**UNIT II**

**PROGRAMMING OF 8085 PROCESSOR**

Instruction -format and addressing modes – Assembly language format – Data transfer, data manipulation& control instructions – Programming: Loop structure with counting & Indexing – Look up table - Subroutine instructions – stack

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

### Instruction Format

* 1. **Instruction**:
* It is a command given to the microprocessor to perform given task on specified data.Each instruction has two parts viz. task to be performed known as operation code or **opcode** and second is the data to be operated upon known as **operand**.
* The Operand can be used in many different ways e.g. 8 bit data or 16 bit data or internal register or memory location or 8 bit or 16 bit address.
* 8085 Instructions can be classified based on the size they occupy in memory or by the functions they perform. Figure shows the classification of the instructions.



**Fig: Classification of Instruction Set of 8085**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Discuss in detail about the 8085 instruction set and explaining about the various types of operation.[December 2013,April 2011,June 2016][April 2018]***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1.1.Instruction Format**

Based on the size, the instructions can be classified are as follows

**One byte Instructions:**

These instructions are of one byte in size and hence occupy one memory location in RAM. Examples are CMA, RLC, RRC, RAL, RAR, STC, CMC etc. These instructions do not require any operand to be specified with the instructions; instead the operand is implied in the instructions.



**Two Byte Instruction:**

These instructions of two byte (16-bits) in size and hence will occupy two memory locations in RAM. Examples of such instructions are MVI C, 0A;



**Three Byte Instructions:**

These are of three byte in size and hence occupy three locations in memory (RAM). Examples of such instructions are CALL, JMP etc.



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Explain the classification of Instruction set with example.**

**Explain logical instruction with example. (December 2015)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* 1. **Instruction Set Classification**

 An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform.

 These instructions can be classified into the following five functional categories:

* Data transfer (copy) operations,
* Arithmetic operations,
* Logical operations,
* Branching operations, and
* Machine-control operations.

**1 Data Transfer Croup**

 The data transfer instructions move data between registers or between memory and registers.

* MOV                              Move
* MVI                                Move Immediate
* LDA                                Load Accumulator Directly from Memory
* STA                               Store Accumulator Directly in Memory
* LHLD                             Load H & L Registers Directly from Memory
* SHLD                             Store H & L Registers Directly in Memory

 An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

* LXI                                 Load Register Pair with Immediate data
* LDAX                             Load Accumulator from Address in Register Pair
* STAX                             Store Accumulator in Address in Register Pair
* XCHG                  Exchange H & L with D & E
* XTHL                             Exchange Top of Stack with H & L

 **2 Arithmetic Group**

  The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

* ADD     Add to Accumulator
* ADI      Add Immediate Data to Accumulator
* ADC     Add to Accumulator Using Carry Flag
* ACI      Add immediate data to Accumulator Using Carry
* SUB     Subtract from Accumulator
* SUI      Subtract Immediate Data from Accumulator
* SBB   Subtract from Accumulator Using Borrow (Carry) Flag
* SBI     Subtract Immediate from Accumulator Using Borrow (Carry) Flag
* INR    Increment Specified Byte by One
* DCR   Decrement Specified Byte by One
* INX    Increment Register Pair by One
* DCX   Decrement Register Pair by One
* DAD   Double Register Add; Add Content of Register Pair to H & L Register Pair

**3 Logical Group**

 This group performs logical (Boolean) operations on data in registers and memory and on condition flags. The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

* ANA      Logical AND with Accumulator
* ANI       Logical AND with Accumulator Using Immediate Data
* ORA      Logical OR with Accumulator
* OR        Logical OR with Accumulator Using Immediate Data
* XRA      Exclusive Logical OR with Accumulator
* XRI       Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

* CMP      Compare
* CPI    Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

* RLC                                Rotate Accumulator Left



* RRC                                Rotate Accumulator Right



* RAL                                Rotate Left Through Carry
* RAR                               Rotate Right Through Carry

**Complement and carry flag instructions**:

* CMA            Complement Accumulator
* CMC            Complement Carry Flag
* STC              Set Carry Flag

**4 Branch Group**

 The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The **unconditional branching instructions** are as follows:

