

# ASSEMBLY LANGUAGE PROGRAMMING OF 8085



Presentation By: Shehrevan Davierwala

Contact : [shehrevan@live.com](mailto:shehrevan@live.com)

Lecturer at GES'S Katgara Polytechnic Institute

# TOPICS

1. Introduction
2. Programming model of 8085
3. Instruction set of 8085
4. Example Programs
5. Addressing modes of 8085
6. Instruction & Data Formats of 8085



## 1. INTRODUCTION

- A **microprocessor** executes instructions given by the user
- Instructions should be in a language known to the **microprocessor**
- **Microprocessor** understands the language of 0's and 1's only
- This language is called **Machine Language**



- For e.g.

01001111

- Is a valid machine language instruction of 8085
- It copies the contents of one of the internal registers of 8085 to another



# A MACHINE LANGUAGE PROGRAM TO ADD TWO NUMBERS

00111110

A

00000010

00000110

B

00000100

10000000

;Copy value 2H in register

;Copy value 4H in register

;A = A + B



# ASSEMBLY LANGUAGE OF 8085

- It uses English like words to convey the action/meaning called as MNEMONICS
- For e.g.
  - MOV to indicate data transfer
  - ADD to add two values
  - SUB to subtract two values



# 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
```

Note:

- Assembly language is specific to a given processor
- For e.g. assembly language of 8085 is different than that of Motorola 6800 microprocessor





## MICROPROCESSOR UNDERSTANDS MACHINE LANGUAGE ONLY!

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



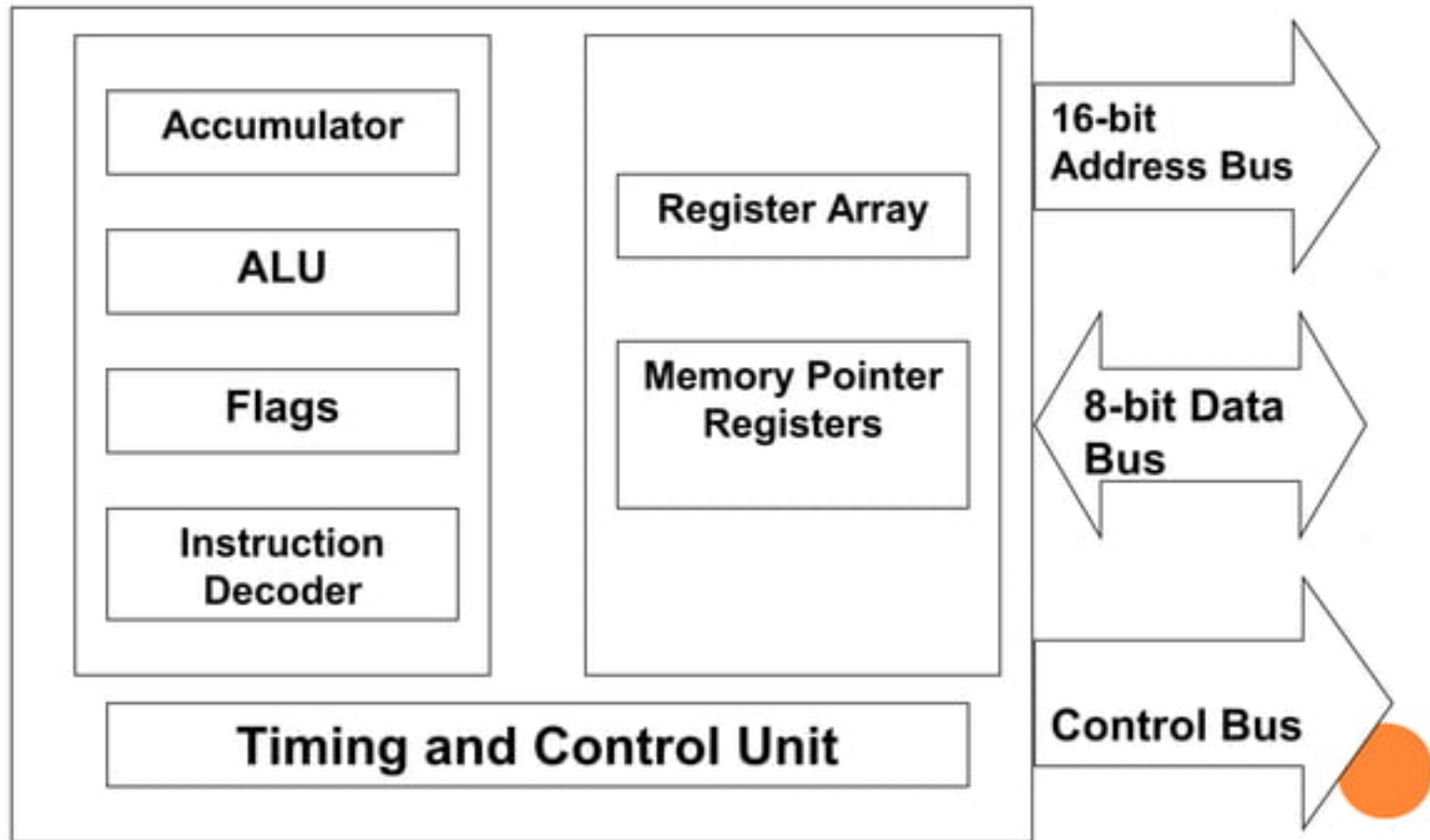


# LOW-LEVEL/HIGH-LEVEL LANGUAGES

- Machine language and Assembly language are both
  - Microprocessor specific (**Machine dependent**)  
so they are called
  - Low-level languages
- **Machine independent** languages are called
  - High-level languages
  - For e.g. BASIC, PASCAL, C++, C, JAVA, etc.
  - A software called **Compiler** is required to convert a high-level language program to machine code



## 2. PROGRAMMING MODEL OF 8085



Accumulator (8-bit)

Flag Register (8-bit)

S Z AC P CY

B (8-bit)

C (8-bit)

D (8-bit)

E (8-bit)

