Microprocessor and Computer Architecture

Diploma in Computer Engineering

Er.Ganga Gautam

CHAPTER THREE:

8085 INSTRUCTION SET

OUTLINES

3.1. Machine language instruction format:

- 3.1.1. Single bytes
- 3.1.2. Two bytes
- 3.1.3. Three bytes instructions

3.2. Various addressing modes

3.3. Data transfer operation and instruction

3.4. Arithmetic operation and instruction

3.5. Logical operation and instruction

3.6. Branch operation and instruction

3.7. Stack operation and instruction

3.8. Input/output and machine control operation and instruction

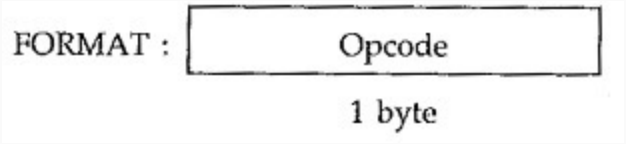
3.9. Simple programs with 8085 instructions

Instruction Format

- The Instruction Format of 8085 set consists of one-, two- and three-byte instructions.
- The first byte is always the mnemonics (opcode); in two-byte instructions the second byte is usually data; in three-byte instructions the last two bytes present address or 16-bit data.
- Types of instruction on the terms of byte size or word size:

1. One byte instruction
2. Two byte instruction
3. Three byte instruction

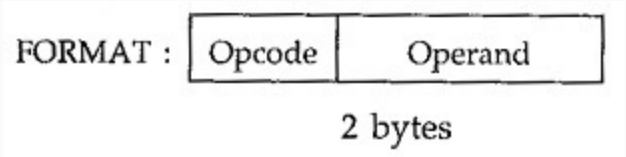
One-byte instruction



- includes an opcode and an operand in the same byte.
- Operand(s) are internal registers and are in the instruction
- no numeral present in the instruction
- for example:

➢ CMA Hex code = 2FH(one byte)
➢ ADD B Hex code = 80H (one byte)
➢ MOV C, A Hex code = 4FH (one byte)

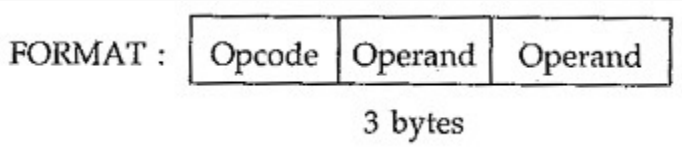
Two byte instruction



- instruction contains opcode in the first byte and eight-bit operand on the second byte.
- The source operand is a data byte and immediately following the opcode.
- 8-bit numeral is present in the instruction (the numeral may be a data or an address)
- For example,

➢ MVI A, 09H Hex code = 3E, 09 (two bytes)
➢ ADD B, 07H Hex code = 80, 07 (two bytes)
➢ SUB A, 05H Hex code = 97, 05 (two bytes)

Three- byte instruction



- instruction have opcode in the first byte and 16- bit operand is contained in the following second or third byte of the instruction.
- a 16-bit numeral is present in the instruction (the numeral may be a data or an address)
- the second byte includes low-order address and the third byte includes high-order address.
- for example:

➢ LXI H, 8509 Hex code = 21, 09, 85 (Three bytes)
➢ LDA 8509 Hex code = 3A, 09, 85 (Three bytes)
➢ JMP 9567 Hex code = C3, 67, 95 (Three bytes)
➢ STA 3525 Hex code = 32, 35, 25 (Three bytes)

Addressing Modes

- Addressing modes define how an instruction in a microprocessor accesses operands or data.
- It is a way of specifying operands to the instruction.
- The 8085 microprocessor supports various addressing modes to facilitate efficient programming.

Types of Addressing Modes

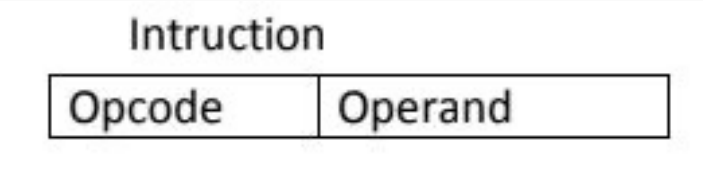
1. Implied Addressing mode
2. Immediate Addressing mode
3. Direct Addressing mode
4. Indirect Addressing mode
5. Register Addressing mode
6. Register Indirect Addressing mode
7. Displacement Addressing mode

1. **Implied Addressing mode**
   - Address of the operands are specified implicitly in the definition of instruction.
   - No need to specify the address in the instruction.
     e.g. CMA
   – Complement the content of accumulator

