Chapter

7 8051 INSTRUCTION SET

A processor performs a job by reading and executing the set of instructions written in its program memory. This set of instructions to perform a specific job is called a program. Each processors has its own set of instructions. 8051 has 111 different types of instructions with 241 total numbers of instructions opcodes. These instructions are one byte, two byte and three byte long. An instruction may not have any operand or can have one or maximum two operands. Each instruction is normally made up of three parts :

part 1	part 2	part 3
mnemonic	Destination operand	Source operand

part 1, the instruction mnemonic tells the processor the operation it has to perform like ADD, SUB tract, Data MOVE etc.

part 2, the destination operand is an operand or data address that specifics the destination for the data.

part 3, the source operand is an operand or data address and specifies the source for the data.

8051 Instruction set can be divided into following five types :

(*i*) Data Transfer Instructions
(*ii*) Arithmetic Instructions
(*iii*) Logical Instructions
(*iv*) Boolean Instructions
(*v*) Program Branching Instructions.

7.1 DATA TRANSFER INSTRUCTIONS

Data transfer instructions are used to copy a data from one location to another. The location may be a register or a memory location or an I/O port. The following five types of opcodes are used to move data :

(*i*) MOV
(*ii*) MOVX

(iii) MOVC
(iv) PUSH and POP
(v) XCH and XCHD.

Data transfer instructions do not affect any flag of the PSW register. However, if PSW is specified as destination in MOV, POP or XCH instruction, then flags gets affected.

7.1.1 MOV Instruction

MOV destination-byte, source-byte

This instruction copies a byte from source to destination. There are different possible combinations for this instruction. Registers A, DPTR, and R0-R7 of current register bank can also be specified by their names while other registers can be specified by their address only. The source can be a internal data memory location (internal RAM and SFR), or directly a data byte specified in opcode. Some combinations of MOV instructions are :

MOV A, #*n* : Copies the immediate data byte *n* to the register A. MOV A, # 85 H Copies data byte 85 H to register A.

MOV A, R_n : Copies the contents of register R_n (R0-R7) to register A. MOV A, R5 copies the contents of register R5 of current register bank into Register A.

MOV R_n, A : Copies the contents of register A into register R_n (R0-R7) MOV R3, A copies the contents of register A into R_3.

MOV R_n, 8 bit Add : Copies the contents of internal data memory specified by 8-bit address into register R_n.

MOV 8-bit Address, @ R_i : Copies the contents of internal data memory specified by register R_i (R0 or R1) into memory location specified by 8-bit address.

MOV DPTR, # 16-bit Data : Copies 16 bit data specified in the instruction into DPTR register.

Note : A data MOV instruction does not affects the contents of source location.

7.1.2 Addressing Modes

Addressing modes means method of specifying operand in a instruction. 8051 supports five addressing modes :

(i) Register Addressing Mode
(ii) Immediate Addressing Mode
(iii) Direct Addressing Mode

(*iv*) Register-Indirect Addressing Mode

(*v*) Indexed Addressing Mode

7.1.3 Register Addressing Mode

In register addressing mode, operand lies in a register and is specified by name of the register. In 8051, Registers A, DPTR and R0 to R7 may be named as a part of opcode mnemonic.

Example :

MOV A, R2 : Moves the contents of source register R2 to destination register A. In this instruction, both the operands (source and destination) are specified using register addressing mode. Contents of register R2 remains unchanged.

MOV R7, A : Moves the contents of source register A to register R7 while the contents of register A remains unchanged.

7.1.4 Immediate Addressing Mode

In immediate addressing mode, the operand is a part of the instruction itself. There is no need to specify register or memory location for operand. Immediate addressing mode can be used for source operand only, not for destination operand. In an instruction, only one operand can be specified using immediate addressing mode.

```
MOV A, #4F H          ;   Loads 4FH into A
MOV R1, # 18 H        ;   Loads 18 H into R1
MOV DPTR, # 3982 H    ;   Loads 16 bit data 3982H into 16-bit register DPTR.
```

symbol is used to indicate immediate addressing mode.