H (8-bit)

L (8-bit)

Stack Pointer (SP) (16-bit)

Program Counter (PC) (16-bit)

8- Lines

Bidirectional



16- Lines

Unidirectional



# OVERVIEW: 8085 PROGRAMMING MODEL

1. Six general-purpose Registers
2. Accumulator Register
3. Flag Register
4. Program Counter Register
5. Stack Pointer Register



## 1. **Six general-purpose registers**

- B, C, D, E, H, L
- Can be combined as register pairs to perform 16-bit operations (BC, DE, HL)

## 1. **Accumulator – identified by name A**

- This register is a part of ALU
- 8-bit data storage
- Performs arithmetic and logical operations
- Result of an operation is stored in accumulator



### 3. **Flag Register**

- This is also a part of ALU
- 8085 has five flags named
  - **Zero** flag (Z)
  - **Carry** flag (CY)
  - **Sign** flag (S)
  - **Parity** flag (P)
  - **Auxiliary Carry** flag (AC)



- These flags are five flip-flops in flag register
- Execution of an arithmetic/logic operation can **set** or **reset** these flags
- Condition of flags (set or reset) can be tested through software instructions
- 8085 uses these flags in decision-making process





#### 4. Program Counter (PC)

- A 16-bit memory pointer register
- Used to sequence execution of program instructions
- Stores address of a memory location
  - where next instruction byte is to be fetched by the 8085
- when 8085 gets busy to fetch current instruction from memory
  - PC is incremented by one
  - PC is now pointing to the address of next instruction



## 5. **Stack Pointer Register**

- a 16-bit memory pointer register
- Points to a location in **Stack** memory
- Beginning of the stack is defined by loading a 16-bit address in stack pointer register



### 3.INSTRUCTION SET OF 8085

- Consists of
  - 74 operation codes, e.g. MOV
  - 246 Instructions, e.g. MOV A,B
- 8085 instructions can be classified as
  1. Data Transfer (Copy)
  2. Arithmetic
  3. Logical and Bit manipulation
  4. Branch
  5. Machine Control



# 1. DATA TRANSFER (COPY) OPERATIONS

1. **Load** a 8-bit number in a **Register**
2. **Copy** from **Register** to **Register**