2. Immediate Addressing

- the operand is a constant value or data.
- Instead of specifying the address of operand, Operand itself is specified in the instruction.
- Fast to acquire an operand
- e.g. ADD 05H

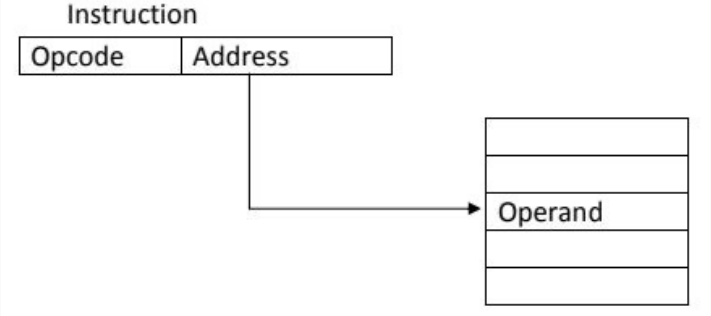
— Add 05H to contents of accumulator



3. Direct Addressing

- In direct addressing mode, the operand is a memory location.
- Instruction specifies the memory address which can be used directly to access the memory.
- Address field contains address of operand
- Example: MOV A, M

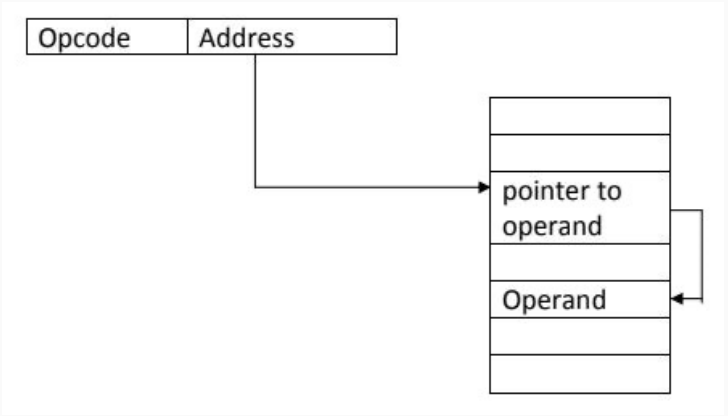
- Move the value stored at the memory location specified by HL into the accumulator



4. Indirect Addressing

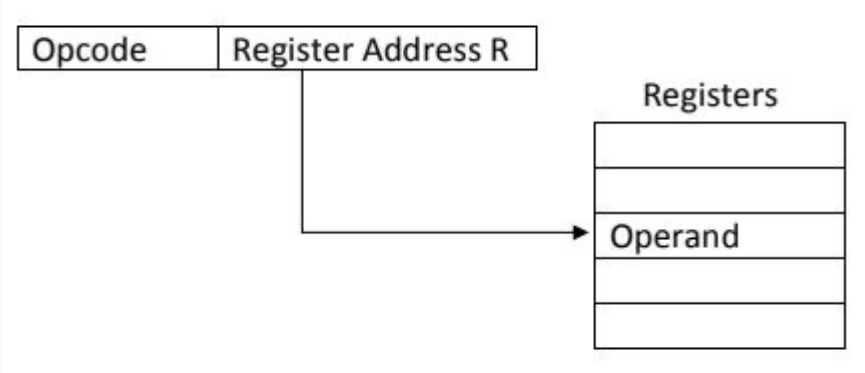
- In indirect addressing mode, the operand is a memory location pointed by a register pair.
- The address is obtained by combining the values of two registers.
- Example: LXI H, 2030H; MOV A, M

Load HL register pair with 2030H, then move the value stored at the memory location specified by HL into the accumulator.



5. Register Addressing Mode

- In register addressing mode, the operand is a register. Shorter address than the memory address.
- Faster to acquire an operand than the memory addressing.
- The 8085 has various registers like Accumulator (A), B, C, D, E, H, and L.
- Example: ADD B (Add the value in register B to the accumulator)



6. Register Indirect Addressing

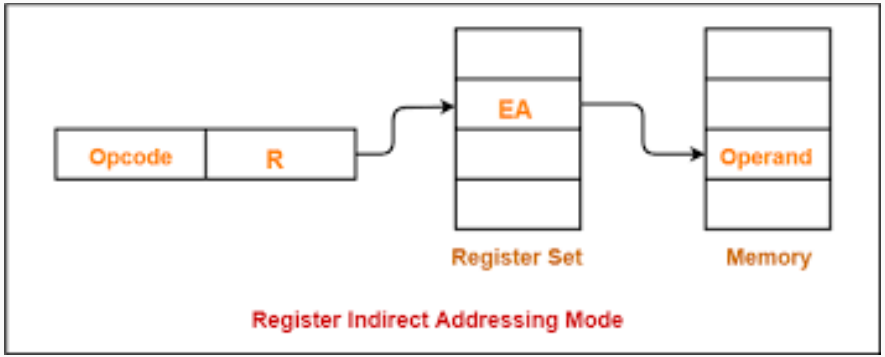
• Instruction specifies a register which contains the memory address of the operand.

