Subroutine is implemented using:

❑ **Call -** Call a subroutine

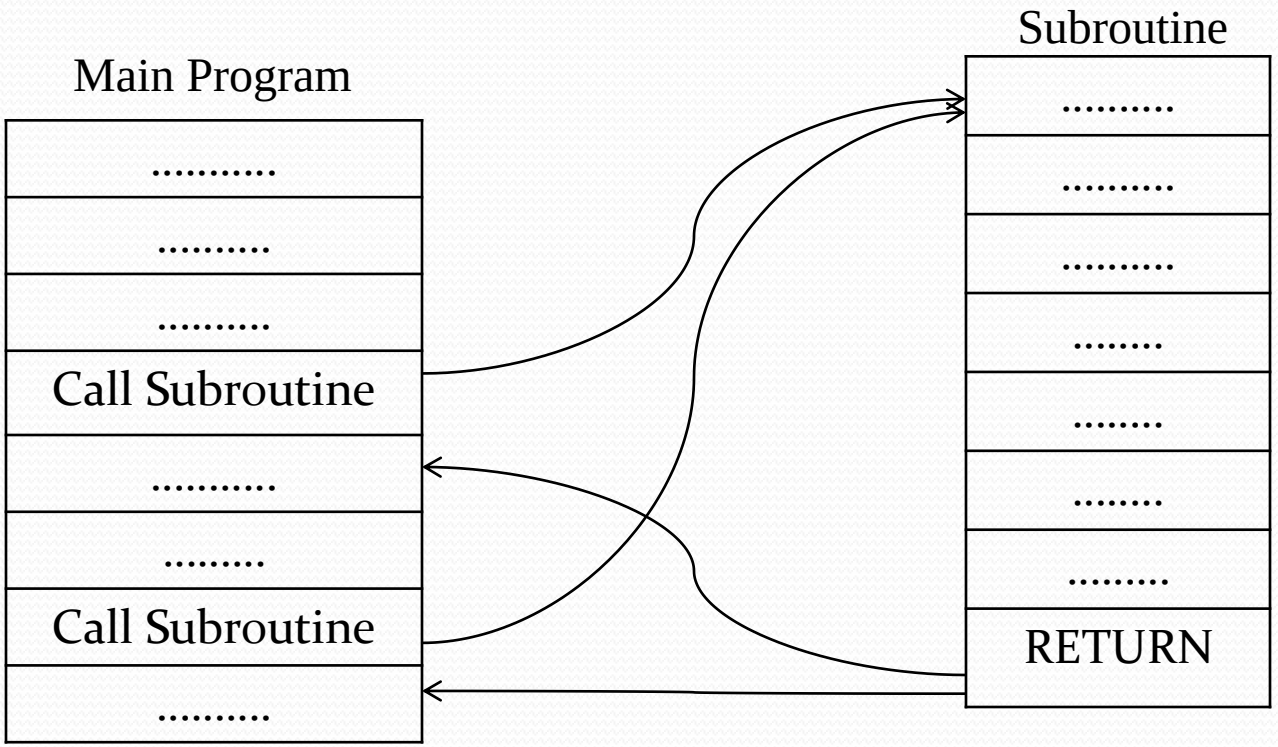
❑ **RET -** Return to main program from a subroutine

Subroutine is called
CALL

Content of the Program Counter is stored on the stack

At the end of the Subroutine
RET

Content of the Program Counter is retrieved from the stack



- **Conditional Call Instructions**
- The program is transferred to the subroutine if the condition is met otherwise the main program is continued.

CC	Call subroutine if Carry flag is set
CNC	Call subroutine if Carry flag is reset
CZ	Call subroutine if Zero flag is set
CNZ	Call subroutine if Zero flag is reset
CM	Call subroutine if Sign flag is set
CP	Call subroutine if Sign flag is reset
CPE	Call subroutine if Parity flag is set
CPO	Call subroutine if Parity flag is reset

Conditional RETURN Instructions

- The sequence returns to the main program if the condition is met, otherwise the sequence in the subroutine is continued.

RC	Return from subroutine if Carry status is set
RNC	Return from subroutine if Carry status is reset
RZ	Return from subroutine if Zero status is set
RNZ	Return from subroutine if Zero status is reset
RM	Return from subroutine if Sign status is set
RP	Return from subroutine if Sign status is reset
RPE	Return from subroutine if Parity status is set
RPO	Return from subroutine if Parity status is reset

Restart (RST) Instruction

- It is a one byte CALL instruction which transfers the program execution to a specific location.
- It is generally used in conjunction with the interrupt process.