3. **Copy** between **Register** and **Memory**
4. **Copy** between **Input/Output** Port and **Accumulator**
5. **Load** a 16-bit number in a **Register** pair
6. **Copy** between **Register** pair and Stack **memory**



# (COPY) OPERATIONS / INSTRUCTIONS

1. **Load** a 8-bit number 4F in register **B**
2. **Copy** from Register **B** to Register **A**
3. **Load** a 16-bit number 2050 in Register pair **HL**
4. **Copy** from Register **B** to **Memory** Address 2050
5. **Copy** between **Input/Output** Port and **Accumulator**

**MVI B, 4FH**

**MOV A,B**

**LXI H, 2050H**

**MOV M,B**

**OUT 01H**

**IN 07H**



## 2. ARITHMETIC OPERATIONS

1. **Addition** of two 8-bit numbers
2. **Subtraction** of two 8-bit numbers
3. **Increment/ Decrement** a 8-bit number



# EXAMPLE ARITHMETIC OPERATIONS / INSTRUCTIONS

1. **Add** a 8-bit number 32H to  
Accumulator

ADI 32H

2. **Add** contents of Register B to  
Accumulator

ADD B

3. **Subtract** a 8-bit number 32H  
from Accumulator

SUI 32H

4. **Subtract** contents of Register C  
from Accumulator

SUB C

5. **Increment** the contents of  
Register D by 1

INR D

6. **Decrement** the contents of  
Register E by 1

DCR E





# 3. LOGICAL & BIT MANIPULATION OPERATIONS

1. **AND** two 8-bit numbers
2. **OR** two 8-bit numbers
3. **Exclusive-OR** two 8-bit numbers
4. **Compare** two 8-bit numbers
5. **Complement**
6. **Rotate** Left/Right Accumulator bits



# MANIPULATION OPERATIONS / INSTRUCTIONS

- |  |              |
|--|--------------|
| 1. Logically <b>AND</b> Register <b>H</b> with <b>A</b> ccumulator       | <b>ANA H</b> |
| 2. Logically <b>OR</b> Register <b>L</b> with <b>A</b> ccumulator        | <b>ORA L</b> |
| 3. Logically <b>XOR</b> Register <b>B</b> with <b>A</b> ccumulator       | <b>XRA B</b> |
| 4. <b>Compare</b> contents of Register <b>C</b> with <b>A</b> ccumulator | <b>CMP C</b> |
| 5. <b>Complement</b> <b>A</b> ccumulator                                 | <b>CMA</b>   |
| 6. <b>Rotate</b> <b>A</b> ccumulator Left                                | <b>RAL</b>   |



