MODULE-5

Part 2

8051 Addressing Modes

- An **addressing mode** refers to how you are addressing a given memory location. The five different ways of addressing are as follows −
- Immediate addressing mode
- Direct addressing mode
- Register direct addressing mode
- Register indirect addressing mode
- Indexed addressing mode

Immediate Addressing Mode

- Data operand is a constant and it is a part of the instruction itself
- The data must be preceded by a # sign.
- Eg1:-MOV A, #63H

The data (constant) 63 is moved to the accumulator register

- Eg2: MOV DPTR, # 2000H

Direct addressing mode

- Only for Internal RAM and the SFRs

Eg:-**MOV R1, 42H** : Move the contents of RAM location 42 into R1 register

MOV 49H,A : Move the contents of the accumulator into the RAM location 49

- MOV 30H,#30H
- MOV 01H,00H --------allowed in MC (not in MP)
- MOV A,80H ----------A P0 {SFR}
- MOV 30H,A

Register addressing mode

- Data operand to be manipulated lies in one of the registers
- Eg:-
  - **MOV A,R0** : Move the contents of the register R0 to the accumulator
  - **ADD A,R6** :Add the contents of R6 register to the accumulator
  - **MOV P1, R2 :** Move the contents of the R2 register into port 1

Note: MOV R0,R1 (not allowed)

So use MOV A, R1

MOV R0,A

Register Indirect addressing mode

- This is for internal RAM & External RAM
- To access Int RAM ,8-bit addresses required(R0/R1 only)
- To access ext RAM , 16-bit addresses required(DPTR)
- Eg:- Internal RAM
  - **MOV A,@ R0 :**Move the contents of RAM location whose address is in R0 into **A (accumulator)**
  - **MOV @ R1 , B** : Move the contents of B into RAM location whose address is held by R1

Eg:- External RAM

```
MOVX  A,@DPTR
MOVX  DPTR, @A
```

Indexed addressing mode

- Only for ROM
- Address obtained from two registers
- Eg:-
  - MOVC A, @ A+DPTR
  - MOVC A, @A+PC

8051 INSTRUCTION SET

- **1. Instruction Timings**
- **T-state, Machine cycle and Instruction cycle** are terms used in instruction timings.
- **T-state** is defined as one subdivision of the operation performed in one clock period.
- **Machine cycle** is defined as 12 oscillator periods. A machine cycle consists of six states and each state lasts for two oscillator periods. An instruction takes one to four machine cycles to execute an instruction.
- **Instruction cycle** is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

The instructions of 8051

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Boolean/Bit manipulation instructions

DATA TRANSFER INSTRUCTIONS

a. Move the contents of a register Rn to A

i. MOV A,R2

b. Move the contents of a register A to Rn

i. MOV R4,A

c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)

- i. MOV A, #45H
- ii. MOV R6, #51H
- iii. MOV 30H, #44H
- iv. MOV @R0, #0E8H
- v. MOV DPTR, #0F5A2H

- **d. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing**
  - i. MOV A, 65H
  - ii. MOV A, @R0
  - iii. MOV 45H, A
  - iv. MOV @R1, A
- **e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing**
  - i. MOV R3, 65H
  - ii. MOV 45H, R2

- **f. Move the contents of memory location to another memory location using direct and indirect addressing**
  - i. MOV 20H, 30H
  - ii. MOV 45H, @R0
- **g. Move the contents of an external memory to A or A to an external memory**
  - i. MOVX A,@R1
  - ii. MOVX @R0,A
  - iii. MOVX A,@DPTR
  - iv. MOVX@DPTR,A

- **h. Move the contents of program memory to A**
  - i. MOVC A, @A+PC
  - ii. MOVC A, @A+DPTR

- **i. Exchange instructions :-The content of source ie., register, direct memory or indirect memory will be exchanged with the contents of destination ie., accumulator**.
  - i. XCH A,R3
  - ii. XCH A,@R1
  - iii. XCH A,54h
- **j. Exchange digit:- Exchange the lower order nibble of Accumulator (A0-A3) with lower order nibble of the internal RAM location which is indirectly addressed by the register.**
  - i. XCHD A,@R1
  - ii. XCHD A,@R0