RST 0	Call 0000H
RST 1	Call 0008H
RST 2	Call 0010H
RST 3	Call 0018H
RST 4	Call 0020H
RST 5	Call 0028H
RST 6	Call 0030H
RST 7	Call 0038H

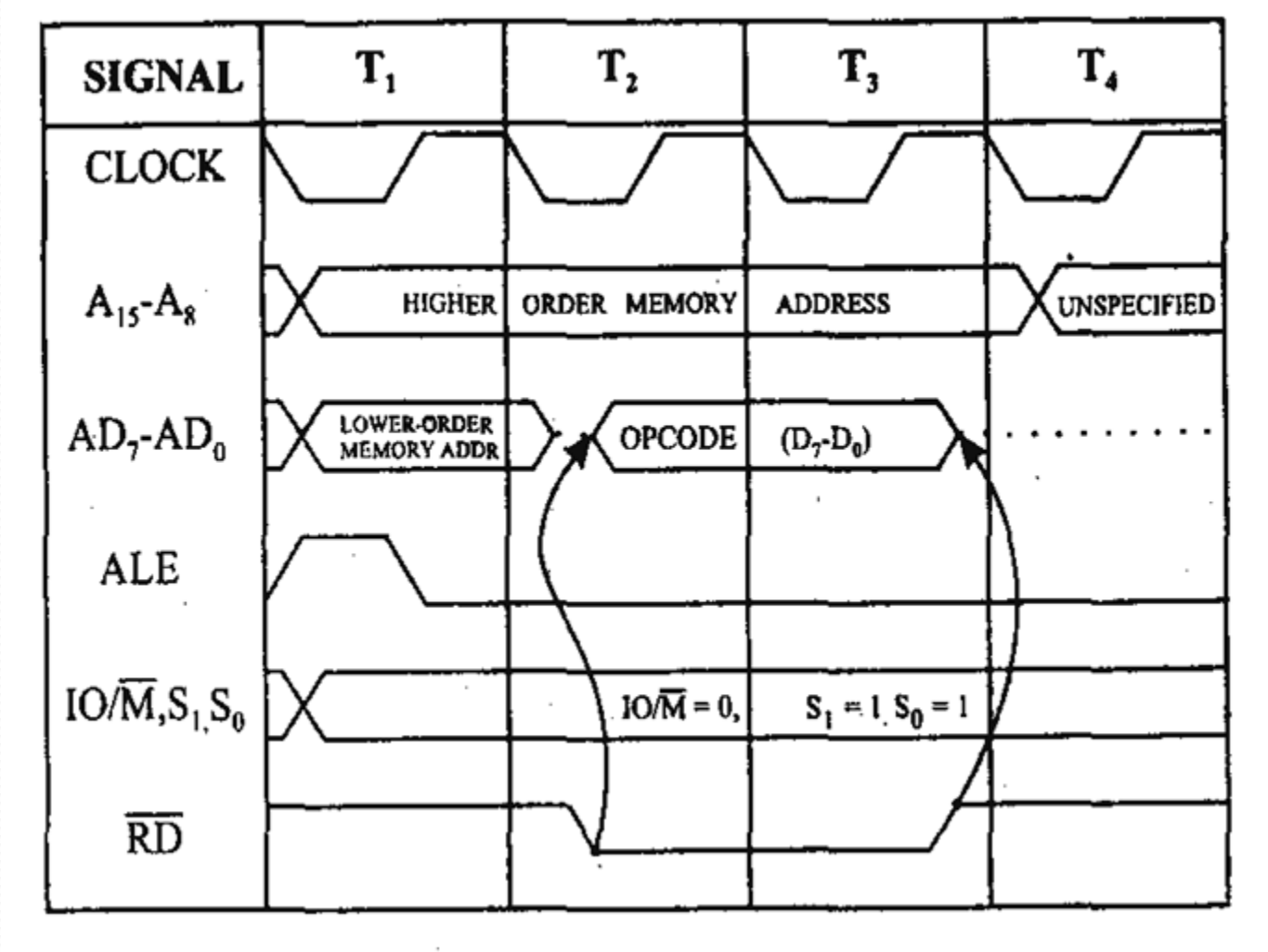
	CALL and RET	PUSH and POP
1.	When CALL is executed, the microprocessor automatically stores the 16 bit address of the instruction next to CALL on the stack	The programmer uses the instruction PUSH to save the contents of the register pair on the stack
2.	When CALL is executed, the stack pointer register is decremented by two	When PUSH is executed, the stack pointer register is decremented by two
3.	The RET instruction transfers the contents of the top two locations of the stack to the program counter	The instruction POP transfer the contents of the top two locations of the stack to the specified register pair
4.	When the instruction RET is executed, the stack pointer is incremented by two	When the instruction POP is executed the stack pointer is incremented by two
5.	There are eight conditional CALL and RETURN instruction	There are no conditional PUSH and POP instruction