## 4. BRANCHING OPERATIONS

These operations are used to control the flow of program execution

### 1. Jumps

- Conditional jumps
- Unconditional jumps

### 1. Call & Return

- Conditional Call & Return
- Unconditional Call & Return



# EXAMPLE BRANCHING OPERATIONS / INSTRUCTIONS

1. **Jump** to a 16-bit Address 2080H if Carry flag is **SET**
2. Unconditional **Jump**
3. **Call** a subroutine with its 16-bit Address
4. **Return back** from the Call
5. **Call** a subroutine with its 16-bit Address if Carry flag is **RESET**
6. **Return** if Zero flag is **SET**

**JC 2080H**

**JMP 2050H**

**CALL 3050H**

**RET**

**CNC 3050H**

**RZ**



## 5. MACHINE CONTROL INSTRUCTIONS

These instructions affect the operation of the processor. For e.g.

HLT	Stop program execution
NOP	Do not perform any operation



## 4. WRITING A ASSEMBLY LANGUAGE PROGRAM

- Steps to write a program
  - **Analyze** the problem
  - Develop program **Logic**
  - Write an **Algorithm**
  - Make a **Flowchart**
  - Write program **Instructions** using Assembly language of 8085



# LANGUAGE TO ADD TWO 8-BIT NUMBERS AND STORE 8-BIT RESULT IN REGISTER C.

## 1. Analyze the problem

- Addition of two 8-bit numbers to be done

## 1. Program Logic

- Add two numbers
- Store result in register C
- Example

10011001	(99H)	A
+00111001	(39H)	D
11010010	(D2H)	C





### 3. ALGORITHM

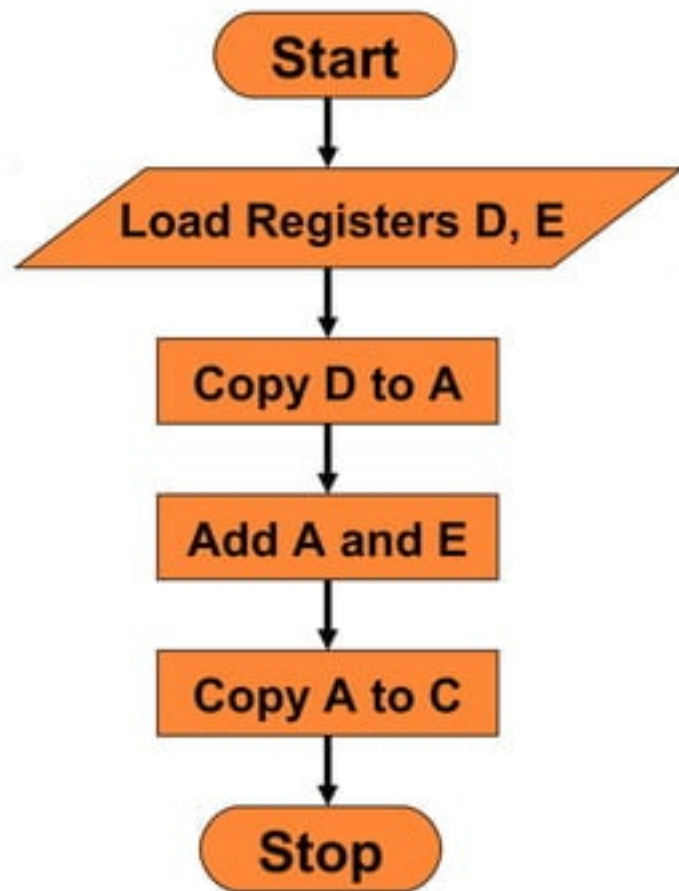
1. Get two numbers
2. Add them
3. Store result
4. Stop

### Translation to 8085 operations

- Load 1<sup>st</sup> no. in register D
- Load 2<sup>nd</sup> no. in register E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Stop processing



## 4. MAKE A FLOWCHART



- Load 1<sup>st</sup> no. in register D
- Load 2<sup>nd</sup> no. in register E

- Copy register D to A
- Add register E to A

- Copy A to register C

- Stop processing

## 5. ASSEMBLY LANGUAGE PROGRAM

1. Get two numbers

- a) Load 1<sup>st</sup> no. in register D
- b) Load 2<sup>nd</sup> no. in register E

2. Add them

- a) Copy register D to A
- b) Add register E to A

3. Store result

- a) Copy A to register C

4. Stop

- a) Stop processing

