Sub Code: 18BIT46S

Skill Based Subject – II: MICRO PROCESSOR & ASSEMBLY LANGUAGE PROGRAMMING

UNIT II: Instruction set of INTEL 8085: Introduction – Instruction and Data formats - Addressing Modes - Status Flags - Intel 8085 Instructions: Data transfer group – Arithmetic group- Logical group - Branch group - Stack, I/O machine control group - Assembly language: stacks – subroutines – MACRO.

Prepared by Dr.P.SUMATHI

INSTRUCTION AND DATA FORMATS

The total memory location required to feed the instruction in memory is called as **instruction word size**. The memory location of 8085 microprocessor can accommodate 8-bits of data. To store 16-bits data, they are stored in two consecutive memory locations (i.e. 2 Bytes).

According to the instruction word size in 8085 microprocessor, there are three types of instructions:

a. 1-Byte instruction
b. 2-Byte instruction
c. 3-Byte instruction

1 – Byte Instruction

- They include opcode and operands in the same byte.
- Operands are internal registers and coded into the instruction.
- Instructions require one memory location to store the single byte in the memory.

Examples: MOV B,C ; LDAX B ; NOP ; HLT

2-Byte instruction

- 1st byte specifies opcode and 2nd byte specifies operand.
- Instructions require two memory locations to store in the memory.
- Examples: MVI B, 26 H ; IN 56 H

3-Byte instruction

- In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address.
- The 2nd byte holds the low order address.
- The 3rd-byte holds the high order address.
- Instructions require three memory locations to store the single byte in the memory.
- Examples: LDA 2050 H ; JMP 2085 H

ADDRESSING MODES OF 8085

Every instruction of a program has to operate on a data.

The method of specifying the data to be operated by the instruction is called Addressing.

1. Direct Addressing:

- In direct addressing mode, the address of the operand (data) is given in the instruction itself. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory.
- Ex: LDA 1050H - Load the data available in memory location 1050H in to accumulator. SHLD 3000H

2. Register Addressing:

- In register addressing mode, the instruction specifies the name of the register in which the data is available.
- Ex: MOV A, B - Move the content of B register to A register.
  SPHL, ADD C.

3. Register Indirect Addressing:

- In register indirect addressing mode, the instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair.
- Ex. MOV A, M - The memory data addressed by H-L pair is moved to A register.
  LDAX B.

4. Immediate Addressing:

- In immediate addressing mode, the data is specified in the instruction itself. The data will be a part of the program instruction.
- Ex. MVI B, 3EH - Move the data 3EH given in the instruction to B register;
  LXI SP, 2700H.

5. Implicit Addressing:

- There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand.
- Ex. CMA, RAL, RAR etc.,

Status Flags

There is a set of five flip-flops which indicate status (condition) arising after the execution of arithmetic and logic instructions. These are:

- **Carry Flag (CS)**
- **Parity Flag (P)**
- **Auxiliary Carry Flags (AC)**
- **Zero Flags (Z)**
- **Sign Flags (S)**

INSTRUCTION SET OF INTEL 8085

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.

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.

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.

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.

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.

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.

Intel 8085 Instructions

Data Transfer Group

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 to 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.

2. 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.

3. Logical Group

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]

4. Branch Group

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)

1. CC addr(label) - Call subroutine if carry status CS=1
2. CNC addr (label) -Call subroutine if carry status CS=0
3. CZ addr (label) -Call Subroutine if the result is zero
4. CNZ addr (label) -Call Subroutine if the result is not zero
5. CP addr (label) -Call Subroutine if the result is plus
6. CM addr (label) -Call Subroutine if the result is minus
7. CPE addr(label) -Call subroutine if even parity
8. CPO addr(label) -Call subroutine if odd parity

5. RET - Unconditional Return, Return from subroutine.
[PCL] ← [[SP]], [PCH] ← [[SP] + 1], [SP] ← [SP] + 2

5. 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)

Assembly language:

What is an Assembly language?

Assembly language is a low level computer programming language that does not have system independency. It uses some special symbols for the operations; these symbols are known as **Mnemonics.**

What is an Assembler?

he Assembler is used to translate the program written in Assembly language into machine code. The source program is an input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

Stack

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element.

During the execution of a program sometimes it becomes necessary to save the contents of certain registers because the registers are required for some other operation in subsequent steps. These contents are moved to certain memory locations by PUSH operation. Then the registers are used for other operations. After completing these operations those contents which were saved in the memory are transferred back to the registers by POP operation. Memory locations for this purpose is set aside by the programmer in the beginning. The set of memory locations kept for this purpose is called stack. The last memory location of the occupied portion of the stack is called **stacktop.**

The two operations of a stack are:

1. Push: Inserts an item on top of stack.
2. Pop: Deletes an item from top of stack.

A special 16-bit register known as **stack pointer** holds the address of stacktop. The stack pointer is initialized in the beginning of the program by LXI or SPHL instruction. Any area of the RAM can be used as a stack. There is no restriction on the location of the stack in the memory. Data are stored in the stack on last-in-first-out (LIFO) principle.

Similarly, POP operation is used to transfer the contents from the stack to the register. The stack position before and after POP operation has been shown in figure 5.3 (a) and (b) respectively.

Subroutines:

Subroutine is a block of code that is called from different places from within a main program or other subroutines.

- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist anywhere in the code.
- Saves code space in that the subroutine code does not have to be repeated in the program areas that need it
- Only the code for the subroutine call is repeated
- Subroutine can have parameters that control its operation. A subroutine may pass a return value back to the caller
- Space in data memory must be reserved for parameters, local variables, and the return value.

The 8085 has two instructions for dealing with subroutines.
– The CALL instruction is used to redirect program execution to the subroutine.
– The RET instruction is used to return the execution to the calling routine.

The CALL Instruction
– When CALL instruction is fetched, the microprocessor knows that the next two memory location contains subroutine address in the memory.

The RET Instruction
– Retrieve the return address from the top of the stack – Load the program counter with the return address.

The CALL instruction places the return address at the two memory locations immediately before where the Stack Pointer (SP) is pointing.
– We must set the SP correctly BEFORE using the CALL instruction.

The RET instruction takes the contents of the two memory locations at the top of the stack and uses these as the return address.
– Do not modify the stack pointer in a subroutine. We will lose the return address.

Number of PUSH and POP instruction used in the subroutine must be same, otherwise, RET instruction will pick wrong value of the return address from the stack and program will fail.

3. What is Macro?
A sequence of instructions to which a name is assigned is called a MACRO.
The macro is invoked by using the macro name along with the necessary parameters. When we need to use some sequence of instructions many times in a program, we can put those instructions in a macro and use it instead of writing the instructions all the time.

Example1:

```
MACRO   ADDRESS
LXI       H, ADDRESS
MOV      A,M
CMA
ENDM
```