Timing diagram

- It is a graphical representation of necessary steps which are carried out in a machine cycle.
- Pictorial representation of execution of an instruction with the help of various control/timing and status signals.
- Based on different sets of operations carried out in 8085 microprocessor, there are different timing diagram:

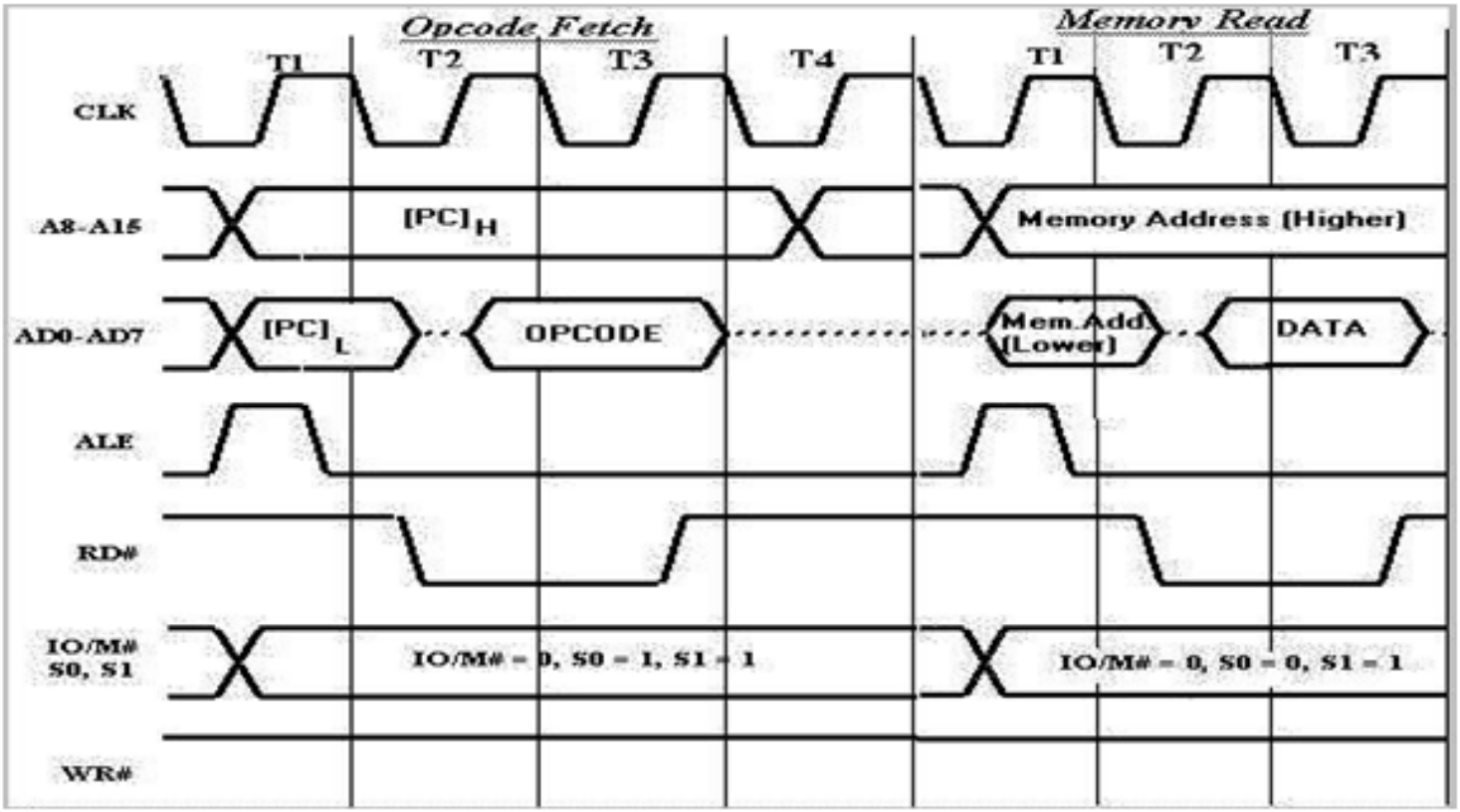
a. Opcode fetch
b. Memory read
c. Memory write
d. I/O read
e. I/O write