7.1.5 Direct Addressing Mode

In direct addressing mode, the address of the memory location, where operand lies is directly specified in the instruction. In 8051, if there are two operands, then both operands also can be specified by direct addressing mode. Only internal RAM (00H to 7FH) and SFRs (80 H to FFH) with 8-bit address can be specified using direct addressing mode not the external data memory.

Example :

MOV R0, 23H : Copies the contents of RAM location 23H into register R0.

MOV A, 81 H : Copies the contents of location 81 H i.e. SP special function register into register A.

MOV D0H, R7 : Copies the contents of register R7 into location with address D0H *i.e.* PSW register.

MOV 8AH, 03H ; Copies the contents of RAM location 03H (Register R3 of Register bank 0) into location with address 8AH i.e. TL0 Timer 0 register.

7.1.6 Register Indirect Addressing Mode

In register indirect addressing mode, a register is specified which contains the address of the memory location where operand lies. If the operand is in internal RAM or in SFR then only registers R0 and R1 can be used to specify the 8-bit address of the memory location. The sign @ is prefixed R0/R1 to indicate register indirect addressing mode :

MOV A, @ R0 : Copy the contents of internal data memory location whose address is held by R0, into A.

MOV @ R1, A : Copy the contents of A into memory location (internal) whose address is held by R1.

Note that command MOV A, R0 copies the contents of R0 into A, instead of the contents of memory location pointed by R0.

If the operand is in external data memory, then registers R0 and R1 can be used to specify the 16-bit address of the memory location from 0000H to 00FFH, *i.e.* R0 and R1 specify only lower byte of the address and upper byte is 00H. 16-bit register DPTR can also be used to specify the 16 bit address of external data memory location from 0000H to FFFFH. MOVX command is used for this purpose.

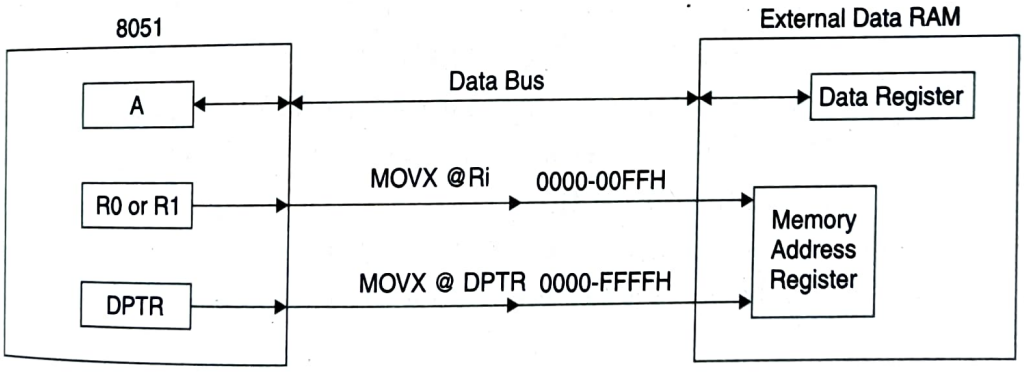


Fig. 7.1 : External Data Memory Access

MOVX A, @ R1 : Copies the contents of external data memory location, whose address lower byte is held by R1 and upper byte is 00H, into register A.

MOVX @ R0, A : Copies the contents of A into external data memory location, whose address lower byte is held by R0 and upper byte is 00H.

MOVX A, @ DPTR : Copies the contents of external data memory location pointed by DPTR (16 bit) into A.

MOVX @ DPTR, A : Copies the contents of A into external data memory location pointed by register DPTR.

Note : Symbol @ is prefixed with R0, R1 and DPTR to indicate register indirect addressing mode. Only register "A" can be used to exchange 8-bit data with external data memory and the command used is MOVX.

7.1.7 Indexed Addressing Mode

In Indexed Addressing mode, the contents of two registers are added to form the 16-bit address of the memory location where operand lies. In 8051, only program memory can be accessed using indexed addressing mode. This mode is widely used in accessing data elements of look-up table enteries. MOVC command is used for this purpose and contents of ROM can be transferred to accumulator (A) only.