* JMP            Jump
* CALL          Call
* RET            Return

**Conditional branching instructions** examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

* NZ          Not Zero (Z = 0)
* Z             Zero (Z = 1)
* NC          No Carry (C = 0)
* C             Carry (C = 1)
* PO          Parity Odd (P = 0)
* PE          Parity Even (P = 1)
* P             Plus (S = 0)
* M            Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

**Jumps         Calls  Returns**

INC   CNC RNC (No Carry)

JNZ  CNZ  RNZ (Not Zero)

JM     CM     RM   (Minus)

JP0    CPO  RPO (Parity Odd)

JM     CM    RM  (Minus)

JPE  CPE  RPE (Parity Even)

JP0   CPO  RPO  (Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

* PCHL       Move H & L to Program Counter
* RST          Special Restart Instruction Used with Interrupts

**5 .Stack Instructions**

 The following instructions affect the Stack and/or Stack Pointer

* PUSH       Push Two bytes of Data onto the Stack
* POP          Pop Two Bytes of Data off the Stack
* XTHL       Exchange Top of Stack with H & L
* SPHL        Move content of H & L to Stack Pointer

**6 .I/0 instructions**

* IN             Initiate Input Operation
* OUT          Initiate Output Operation

 **7 Machine Control instructions**

* EI             Enable Interrupt System
* DI             Disable Interrupt System
* HLT          Halt
* NOP          No Operation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Define addressing mode.Write the types of addressing modes with example.(December 2014)(December 2015) (April 2015)(April 2018)(Dec 2018)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**3. Addressing Modes**

Various ways of specifying the operands or various formats for specifying the operands is called addressing mode

* **Implicit addressing**

This mode doesn’t require any operand; the data is specified by the opcode itself.

* CMA – Complement the contents of accumulator
* **Immediate addressing**

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand.

* MVI B, 05H means 05 is copied into register B.
* ADI 06H
* **Direct addressing** –

 In this mode, the data is directly copied from the given address to the register.

* STA 2400H, IN 02H
* STA 2400H means the data at address 2400 is copied to register A.
* **Register addressing**

In this mode, the data is copied from one register to another.

* MOV A, B
* ADD B
* **Register indirect addressing**

In this mode, the data is transferred from one register to another by using the address pointed by the register.

* LDAX B,
* STAX D

 LDAX B means data is transferred from the memory address pointed by the register pair BC to the register A.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Write short notes on STACK and Subroutine**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **STACK:**
 Stack is an area of memory identified by the programmer for temporary storage of information.
* Stack is a LIFO structure.
* Stack normally grows backwards into memory-the programmer defines the bottom of the stack and the stack grows up into reducing address range.
* Stack is defined by setting the SP(stack pointer) register.LXI SP,FFFFH
* The size of the stack is limited only by the available memory.
* Information is saved on the stack by PUSHing it on.
* Information is retrieved from the stack by POPing it off.
* The 8085 provides two instructions:PUSH and POP for storing information on the stack and retrieving it back.
* Both PUSH and POP work with register pairs only.

**PUSH instruction**
 **EX: PUSH B**

**Steps to be followed for PUSH  B one byte instruction**

* Decrement SP
* Copy the contents of register B to the memory location pointed by SP.
* Decrement SP
* Copy the contents of register C to the memory location pointed to by SP.

**POP instruction:**
 **EX: POP D**

**Steps to be followed for POP D one byte instruction**

* Copy the contents of the memory location pointed to by the SP to register E.
* Increment SP
* Copy the contents of the memory location pointed to by the SP to register D.
* Increment SP

**Operation of the stack:**

* During pushing, the stack operates in a ' decrement then store' style-The stack pointer is decremented first, then the information is placed on the stack
* During popping, the stack operates in a "use then increment' style.-The information is retrieved from the top of the stack and then the pointer is incremented.
* The SP pointer always points to the "top of the stack"

**LIFO:**

The order of PUSHs and POPs  must be opposite of each other in order to retrieve information back into its original location.

PUSH B
PUSH D
.....
POP D
POP B

Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE

**PSW register pair:**
 The 8085 recognizes one additional register pair called the PSW(PROGRAM STATUS WORD). This register pair is made up of the accumulator and the flag registers.