• Saving instruction bits since register address is shorter than the memory address.

•Slower to acquire an operand than both the register addressing or memory addressing.

Eg: MOV AX, [BX] ;

- Suppose the register BX contains 4895H, then the contents of 4895H are moved to AX



Register Indirect Addressing Mode

7. Displacement Addressing

- In indexed addressing mode, the operand is a memory location obtained by adding an index register to a base register.

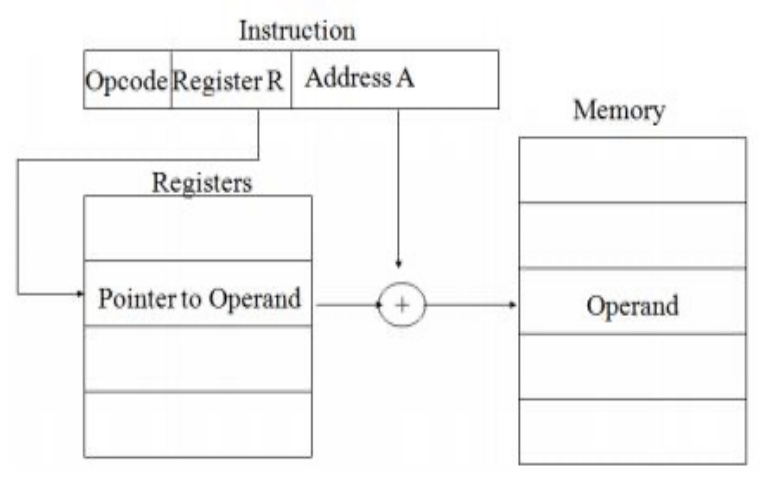
  EA = A + (R)

- Address field hold two values

— A = base value

— R = register that holds displacement

- The address field of an instruction specifies the part of address which can be used along with a designated register to calculate the address of the operand.
- Example: MOV A, (B + C) (Move the value stored at the memory location specified by the sum of registers B and C into the accumulator)



Intel 8085 instructions

- Instruction is a command given to the microprocessor to perform some specific task.
- 8085 has 246 instructions.
- Each instruction (instruction format) is of two parts: opcode and operand
- Operands or data can be specified in different ways, includes an 8-bit or 16-bit data, an internal register, a memory location, or it or 16-bit address.
- In some instructions, the operand is implicit.

Operation and Opcode

- The **opcode** is a binary code that specifies the operation to be performed by the processor. Eg: arithmetic operation like addition, subtraction, or multiplication and logical operations: AND or OR, or memory-related operations like load or store.
- The **operand** is a value or memory address that specifies the data to be operated on by the opcode. Eg: in an addition operation, the operands would be the two values to be added together.

Instruction Set

- An instruction set is collection of the different binary patterns designed inside a microprocessor to perform the specific function.
- The entire group of instruction that microprocessor supports is called instruction set.

Classification of Instruction Set

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions

1. Data Transfer Instructions

- These instructions move data between registers, or between memory and registers.
- These helps in transferring the data from source operand to destination operand (no change in original data)

Mnemonics	Example	Description
MVI R, 8-bit data	MVI B, 4FH	Load 8-bit data in the register R
MOV Rd, Rs	MOV B, A	Copy data from source register Rs to destination register Rd
LXI Rp, 16-bit data	LXI B, 2050H	Load 16-bit number in register pair Rp
LDA 16-bit	LDA 2050H	Copy the data byte from memory into Accumulator
STA 16-bit	STA 2070H	Copy the data byte into memory from Accumulator
LDAX Rp	LDAX B	Copy the data byte from register pair into Accumulator.
STAX Rp	STAX D	Copy the data type from Accumulator into register pair
MOV R, M	MOV B, M	Copy the data from memory to register R
MOV M, R	MOV M, C	Copy the data byte from register R to memory
OUT 8-bit	OUT 01H	Send the data byte from Accumulator to an output device
IN 8-bit	IN 07H	Accept the data from input devices and store in Accumulator

Figure: Data transfer Instruction

2. Arithmetic Instructions

The arithmetic instructions perform addition, subtraction, increment and decrement operations.

