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. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions. 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. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source. For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

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

1. 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 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.
6. LXI rp, data 16. (Load register pair immediate). [rp] <-- data 16 bits, [rh] <-- 8 LSBs of data.
7. LDA addr. (Load Accumulator direct). [A] <-- [addr].
8. STA addr. (Store accumulator direct). [addr] <-- [A].
9. LHLD addr. (Load H-L pair direct). [L] <-- [addr], [H] <-- [addr+1].
10. SHLD addr. (Store H-L pair direct) [addr] <-- [L], [addr+1] <-- [H].
11. LDAX rp. (LOAD accumulator indirect) [A] <-- [[rp]]

12. STAX rp. (Store accumulator indirect) [[rp]] <-- [A].
13. XCHG. (Exchange the contents of H-L with D-E pair) [H-L] <--> [D-E].

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.
6. ACI data (Add with carry immediate data to accumulator). [A] <-- [A] + data + [CS].
7. DAD rp. (Add register paid to H-L pair). [H-L] <-- [H-L] + [rp].
8. SUB r. (Subtract register from accumulator). [A] <-- [A] – [r].
9. SUB M. (Subtract memory from accumulator). [A] <-- [A] – [[H-L]].
10. SBB r. (Subtract register from accumulator with borrow). [A] <-- [A] – [r] – [CS].
11. SBB M. (Subtract memory from accumulator with borrow). [A] <-- [A] – [[H-L]] – [CS].
12. SUI data. (Subtract immediate data from accumulator) [A] <-- [A] – data.
13. SBI data. (Subtract immediate data from accumulator with borrow). [A] <-- [A] – data – [CS].
14. INR r (Increment register content) [r] <-- [r] +1.
15. INR M. (Increment memory content) [[H-L]] <-- [[H-L]] + 1.
16. DCR r. (Decrement register content). [r] <-- [r] – 1.
17. DCR M. (Decrement memory content) [[H-L]] <-- [[H-L]] – 1.
18. INX rp. (Increment register pair) [rp] <-- [rp] – 1.
19. DCX rp (Decrement register pair) [rp] <-- [rp] -1.
20. DAA (Decimal adjust accumulator) .

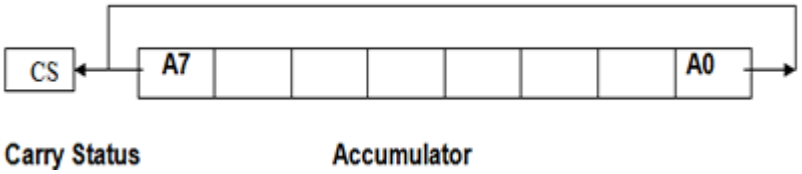
The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in the decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

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]
14. CMP M. (Compare memory with accumulator) [A] – [[H-L]]
15. CPI data. (Compare immediate data with accumulator) [A] – data.

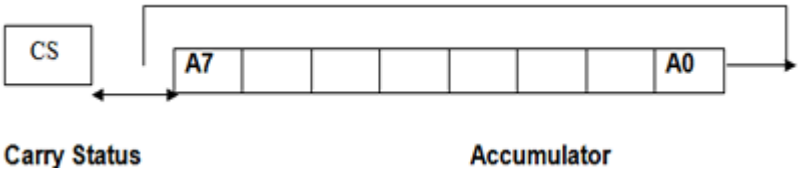
The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

16. RLC (Rotate accumulator left) [An+1] <-- [An], [A0] <-- [A7],[CS] <-- [A7]**.**



The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected.

17. RRC. (Rotate accumulator right) [A7] <-- [A0], [CS] <-- [A0], [An] <-- [An+1].



The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected.

18. RAL. (Rotate accumulator left through carry) [An+1] <-- [An], [CS] <-- [A7], [A0] <-- [CS].
19. RAR. (Rotate accumulator right through carry) [An] <-- [An+1], [CS] <-- [A0], [A7] <-- [CS]

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)
3. CALL addr (label) (Unconditional CALL: call the subroutine identified by the operand)

    CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stack top. Then the program jumps to subroutine starting at address specified by the label.

4. RET (Return from subroutine)

5. RST n (Restart) Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location.

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)
6. POP PSW (Pop Processor Status Word)
7. HLT (Halt)
8. XTHL (Exchange stack-top with H-L)
9. SPHL (Move the contents of H-L pair to stack pointer)
10. EI (Enable Interrupts)
11. DI (Disable Interrupts)
12. SIM (Set Interrupt Masks)
13. RIM (Read Interrupt Masks)
14. NOP (No Operation)