8051 Instructions

The instructions of 8051 can be broadly classified under the following headings.

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

Data transfer instructions.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A
- *i.* MOV A,R2
- *ii.* MOV A,R7

b. Move the contents of a register A to Rn
- *i.* MOV R4,A
- *ii.* MOV R1,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
- *vi.* MOV DPTR, #5467H

d. Move the contents of a memory location to A or A to a memory location using directand 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 47H, 65H
- *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

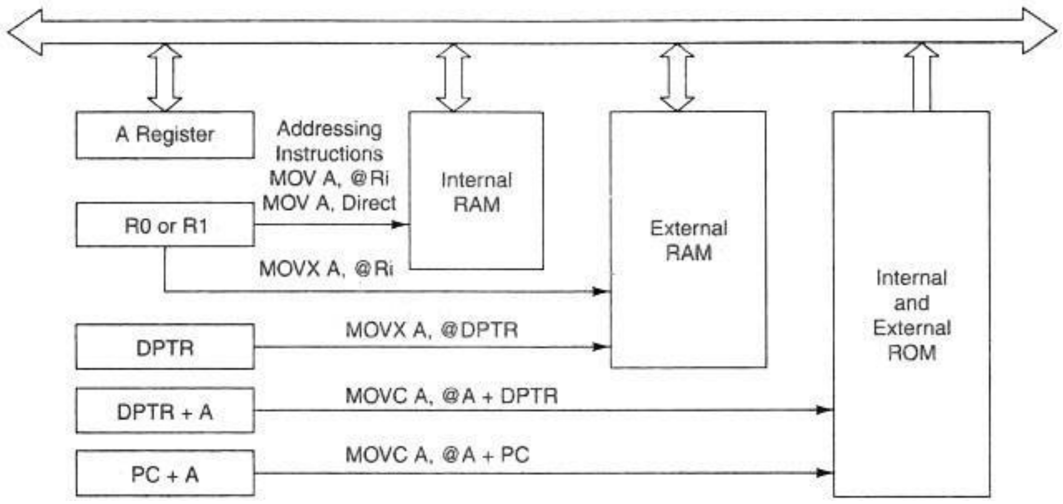


FIG. Addressing Using MOV, MOVX and MOVC

i. Push and Pop instructions

```
                  [SP]=07        //CONTENT OF SP IS 07 (DEFAULT VALUE)
MOV R6, #25H      [R6]=25H       //CONTENT OF R6 IS 25H
MOV R1, #12H      [R1]=12H       //CONTENT OF R1 IS 12H
MOV R4, #0F3H     [R4]=F3H       //CONTENT OF R4 IS F3H

PUSH 6            [SP]=08        [08]=[06]=25H //CONTENT OF 08 IS 25H
PUSH 1            [SP]=09        [09]=[01]=12H //CONTENT OF 09 IS 12H
PUSH 4            [SP]=0A        [0A]=[04]=F3H //CONTENT OF 0A IS F3H

POP 6             [06]=[0A]=F3H [SP]=09        //CONTENT OF 06 IS F3H
POP 1             [01]=[09]=12H [SP]=08        //CONTENT OF 01 IS 12H
POP 4             [04]=[08]=25H [SP]=07        //CONTENT OF 04 IS 25H
```

j. Exchange instructions

The content of source i.e., register, direct memory or indirect memory will beexchanged with the contents of destination ie., accumulator.

i. XCH A,R3
ii. XCH A,@R1
iii. XCH A,54h

k. 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

Arithmetic instructions.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

Addition

In this group, we have instructions to

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, @R1
   iv. ADDC A, @R0

CY AC and OV flags will be affected by this operation.

Subtraction

In this group, we have instructions to

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
   iv. SUBB A, @R0

CY AC and OV flags will be affected by this operation.

Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

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

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Eg.     MOV A,#45H          ;[A]=0E8H
        MOV B,#0F5H         ;[B]=1BH
        DIV AB              ;[A] / [B] = E8 /1B = 08 H with remainder  10H
                            ;[A] = 08H, [B]=10H
```

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.

```
Eg 1:   MOV A,#23H
        MOV R1,#55H
        ADD A,R1        // [A]=78
        DA A            // [A]=78          no changes in the accumulator after da a