- k.PUSH 25H

  SP□ SP+1 ,

Then [SP]□ [25H]

- l.POP 25H

First [25H] □[SP] ,Then SP □ SP-1

ARITHMETIC INSTRUCTIONS

- **1.Addition**

i. Add the contents of A with immediate data with or without carry.

i. ADD A, #45H

ii. ADDC A, #OB4H

ii. Add the contents of A with register Rn with or without carry.

i. ADD A, R5

ii. ADDC A, R2

iii. Add the contents of A with contents of memory with or without carry using direct and indirect addressing

i. ADD A, 51H

ii. ADDC A, 75H

iii ADD A, @R0

iv ADDC A, @R1

2.Subtraction

- **i. Subtract the contents of A with immediate data with or without carry.**
  - i. SUBB A, #45H
  - ii. SUBB A, #OB4H
- **ii. Subtract the contents of A with register Rn with or without carry.**
  - i. SUBB A, R5
  - ii. SUBB A, R2

- **iii. Subtract the contents of A with contents of memory with or without carry using direct and indirect addressing**
  - i. SUBB A, 51H
  - ii. SUBB A, 75H
  - iii. SUBB A, @R1
  - Simple Subtraction also use SBB, so use CLR C instn before subtraction

3.Multiplication

- **MUL AB.**

- MOV A,#45H ; [A]=45H
- MOV B,#0F5H ; [B]=F5H
- MUL AB ; [A] x [B] = 45 x F5 = 4209 ;
  - [A]=09H, [B]=42H

4.Division

- **DIV AB.**

Eg:

- MOV A,#0F5H ;
- MOV B,#45H ;
- DIV AB ;

A □----Quotient, B □----Remainder

5. **DA A (Decimal Adjust After Addition).**

- When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, we use DA A instruction after the addition. DA A works as follows. If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble. If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

6. Increment:

- INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0. In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.
- INC A
- INC R1
- INC 25H
- INC @R0
- INC DPTR

7.Decrement:

- DEC decrements the value of source by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from FFh to 0
- DEC A
- DEC R0
- DEC 25H
- DEC @R0
- DEC DPTR (Invalid)

LOGICAL INSTRUCTIONS

1. **Logical AND**

- **ANL** destination, source
- Eg:-
- ANL A,#25 ANL A,@R0 ANL 25H,A
- ANL A, R0 ANL A,20H ANL 25H, #25
- Note:

Clear LN, Suppose A= 0011 0101 (AND LN with 0's & HN with 1's)

ANL A, #0F0H

Logical OR

2.ORL destination, source

- Eg:-
- ORL A,#25 ORL A,@R0 ORL 25H,A
- ORL A, R0 ORL A,20H ORL 25H, #25
- Note:

SET LN , Suppose A= 0011 0101 (OR LN with 1's & HN with 0's)

ORL A, #0FH

Logical Ex-OR

- **3.XRL destination, source**
- XRL A,#25 XRL A,@R0 XRL 25H,A
- XRL A, R0 XRL A,20H XRL 25H, #25
- Note:

Complement LN, Suppose A= 0011 0101 (XRL LN with 1's & HN with 0's)

XRL A, 0FH

4.Logical NOT

CPL A, CPL C, CPL bit address

5.SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

- **RR A**
- This instruction is rotate right the accumulator. Each bit is shifted one location to the right, with bit 0 going to bit 7.
- **RL A**
- Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0
- **RRC A**
- Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7
- **RLC A**
- Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0

BRANCH INSTRUCTIONS- Unconditional

- Branch instruction -2 types JUMP & CALL

There are 3 types of jump instructions. They are:-

- 1. Short Jump (SJMP radd)
- 2. Absolute Jump (AJMP sadd)
- 3. Long Jump(LJMP ladd)