```
MVI D, 2H  
MVI E, 3H
```

```
MOV A, D  
ADD E
```

```
MOV C, A
```

```
HLT
```

LANGUAGE TO ADD TWO 8-BIT NUMBERS. RESULT CAN BE MORE THAN 8-BITS.

1. Analyze the problem

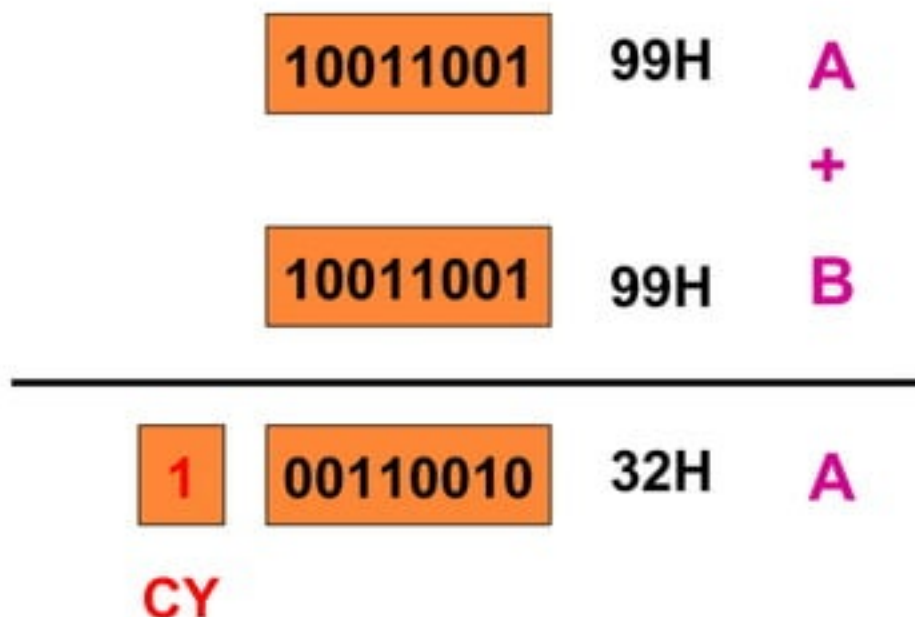
- Result of addition of two 8-bit numbers can be 9-bit
- Example