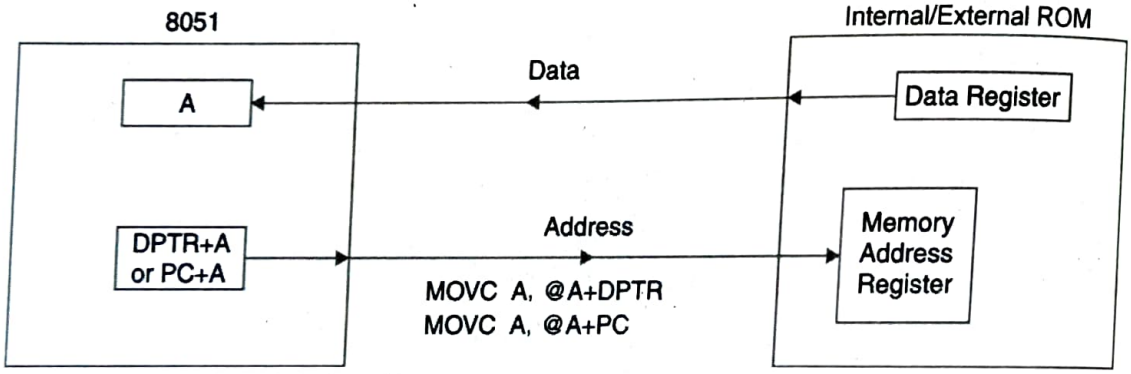


Fig. 7.2 : ROM Access for Code Data

MOVC A, @ A + DPTR : Moves the 8-bit contents of program memory location, whose address is obtained by adding contents of A and DPTR register, into register A.

MOVC A, @ A + PC : Moves the 8-bit contents of program memory location, whose address is obtained by adding contents of A and PC registers, into register A.

Here PC is incremented first to point to next instruction before added to A to get the address.

7.1.8 MOVX Instruction

MOVX instruction is used for external data transfers. This instruction transfers data between CPU register A and external data memory or I/O devices. To specify external memory, only register indirect addressing mode is available. Memory location cannot be specified using direct addressing mode or indexed addressing mode. If a data from any SFR other than A or any internal RAM location is to be

transferred to external data memory, then first that data is to be transferred to CPU register A using MOV instruction and then to external memory using MOVX instruction. X in MOVX instruction indicates external transfer.

Registers R0, R1 and DPTR can be used to hold the memory address. Registers R0, R1 are 8-bit registers and hold only lower byte of the address. Upper byte of the address is taken as 00H. DPTR register holds 16-bit address.

- **MOVX A, @ R0 :** Copy data from external data memory location (0000H to 00FFH), whose lower 8-bit address is in R0, to register A.
- **MOVX A, @ R1 :** Copy data from external data memory location (0000H to 00FFH), whose lower 8-bit address is in R1, to register A.
- **MOVX @ Rp, A :** Copy the contents of A to external data memory location (0000H to 00FFH) whose lower 8-bit address is in register Rp (R0 or R1).
- **MOVX A, @ DPTR :** Copy the contents of external data memory, whose 16-bit address is in DPTR, to register A.
- **MOVX @ DPTR, A :** Copy the contents of register A into external data memory, whose 16 bit address is in DPTR.

7.1.9 MOVC Instruction

MOVC instruction is used to read data stored in program memory (internal ROM and/or external ROM). All data is moved from the program memory to the register A. Code memory location can be specified only by indexed addressing mode. Registers A and PC/DPTR can be used for this purpose. This statement is normally used to read preprogrammed data bytes like look-up table entries which are stored in program area alongwith the codes. Letter 'C' in MOVC instruction indicates code memory.

- **MOVC A, @ A + DPTR :** Copy a byte stored at program memory location, whose address is formed by adding contents of A and DPTR, into register A.
- **MOVC A, @ A + PC :** Copy a byte stored at program memory location, whose address is formed by adding the contents of A and PC, into register A. The PC is incremented by 1 (to point to the next instruction) before it is added to A to form the address of ROM.

7.1.10 PUSH and POP Instructions

Push and Pop instructions are used to store and get back 8-bit data from stack. In 8051, internal RAM area can be used as stack. An 8-bit SFR SP is used as pointer

to stack area and it contains address of the top most filled position of the stack. Default value of SP is 07H. Stack area can grow upto maximum 7FH. Only direct addressing mode can be used to specify the operand in PUSH and POP instructions.