CALL-

- LCALL sub
- SCALL sub

RETURN instn – 2 types

- RET
- RETI

- .1.Short Jump (SJMP radd)
    - Radd- 8-bit signed no
    - Ranges (-128 to +127)
    - Radd is calculated as the relative distance from the next instn to the branch locn
    - Telling how far we jump not where we want to jmp
    - Eg: SJMP 08H
- 2. Absolute Jump (AJMP sadd)
- Here entire 64 KB divided into 32 pages, each page is of 2 KB
- As JMP is in the same page ,only 11 bits of address will change

Syntax: **AJMP sadd**;// Absolute Jump using the short address

Range: **max 2KB** as long as the Jump is within the **Same Page**

Size of instruction: **2 Bytes** (Opcode of AJMP= 1Byte, sadd = 1Byte)

New address calculation:

PC ← (16)	**PC**	**Opcode of AJMP**	**Sadd**
	5 bits Remains the same as branch is in the same page	3 bits Hence AJMP has 8 opcodes	8 bits Lower 8 bits of the jump location

- 3. Long Jump(LJMP ladd)
  - ladd- 16 bit address (0000H to FFFFH)
  - 3 bytes (1 byte opcode +2 byte ladd)
  - Eg: LJMP 5000H
  - LCALL also like this

	Jump operation	Call operation
1	In a Jump, we simply branch to the new location and continue from there on.	In a Call, we branch to the new location, which is basically a subroutine, we execute the subroutine, and at the end, we return to the main program right at the NEXT instruction after the Call instruction.
2	There is no intention to return to the previous location..	To invoke the subroutine we use Call instruction. To return to the main program we use RET instruction.
3	There is no need for storing any return address into the stack	During Call, the return address (PC), is pushed into the stack. During RET, the return address is popped from the stack and put back into PC.
4	Jumps can be of three types: Short, Absolute or Long Jump.	Call can be of two types: Absolute or Long Call.
5	Furthermore, Jumps can be unconditional or conditional.	Calls are always unconditional in 8051.

	RET	RETI
1	RET is used at the end of an ordinary Subroutine	RETI is used at the end of an ISR
2	RET will simply return back to the next instruction of the main program.	RETI will not only return back to the next instruction of the main program, but will also re-enable the interrupts, by making EA bit ← 1.
3	Operation: **POP PC;** PCH← [SP] PCL← [SP-1] SP← SP-2	Operation: **POP PC;** PCH← [SP] PCL← [SP-1] SP← SP-2 **EA ←1;** Enables interrupts by making EA bit "1" in the IE-SFR.

Conditional jump instructions

- 1.CJNE A,#25H ,label

(Compare & jmp if not equal)

- CJNE A,25H ,label
- CJNE R0,#25H ,label
- CJNE @R0,25H ,label
- 2. DJNZ count, label (Decrement and jmp if not Zero)
- DJNZ R2,BACK
- DJNZ 25H,BACK

- JC label
- JNC label
- JZ label
- JNZ label

Bit wise Conditional jump instructions

- JB bit,label

Eg: JB P0.2,down

- JNB bit,label

Eg: wait :JNB TF0,wait

- JBC bit,label *(jmp if bit =1 then goto label ,while jumping clear bit also)*

Eg: JBC P0.7,down, *(if p0.7=1 jmp to locn down and make po.7=0)*

CALL /SUBROUTINE INSTRUCTIONS

- Subroutines are handled by CALL and RET instructions There are two types of CALL instructions
- **LCALL address(16 bit)**
- This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
-
- a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254; during execution of this instruction [PC] = 3254h + 3h = 3257h

  b. [SP]=[SP]+1;

  c. [[SP]] = [PC7-0];

  d. [SP]=[SP]+1;

  e. [[SP]] = [PC15-8];

ACALL address(11 bit)

- This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The ACALL instruction works as follows.

  a. During execution of ACALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh

  b. [SP]=[SP]+1

  c. [[SP]] = [PC7-0]

  d. [SP]=[SP]+1; (SP increments again and [SP]=09)

  e. [[SP]] = [PC15-8]

  With these the address (0x854B) which was in PC is stored in stack.

  f. [PC10-0]= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.

- g. [PC15-8] = [[SP]]
- h. [SP]=[SP]-1
- i. [PC7-0] = [[SP]]
- j. [SP]=[SP]-1; (SP decrements again)

Boolean/ BIT MANIPULATION INSTRUCTIONS

- 8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.