```
      10011001 (99H) A
    +10011001 (99H) B
    100110010 (132H)
```

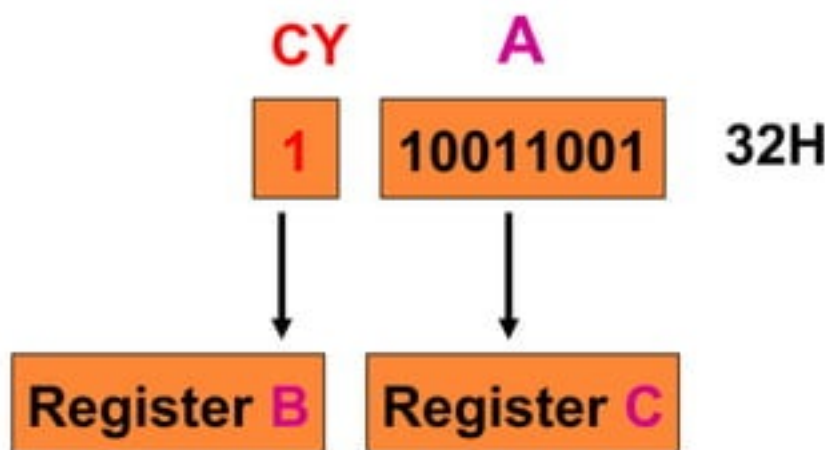
- The 9<sup>th</sup> bit in the result is called CARRY bit.



- How 8085 does it?
  - Adds register **A** and **B**
  - Stores 8-bit result in **A**
  - SETS carry flag (CY) to indicate carry bit



- Storing result in Register memory



Step-1 Copy **A** to **C**

Step-2

- Clear register **B**
- Increment **B** by 1



## 2. PROGRAM LOGIC

1. Add two numbers
2. Copy 8-bit result in A to C
3. If CARRY is generated
  - Handle it
1. Result is in register pair BC





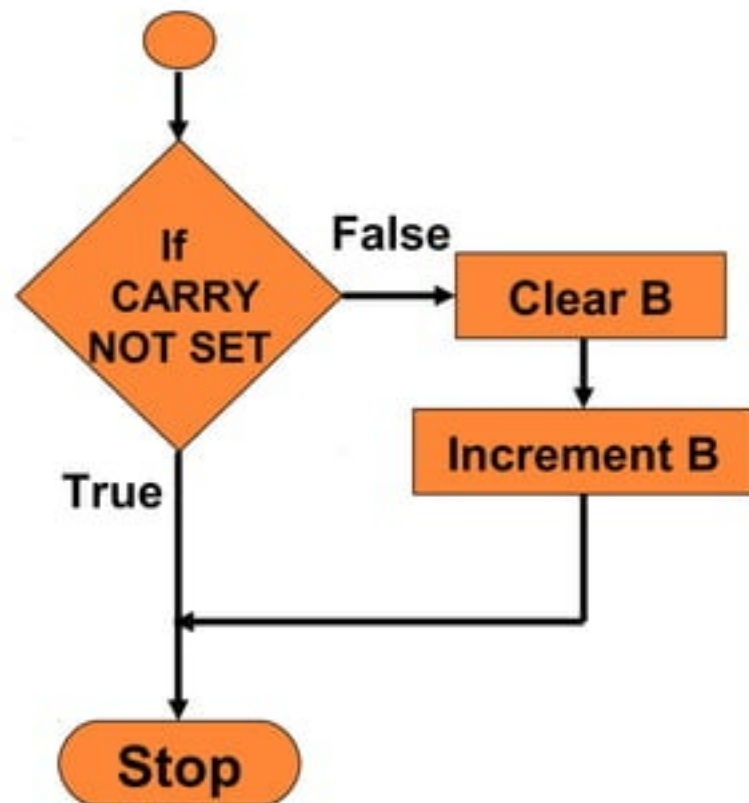
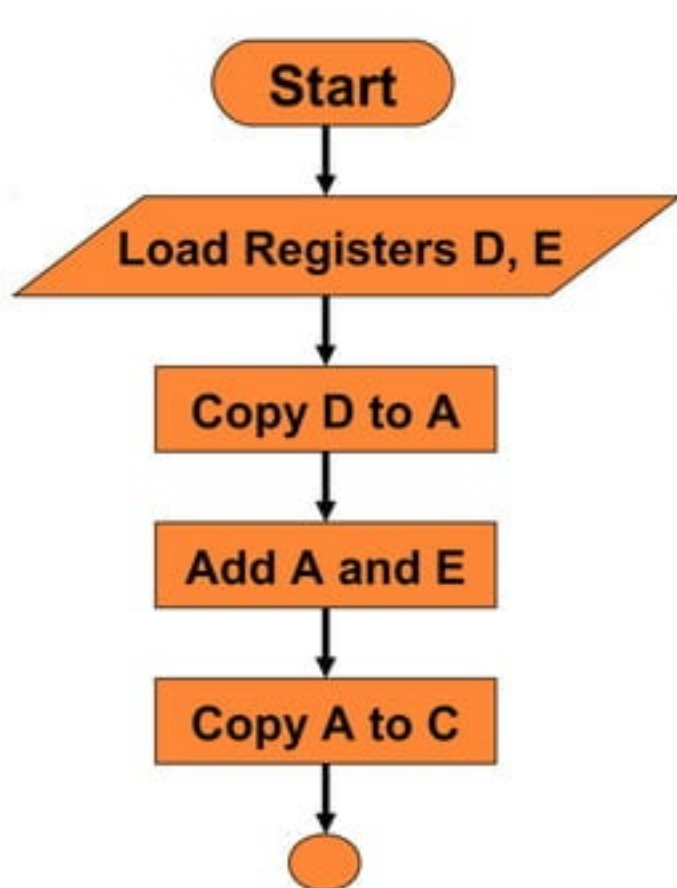
### 3. ALGORITHM

1. Load two numbers in registers D, E
2. Add them
3. Store 8 bit result in C
4. Check CARRY flag
5. If CARRY flag is SET
  - Store CARRY in register B
4. Stop

### Translation to 8085 operations

- Load registers D, E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Use Conditional Jump instructions
- Clear register B
- Increment B
- Stop processing

## 4. MAKE A FLOWCHART



## 5. ASSEMBLY LANGUAGE PROGRAM

- Load registers D, E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Use Conditional Jump instructions
- Clear register B
- Increment B
- Stop processing