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explain the use of lookup table in 8085.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Lookup table**
* A lookup table is an array that replaces runtime computation with a simpler array indexing operation.
* The savings in terms of processing time can be significant, since retrieving a value from memory is often faster than undergoing an 'expensive' computation or input/output operation.
* The tables may be pre-calculated and stored in static program storage, calculated (or "pre-fetched") as part of a program's initialization phase (memorization), or even stored in hardware in application-specific platforms.
* Lookup tables are also used extensively to validate input values by matching against a list of valid (or invalid) items in an array and, in some programming languages, may include pointer functions (or offsets to labels) to process the matching input.

**Example of look up table Algorithm**

1. Initialize HL pair to point Look up table

2. Get the data

 3. Check whether the given input is less than 9

 4. If yes go to next step else halt the program

 5. Add the desired address with the accumulator content

6. Store the result

**Program:**

LXI H,5000 ;Initialsie Look up table address

LDA 5050 ;Get the data

CPI 0A ;Check input > 9

 JC AFTER ;if yes error

MVI A,FF ;Error Indication

 STA 5051

HLT

 AFTER: MOV C,A ;Add the desired Address

 MVI B,00

 DAD B

 MOV A,M

STA 5051 ;Store the result

 HLT ;Terminate the program

**LOOK UP TABLE:**

5000 01

5001 04

5002 09

5003 16

5004 25

5005 36

5006 49

5007 64

5008 81

**RESULT:**

 Input: Data : 05H in memory location 5050

Output: Data : 25H (Square of 5) in memory location 5051

 Input: Data : 11H in memory location 5050

Output: Data : FFH (Error Indication) in memory location 5051

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Sample 8085 Assembly Programs

**Example-1: Write assembly program to add two numbers. (December 2014)**

### MVI D, 8CH MVI C, 6EH MOV A, C ADD D OUT PORT1 HLT

**Example-2: Write assembly program to multiply a number by 8
Multiply by 2 is equivalent to shifting.**

### MVI A, 40HRLCRLCRLCOUT PORT1HLT

**Example-2: Write assembly program to multiply two 8 bit number**

|  |  |
| --- | --- |
|  | MVI B, 07H |
|  | MVI C, 06H |
|  | XRA A |
| GO | ADD B |
|  | DCR C |
|  | JNZ GO |
|  | STA 4A00 |
|  | HLT |

### Example-3: Write assembly program to find greatest between the two numbers.

### MVI B, 30HMVI C, 40HMOV A, BCMP CJZ EQUJC GRTOUT PORT1HLTEQU: MVI A, 01HOUT PORT1HLTGRT: MOV A, COUT PORT1HLT

**Write an assembly language program for to generate Fibannoci series using subroutines(Dec 2014)**

MVI A,00

STA 8000

MVI A,01

STA 8001

MVI B,08

LXI H,8000

BACK: MOV A,M

INX H

ADD M

INX H

MOV M,A

DCR B

DCX H

JNZ BACK

HLT

**Write an assembly language program for to multiply two 16 bit numbers. (June 2016)**

|  |  |
| --- | --- |
|  | LXIH, 0000 |
|  | LXIB, 1CD2 |
|  | LXI SP,01AD |
|  | LXI D, 0000 |
| MUL | DAD SP |
|  | JNC DOWN |
|  | INX D |
| DOWN | DCX B |
|  | MOV A,B |
|  | ORA C |
|  | JNZ MUL |
|  | SHLD 4A00 |
|  | XCH G |
|  | SHLD 4A02 |
|  | HLT |

**Programming using subroutine Instructions**

**Generation of Square waveform using DAC**

 **Programming using Loop structure with Counting and Indexing**

**(i) 16 bit Multiplication**



**(ii)Finding the maximum number in the given array**



**Develop an algorithm and 8085 assembly language program to sort 100 byte type data. Explain the instruction used in the program.(Dec 2018)**

**L3 MVI B,00**

 LXI H,4200

 MOV C,M

 INX H

 DCR C

L2 MOV A,M

 INX H

 CMP M

 JC L1

 MOV D,M

 MOV M,A

 DCX H

 MOV M,D

 MVI B,01

L1 DCR C

 JNZ L2

 DCR B

 JZ L3

 HLT

 INPUT

4200 99

4201 data1

…………….

4300 data 100

 &&&&&&&&&&&&&&&&&&&&&&&&&&