- **1.LOGICAL AND**
  - a. ANL C,BIT

  Eg: ANL C, P0.2

  ANL C, 25H

  - b. ANL C, /BIT;

  Eg: ANL C, / P0.3 (C ^/P0.3)
- **2. LOGICAL OR**
  - a. ORL C,BIT Eg: ORL C, P0.2
  - b. ORL C, /BIT; Eg: ORL C, /P0.2

3. CLR bit

- a. CLR bit Eg: CLR P0.2
- b. CLR C (C□-- 0)

4. CPL bit

a. CPL bit Eg: CPL P0.2

b. CPL C

c. CPL 02H

5. SETB C
6. SETB bit eg: SETB PSW.0
7. CLR C
8. CLR bit eg: CLR P0.1
9. MOV C,bit

Eg: MOV C, P0.3

MOV C,25H

10. MOV bit,C

Addition of two 8-bit nos

```
MOV  DPTR, #4200H
MOVX  A, @DPTR
MOV  B, A
INC  DPTR
MOVX  A, @DPTR
MOV  R0, 00H
ADD A, B
JNC  L1
INC RO
L1 :  INC DPTR
      MOVX @DPTR, A
      INC DPTR
      MOV A, R0
      MOVX @DPTR, A
HERE : SJMP   HERE
```

Subtraction of two 8-bit nos

```
MOV  DPTR, #4200
MOVX  A, @DPTR
MOV  B, A
INC  DPTR
MOVX  A, @DPTR
MOV  R0, 00H
CLR C
SUBB A, B
JNC  L1
CPL A
INC A
INC Ro
```

```
L1 :  INC DPTR
      MOVX @DPTR, A
       INC DPTR
        MOV A, Ro
        MOVX @DPTR, A
HERE : SJMP   HERE
```

Multiply two 8-bit nos

```
MOV DPTR, #4200H
MOVX A, @DPTR
MOV B,A
INC  DPTR
MOVX A, @DPTR
MUL AB
INC DPTR
MOVX @DPTR, A
MOV A, B
INC DPTR
MOVX @DPTR, A
HERE : SJMP   HERE
```

- [4200] =FF
- [4201]=FF

FF x
FF

FE 01

DIVIDE TWO 8-BIT NOS

```
MOV DPTR, #4200H
MOVX A, @DPTR
MOV Ro , A
INC  DPTR
MOVX A, @DPTR
MOV B, A
MOV A,Ro
DIV  AB
INC DPTR
MOVX @DPTR, A
MOV A, B
INC DPTR
MOVX @DPTR, A
HERE : SJMP   HERE
```

Find the largest no in the array

```
MOV DPTR ,#5000H
MOVX A, @DPTR
MOV Ro, 05H
L1 : MOV B, A
L3:DJNZ R0,L2
   SJMP L4
L2:  INC DPTR
     MOVX A, @DPTR
     CJNE A,B ,NEG
      SJMP L3
 NEG:    JC L3
     SJMP L1
L4:  MOV A, B
       INC DPTR
        MOVX @DPTR,A
HERE : SJMP   HERE
```

WAP to add the contents of Internal RAM locations 40H and 41H. Store Result at 42H and Carry at 43H

```
        MOV 43H, #00H       ; Initialize Carry as "0"
        MOV A, 40H          ; Read first number
        ADD A, 41H          ; Add second number
        JNC SKIP            ; If no Carry, directly store the Sum
        INC 43H             ; Store Carry as "1"
SKIP:   MOV 42H, A          ; Store Sum
HERE:   SJMP HERE           ; End of program
```

Q2 WAP to multiply the numbers B2H and 2FH. Store Result in registers R0 (LSB) and R1 (MSB) of Bank 2.

```
SOLN:  MOV A, #0B2H        ; Read first number
       MOV 0F0H, #2FH      ; Read second number
       MUL AB              ; Multiply the operands
       SETB PSW.4          ; Select Bank 2
       CLR  PSW.3
       MOV R0, A           ; Store LSB of result
       MOV R1, 0F0H        ; Store MSB of result
HERE:  SJMP HERE           ; End of program
```