Timing diagram for Opcode Fetch Cycle



- **<u>Sequence of steps for opcode fetch operation</u>**
- **First clock cycle (T_1)-** Microprocessor sends the address of the memory where the opcode is available.
- **Second clock cycle (T_2) –** The control unit sends the control signal $\overline{RD}$ to the memory chip to enable the read operation. The byte is placed from the memory location on the data bus.
- **Third clock cycle (T_3) –** The opcode is placed in the IR.
- **Fourth clock cycle (T_4)-** The opcode is decoded.

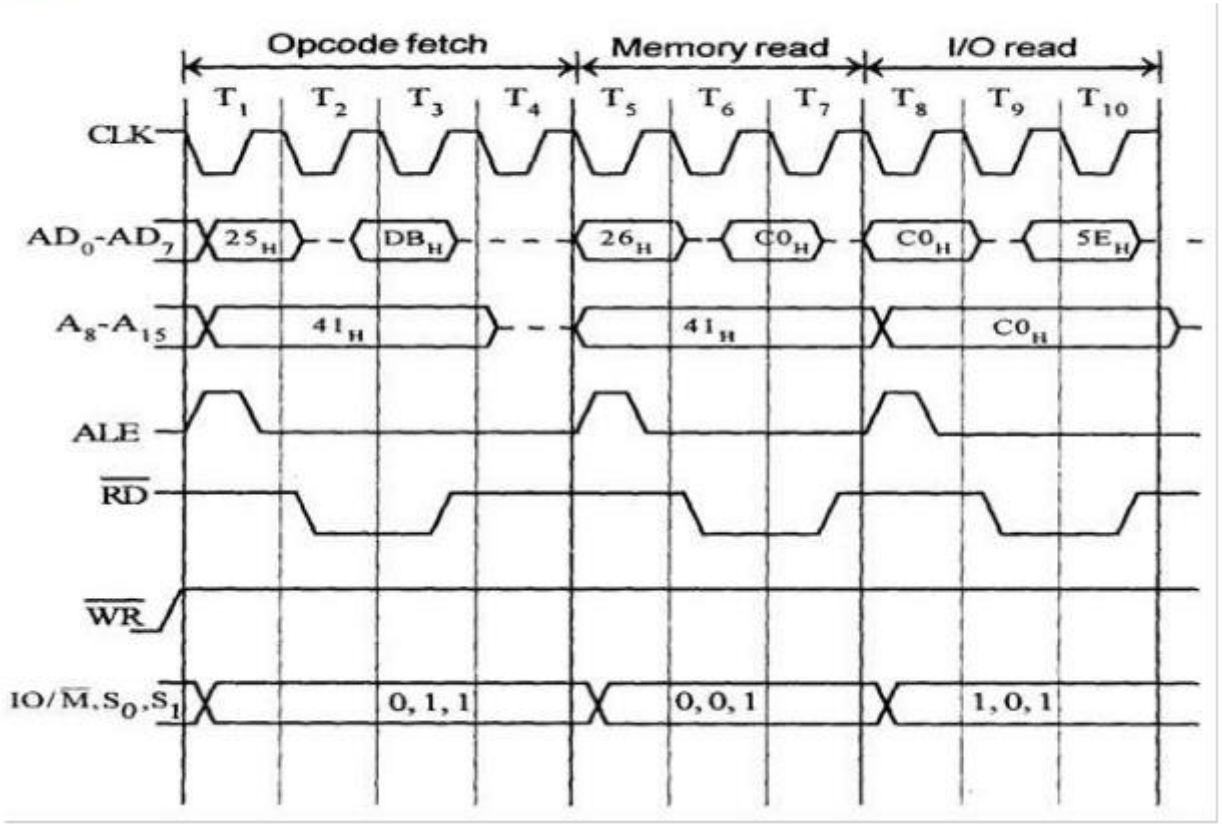
Timing diagram for MVI A, Data



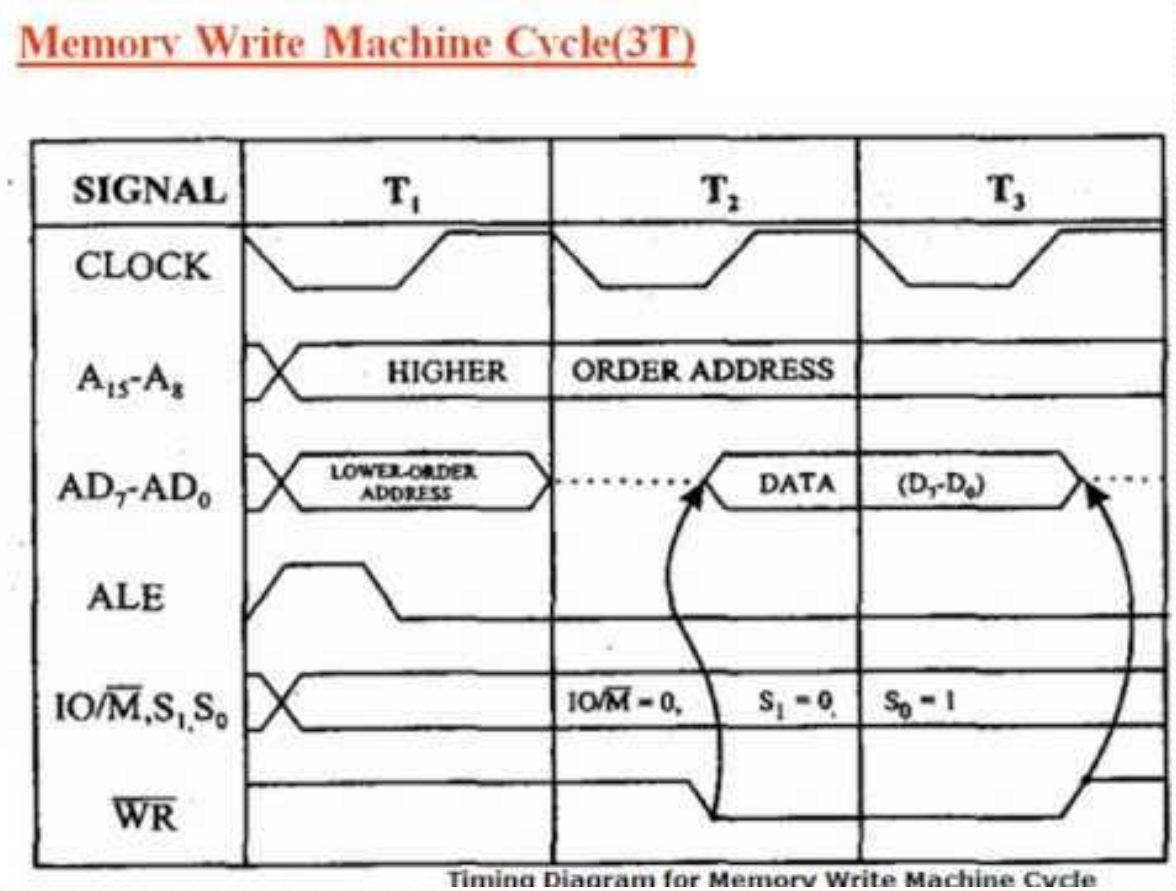
- **Sequence of steps for memory read operation**
- **First clock cycle (T_1)-** Microprocessor sends the address of the memory where the data is available.
- **Second clock cycle (T_2) –** The control unit sends the control signal $\overline{RD}$ to the memory chip to enable the read operation. The byte is placed from the memory location on the data bus.
- **Third clock cycle (T_3) –** The data enters the CPU.

Timing diagram for Opcode fetch, Memory & IO read operation

Eg:



Timing diagram for memory write Operation



- **Sequence of steps for memory write operation**
- **First clock cycle (T_1)-** Microprocessor sends the address of the memory where the data is to be stored.
- **Second clock cycle (T_2) –** The control unit sends the control signal $\overline{WR}$ to the memory chip to enable the write operation.
- **Third clock cycle (T_3) –** The byte is placed from the microprocessor on the data bus and stored in memory.