- **Addition:**
  Any 8-bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and the resulted sum is stored in the accumulator.
  The resulted carry bit is stored in the carry flag.
- **Subtraction:**
  Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the result is stored in the accumulator.
  The resulted borrow bit is stored in the carry flag.
- **Increment/Decrement:**
  The 8085 has the increment and decrement instructions to increment and decrement the contents of any register, memory location or register pair by 1.

Mnemonics	Example	Description
ADD R	ADD B	Add content of register R to the content of Accumulator
ADI 8-bit data	ADI 37H	Add 8-bit data to the content of Accumulator
ADD M	ADD M	Add content of memory to the content of Accumulator
SUB R	SUB C	Subtract the content of register R from the content of Accumulator
SUI 8-bit data	SUI 7FH	Subtract the 8-bit data from the content of the Accumulator
SUB M	SUB M	Subtract the content of memory from the content of Accumulator
INR R	INR D	Increment the content of register R by 1
INR M	INR M	Increment the content of memory by 1
DCR R	DCR E	Decrement the content of register R by 1
DCR M	DCR M	Decrement the content of memory by 1
INX Rp	INX H	Increment the content of register pair by 1
DCX Rp	DCX B	Decrement the content of register pair by 1

Figure: Arithmetic Instruction

3. Logical Instructions

The instruction under this group performs logical operations such as AND, OR, COMPARE, ROTATE, etc. on the data stored in registers, memory and status flags.

- **Logical:**
  Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, ORed, or Exclusive-ORed with the contents of the accumulator
  The result is stored in the accumulator. The result also affects the flags.
- **Rotate:**
  These instructions allow shifting of each bit in the accumulator either left or right by 1 bit position.
- **Compare:**
  Any 8-bit number, or the contents of a register, or the contents of a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
- **Complement:**
  The result of accumulator can be complemented with this It replaces all 0s by 1s and all 1s by 0s.

Mnemonics	Example	Description
ANA R	ANA B	Logically ‘AND’ the content of Register R with the content of Accumulator
ANI 8-bit	ANI 2FH	Logically ‘AND’ the 8-bit data with the content of Accumulator
ANA M	ANA M	Logically ‘AND’ the content of Memory with the content of Accumulator
ORA R	ORA C	Logically ‘OR’ the content of Register R with the content of Accumulator
ORI 8-bit	ORA 3FH	Logically ‘OR’ the 8-bit data with the content of Accumulator
ORA M	ORA M	Logically ‘OR’ the content of memory with the content of Accumulator
XRA R	XRA B	Logically ‘Ex-OR’ the content of Register R with the content of Accumulator
XRI 8-bit	XRI 3EH	Logically ‘Ex-OR’ the 8-bit data with the content of Accumulator
XRA M	XRA M	Logically ‘Ex-OR’ the content of memory with the content of Accumulator
CMP R	CMP B	Compares the content of Register R with the content of Accumulator
CPI 8-bit	CPI 4FH	Compares the 8-bit data with the content of Accumulator

Figure: Logical Instruction

4. Branching Instructions

- These instructions allow the 8085 to change the sequence of the program, either unconditionally or under certain test conditions.
- These instructions include branch instructions, subroutine call and return instructions and restart instructions.

Mnemonics	Example	Description
JMP 16-bit address	JMP 2050H	Change the program sequence to 16- bit address.
JZ 16-bit address	JZ 2070H	If the zero flag is set, change the program sequence to 16-bit address.
JNZ 16-bit address	JNZ 2080H	If the zero flag is reset, change the program sequence to 16-bit address.
JC 16-bit address	JC 2060H	If the carry flag is set, change the program sequence to the 16-bit address.
JNC 16-bit address	JNC 2090H	If the carry flag is reset, change the program sequence to 16-bit address.
CALL 16-bit address	CALL 2070H	Change the program sequence to the location of subroutine.
RET	RET	Return to the calling program.

Figure: Branching Instruction

5. Program Control Instructions

- These instructions control the operation of microprocessor. This group includes the instruction for input/output ports, stack and machine control.

Mnemonics	Example	Description
HLT	HLT	Stop processing and wait
NOP	NOP	Do not perform any operation

Figure: Program Control Instructions

Stack

- The stack in an 8085 micro computer system can be described as a set of memory location in the R/W memory specified by the programmer in the main program. These memory locations are used to store binary information temporarily during the execution of a program.
- Data bytes in the register pairs of the microprocessor can be stored on the stack by using the instruction ‘PUSH’. Data byte can be transferred from the stack to the register pairs by using the instruction ‘POP’.