Q3

WAP to add a series of 10 numbers. The series begins from location 20H in Internal RAM. Store the result at locations 30 and 31H.

```
SOLN:   MOV R0, #20H          ; Initialize Source address
        MOV R1, #0AH          ; Initialize count of 10
        CLR A                 ; A register will accumulate the Sum
        MOV 0F0H, #00H        ; B register will accumulate the Carry
REPEAT: ADD A, @R0            ; Add the current element
        JNC SKIP              ; If no carry, then directly proceed ahead
        INC 0F0H              ; If there is a carry, increment B Register
SKIP:   INC R0                ; Increment source address
        DJNZ R1, REPEAT       ; Decrement count. If Count is NOT ZERO then repeat.
        MOV 30H, A            ; Store Sum
        MOV 31H, 0F0H         ; Store Carry
HERE:   SJMP HERE             ; End of program
```

Q4 **WAP to add a series of 10 "BCD" numbers. The series begins from location 20H. Store the result at locations 30 and 31H.**

```
SOLN:   MOV R0, #20H          ; Initialize Source address
        MOV R1, #0AH          ; Initialize count of 10
        CLR A                 ; A register will accumulate the Sum
        MOV 0F0H, #00H        ; B register will accumulate the Carry
REPEAT: ADD A, @R0            ; Add the current element
        DA A                  ; Decimal adjust as the operands were BCD numbers
        JNC SKIP              ; If no carry, then directly proceed ahead
        INC OFOH              ; If there is a carry, increment B Register
SKIP:   INC R0                ; Increment source address
        DJNZ R1, REPEAT       ; Decrement count. If Count is NOT ZERO then repeat.
        MOV 30H, A            ; Store Sum
        MOV 31H, 0F0H         ; Store Carry
HERE:   SJMP HERE             ; End of program
```

Q5 WAP to add a series of 10 numbers. The series begins from location 2000H in External RAM. Store the result at locations 3000 and 3001H.

```
  SOLN:  MOV DPTR, #2000H      ; Initialize Source address
         MOV R1, #0AH          ; Initialize count of 10
         CLR A                 ; "A" will accumulate the Sum, and ALSO get the data from Ext. RAM
         MOV 0F0H, #00H        ; B register will accumulate the Carry
         MOV R0, #00H          ; R0 will be used to
REPEAT:  MOVX A, @DPTR         ; Get the data from Ext RAM
         ADD A, R0             ; Add the data to the previous sum
         JNC SKIP              ; If no carry, then directly proceed ahead
         INC OFOH              ; If there is a carry, increment B Register
  SKIP:  INC DPTR              ; Increment source address
         MOV R0, A             ; Store current Sum in R0 from next iteration
         DJNZ R1, REPEAT       ; Decrement count. If Count is NOT ZERO then repeat.
         MOV DPTR, #3000H      ; DPTR gets address to store the Sum
         MOVX @DPTR, A         ; Store Sum
         INC DPTR              ; DPTR gets address to store the Carry
         MOV A, OFOH           ; Transfer Carry from B to A register
         MOVX @DPTR, A         ; Store Carry
  HERE:  SJMP HERE             ; End of program
```

Q6 WAP to copy the value 25H to all locations from 2000H to 2100H in the External RAM.

```
SOLN:  MOV A, #25H          ; Number stored in A as MOVX works only on "A"
       MOV R0, #00H         ; R0 will contain the count
       MOV DPTR, #2000H     ; DPTR contains the Dest address
BACK:  MOVX @DPTR, A        ; Store at Ext RAM location pointed by DPTR
       INC DPTR             ; Inc Dest address
       INC R0               ; Inc Count
       CJNE R0, #00H, BACK  ; Check if RO has again become 0. If not: repeat
       MOVX @DPTR, A        ; Store at 2100H
HERE:  SJMP HERE            ; End of program
```

In the above program, you must remember that an 8-bit register like R0 will give 2^8 i.e. 256 counts, Hence the extra step after the loop is required.