Eg 2:   MOV A,#53H
        MOV R1,#58H
        ADD A,R1        // [A]=ABh
          DA A          // [A]=11, C=1 . ANSWER IS 111. Accumulator data is changed after DA A
```

Increment: *increments the operand by one.*

INC A **INC Rn** **INC DIRECT** **INC @RiINC DPTR**

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.

Decrement: *decrements the operand by one.*

DEC A **DEC Rn** **DEC DIRECT** **DEC @Ri**

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 0 to FFh.

Logical Instructions

Logical AND

ANL destination, source:ANL does a bitwise "AND" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.
ANL A,#DATA ANL A, Rn
ANL A,DIRECT ANL A,@Ri
ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORL destination, source:ORL does a bitwise "OR" operation between *source* and *destination*,

leaving the resulting value in *destination*. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.
ORL A,#DATA ORL A, Rn
ORL A,DIRECT ORL A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source:XRL does a bitwise "EX-OR" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.
XRL A,#DATA XRL A,Rn
XRL A,DIRECT XRL A,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

CPL complements *operand*, leaving the result in *operand*. If *operand* is a single bit then the state of the bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed.

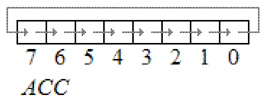
CPL A, CPL C, CPL bit address

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

Rotate Instructions

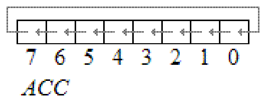
RR A

This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



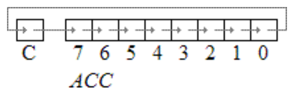
RL A

Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



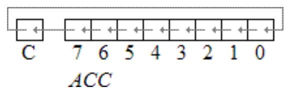
RRC A

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 (JUMP) Instructions

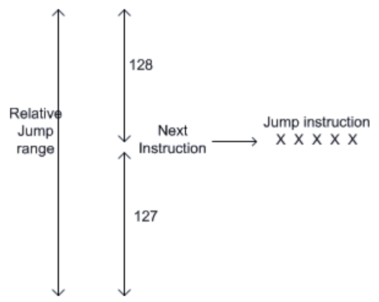
Jump and Call Program Range

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

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump

SJMP <relative address>; *this is unconditional jump*

The remaining relative jumps are conditional jumps

JC <relative address>
JNC <relative address>
JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex)	*Address (Hex)*
00	*0000 - 07FF*
01	*0800 - 0FFF*
02	*1000 - 17FF*
03	*1800 - 1FFF*
.	
.	
1E	*F000 - F7FF*
1F	*F800 - FFFF*

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2 bytes.

Example of short absolute jump: -

```
ACALL <address 11>
AJMP   <address 11>
```

Long Absolute Jump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.

Example: -

```
LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR
```

Another classification of jump instructions is

1. Unconditional Jump
2. Conditional Jump

1. **The unconditional jump** is a jump in which control is transferred unconditionally to the target location.
   a. **LJMP** (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH
   eg: LJMP 3000H
   b. **AJMP:** this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.
   c. **SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump.

2. **Conditional Jump instructions.**

JBC	Jump if bit = 1 and clear bit
JNB	Jump if bit = 0
JB	Jump if bit = 1
JNC	Jump if CY = 0
JC	Jump if CY = 1
CJNE reg,#data	Jump if byte ≠ #data
CJNE A,byte	Jump if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A = 0

All conditional jumps are short jumps.

Bit level jump instructions:

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to the address specified in the instruction. All the bit jumps are relative jumps.

JB bit, rel	; jump if the direct bit is set to the relative address specified.
JNB bit, rel	; jump if the direct bit is clear to the relative address specified.
JBC bit, rel	; jump if the direct bit is set to the relative address specified and then clear the bit.

Subroutine CALL And RETURN Instructions

Subroutines are handled by CALL and RET instructions

There are two types of CALL instructions

1. **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; (if SP contains default value 07, then SP increments and [SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 57 will be stored in memory location 08.
   d. [SP]=[SP]+1; (SP increments again and [SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 32 will be stored in memory location 09.

   With these the address (0x3254) which was in PC is stored in stack.
   f. [PC]= address (16 bit); the new address of subroutine is loaded to PC. No flags are affected.

2. **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 SCALL instruction works as follows.
   a. During execution of SCALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 4B will be stored in memory location 08.
   d. [SP]=[SP]+1; (SP increments again and [SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 85 will be stored in memory location 09.

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

f. [PC_{10-0}]= 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. [PC_{15-8}] = [[SP]] ;content of current top of the stack will be moved to higher byte of PC.
h. [SP]=[SP]-1; (SP decrements)
i. [PC_{7-0}] = [[SP]] ;content of bottom of the stack will be moved to lower byte of PC.
j. [SP]=[SP]-1; (SP decrements again)

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(BIT ADDRESS) ; 'LOGICALLY AND' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ANL C, /BIT; ; 'LOGICALLY AND' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY

2. LOGICAL OR
   a. ORL C,BIT(BIT ADDRESS) ; 'LOGICALLY OR' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ORL C, /BIT; ; 'LOGICALLY OR' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
3. CLR bit
   a. CLR bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE CLEARED.
   b. CLR C ; CONTENT OF CARRY WILL BE CLEARED.
4. CPL bit
   a. CPL bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE COMPLEMENTED.
   b. CPL C ; CONTENT OF CARRY WILL BE COMPLEMENTED.