Assembly language Programming

- Developed to overcome the difficulties of programming in machine language.
- Contains the same instructions as a machine language but each instructions and variable has a symbol (called mnemonics) instead of being just number.
- E.g. : ADD,SUB,DIV,JMP,MOV,INR,DCR etc.
- After developing assembly language it was easier program using symbols
- the program written in assembly language need to be converted to machine language

Assembler

- Program that translates the mnemonics into their machine code.
- A program can be entered in mnemonics in a microcomputer equipped with an ASCII keyboard.
- The assembler will translate mnemonics into the 8085-machine code and assign memory locations to each machine code
- An **assembler** is a program or a software which translate the programs written in assembly language to machine language.

Basic ALP using 8085 Instruction Sets:

Write a program to Perform the following functions and verify the output.

- Load the number 8BH in register D.
- Load the number 6FH in register C.
- Increment the contents of register C by one.
- Add the contents of register D and C and display the result in PORT 1.

Load the accumulator A with the data byte 82H and save the data in register B.

MVI A, 82H
MOV B, A

Write a program to find out 1’s complement of a number stored the memory location 2000H and place the result in the memory location 2001H.

```
LDA 2000H
CMA
STA 2001H
HLT
```

Write a program to find out 2’s complement of a number stored the memory location 5000H and place the result in the memory location 5004H

```
LDA 5000
CMA
INR  A
STA 5004
HLT
```

Write a program to copy the content stored the memory location 4000H to Register B

```
LXI H, 4000H
MOV B, M
```

Or

```
LDA 4000H
MOV B, A
```

Store the data byte 32H into memory location 4000H.

```
MVI A, 32H
STA 4000H
HLT
```

Exchange the contents of memory location 2000H and 4000H

```
LDA 2000H
MOV B, A
LDA 4000H
STA 2000
MOV A, B
STA 4000H
HLT
```

8085 program to add 02 and 04

```
MVI A, 02H
ADI 04H
HLT
```

Or

```
MVI A, O2H
MVI B,04H
ADD B
HLT
```

8085 Assembly language program for adding two 8 bit numbers.

Solution: Let us suppose the 8 –bit numbers are stored in 4300H and 4301H , and result in 4302H and 4303H.

```
MVI C, 00H
LDA 4300H
MOV B, A
LDA 4301H
ADD B
JNC loop
INR C
loop: STA 4302H
MOV A, C
STA 4303H
HLT
```

8085 program to multiply 02 and 04

```
MVI A, 00H ; Initialize accumulator A to 0
MVI B, 02H ; Load the first number (02) into register B
MVI C, 04H ; Load the second number (04) into register C

LOOP: ADD B   ; Add the first number to accumulator A
      DCR C   ; Decrement the second number in register C
      JNZ LOOP ; If the second number in register C is not zero, jump to the LOOP label

STA 4300H ; Store the result (08) in memory location 4300H
HLT       ; Halt the program
```

8085 Assembly language program to multiply two 8 bit numbers.

Solution: Let us suppose the 8 –bit numbers are stored in 4150H and 4151H , and result in 4152H and 4153H.

```
MVI D, 00H
MVI A, 00H
LXI H, 4150H
MOV B, M
LXI H, 4151H
MOV C, M
loop: ADD B
JNC next
INR D
next: DCR C
JNZ loop
STA 4152H
MOV A, D
STA 4153H
HLT
```

8085 Assembly language program to add two 16 bit numbers.

```
MVI C, 00H
LHLD  C050H
XCHG
LHLD C052H
DAD D
JNC loop
INR C
Loop: SHLD C054H
MOV A, C
STA C056H
HLT
```

Write an ASP to generate first 10 even numbers.

```
MVI A, 00H
MVI B, 02H
MVI C, OAH
LXI H, 4200H
LOOP: ADD B
MOV M, A
INX H
DCR C
JNZ LOOP
```

Write an ASP to generate find largest number in array.

Let suppose:

memory	content
4200	05(size of array)
4201	0A
4202	11
4203	03
4204	04
4205	C2

Solution:

```
LXI H, 4200H
MOV B, M
INX H
MOV A, M
DCR B
Loop:INX H
CMP M
JNC skip
MOV A, M
Skip: DCR B
JNC loop
STA 4300H
HLT
```

Practise Questions:

- Write an ASP to generate first 20 odd numbers.
- Write an ASP to divide two 8 bit numbers.
- Write an ASP to generate find smallest number in array.
- 8085 Assembly language program to add two 16 bit numbers.(without DAD instructions)

And more…….

Thank You ☺