```
MVI D, 2H
MVI E, 3H
MOV A, D
ADD E
MOV C, A
JNC END
MVI B, 0H
INR B
END: HLT
```

## 4. ADDRESSING MODES OF 8085

- Format of a typical Assembly language instruction is given below-

[Label:] Mnemonic [Operands] [;comments]

HLT

MVI A, 20H

MOV M, A ;Copy A to memory location

whose address is stored in  
register pair HL

LOAD: LDA 2050H ;Load A with contents of memory  
location with address 2050H

READ: IN 07H ;Read data from Input port with  
address 07H

- The various formats of specifying operands are called addressing modes
- Addressing modes of 8085
  1. **Register** Addressing
  2. **Immediate** Addressing
  3. **Memory** Addressing
  4. **Input/Output** Addressing



## 1. REGISTER ADDRESSING

- Operands are one of the internal registers of 8085
- Examples-

MOV A, B

ADD C





## 2. IMMEDIATE ADDRESSING

- Value of the operand is given in the instruction itself
- Example-

**MVI A, 20H**

**LXI H, 2050H**

**ADI 30H**

**SUI 10H**



### 3. MEMORY ADDRESSING

- One of the operands is a memory location
- Depending on how address of memory location is specified, **memory** addressing is of two types
  - **Direct** addressing
  - **Indirect** addressing





### 3(A) **DIRECT** ADDRESSING

- **16-bit** Address of the memory location is specified in the instruction directly
- Examples-

**LDA 2050H** ;load A with contents of  
memory location with address  
2050H

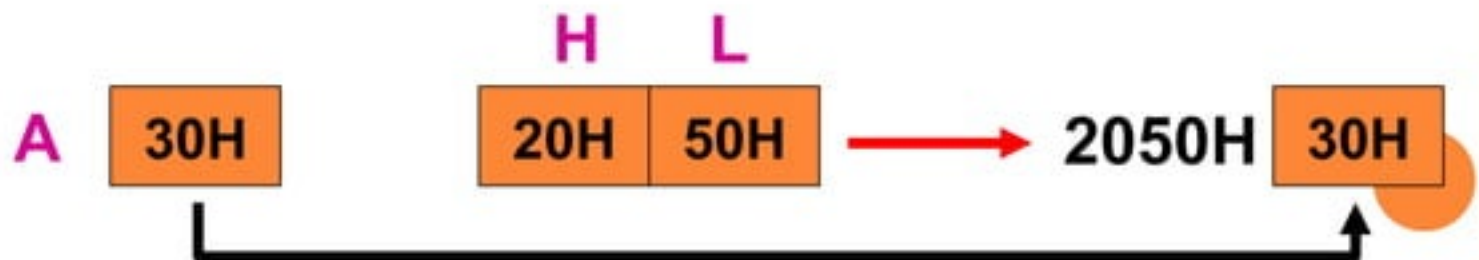
**STA 3050H** ;store A with contents of  
memory location with address  
3050H



### 3(B) INDIRECT ADDRESSING

- A **memory pointer** register is used to store the address of the memory location
- Example-

**MOV M, A** ; copy register A to memory location  
whose address is stored in  
register pair HL



## 4. INPUT/OUTPUT ADDRESSING

- 8-bit address of the port is directly specified in the instruction
- Examples-

**IN 07H**

**OUT 21H**



## 5. INSTRUCTION & DATA FORMATS

8085 Instruction set can be classified according to size (in bytes) as

1. 1-byte Instructions
2. 2-byte Instructions
3. 3-byte Instructions



## 1. ONE-BYTE INSTRUCTIONS

- Includes Opcode and Operand in the same byte
- Examples-

Opcode	Operand	Binary Code	Hex Code
MOV	C, A	0100 1111	4FH
ADD	B	1000 0000	80H
HLT		0111 0110	76H



## 1. TWO-BYTE INSTRUCTIONS

- First byte specifies Operation Code
- Second byte specifies Operand
- Examples-

Opcode	Operand	Binary Code	Hex Code
MVI	A, 32H	0011 1110	3EH
		0011 0010	32H
MVI	B, F2H	0000 0110	06H
		1111 0010	F2H



## 1. THREE-BYTE INSTRUCTIONS

- First byte specifies Operation Code
- Second & Third byte specifies Operand
- Examples-

Opcode	Operand	Binary Code	Hex Code
LXI	H, 2050H	0010 0001	21H
		0101 0000	50H
		0010 0000	20H
LDA	3070H	0011 1010	3AH
		0111 0000	70H
		0011 0000	30H





# HEXADECIMAL NUMBERS AND STORE IT IN TWO DIFFERENT LOCATIONS

- LDA 2200H ; Get the packed BCD number
- ANI F0H ; Mask lower nibble

0100 0101 45

1111 0000 F0

-----

0100 0000 40

- RRC
- RRC
- RRC ; Adjust higher digit as a lower digit.
- RRC 0000 0100 after 4 rotations





## CONTD.

- STA 2300H ; Store the partial result
- LDA 2200H ; Get the original BCD no.
- ANI 0FH; Mask higher nibble

0100 0100	45
0000 1111	0F
-----	
0000 0100	05

- STA 2301H ; Store the result
- HLT ; Terminate program execution



## BLOCK DATA TRANSFER

- `MVI C, 0AH` ; Initialize counter i.e no. of bytes  
Store the count in Register C, ie ten
  - `LXI H, 2200H` ; Initialize source memory pointer  
Data Starts from 2200 location
  - `LXI D, 2300H` ; Initialize destination memory pointer
- BK: `MOV A, M` ; Get byte from source memory block  
i.e 2200 to accumulator.
- `STAX D` ; Store byte in the destination memory block i.e 2300 as stored in D-E pair

## CONTD.

- INX H ; Increment source memory pointer
- INX D ; Increment destination memory pointer
- DCR C ; Decrement counter  
to keep track of bytes moved
- JNZ BK ; If counter 0 repeat steps
- HLT ; Terminate program