PUSH instruction copies a byte of data from location specified by 8-bit address in the instruction, to the top of the stack. Stack pointer SP is incremented by 1 before the data is copied onto stack location pointed by SP. Stack grow up in memory as it is PUSHED. Excessive Pushing can make the stack exceed 7FH, after which data will be lost.

Example :

PUSH 42H : Copies the byte stored at location 42 H onto stack location formed by incrementing SP.

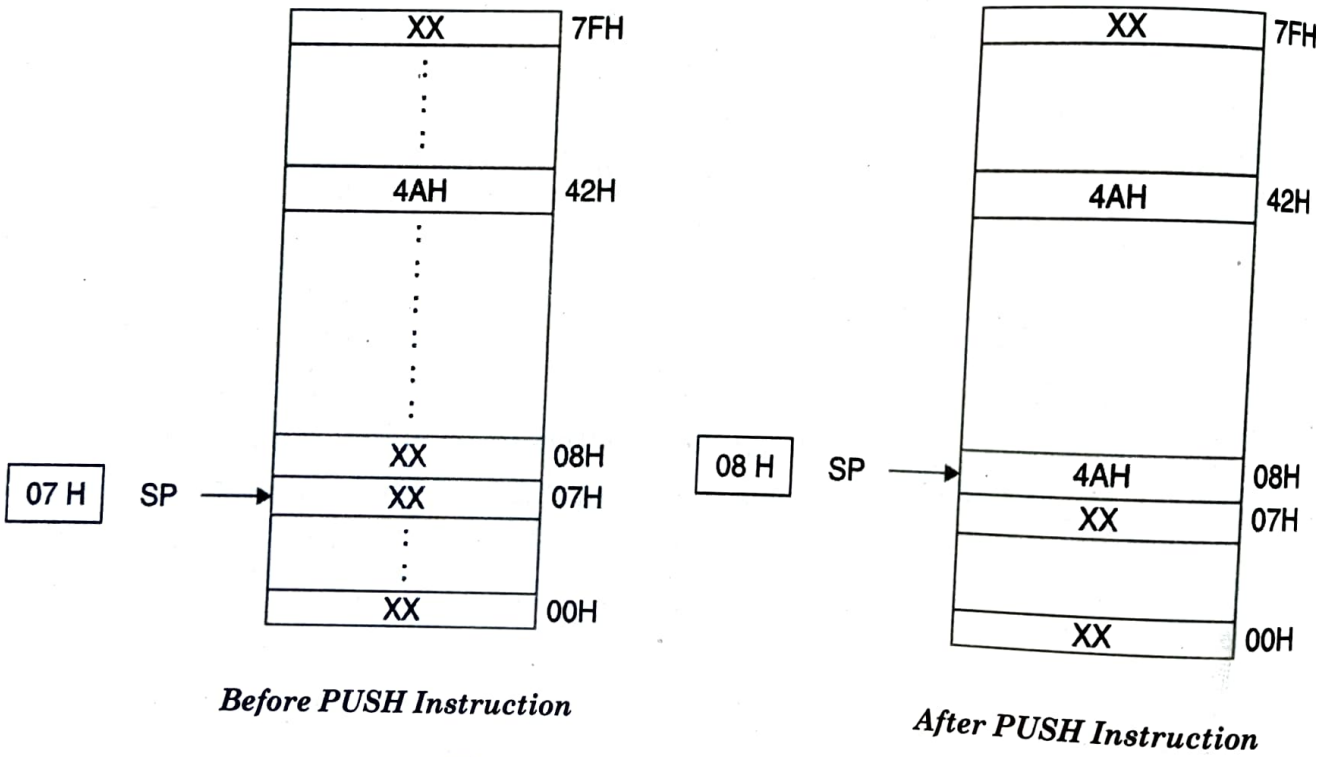


Fig. 7.3 : Push Instruction

POP instruction copies a byte of data from top of stack pointed by SP into the location specified by 8-bit address in the instruction. SP is decremented by one after the data is copied.

Example :

POP 7FH : Copies the byte at the top of the stack pointed by SP into memory location 7FH.

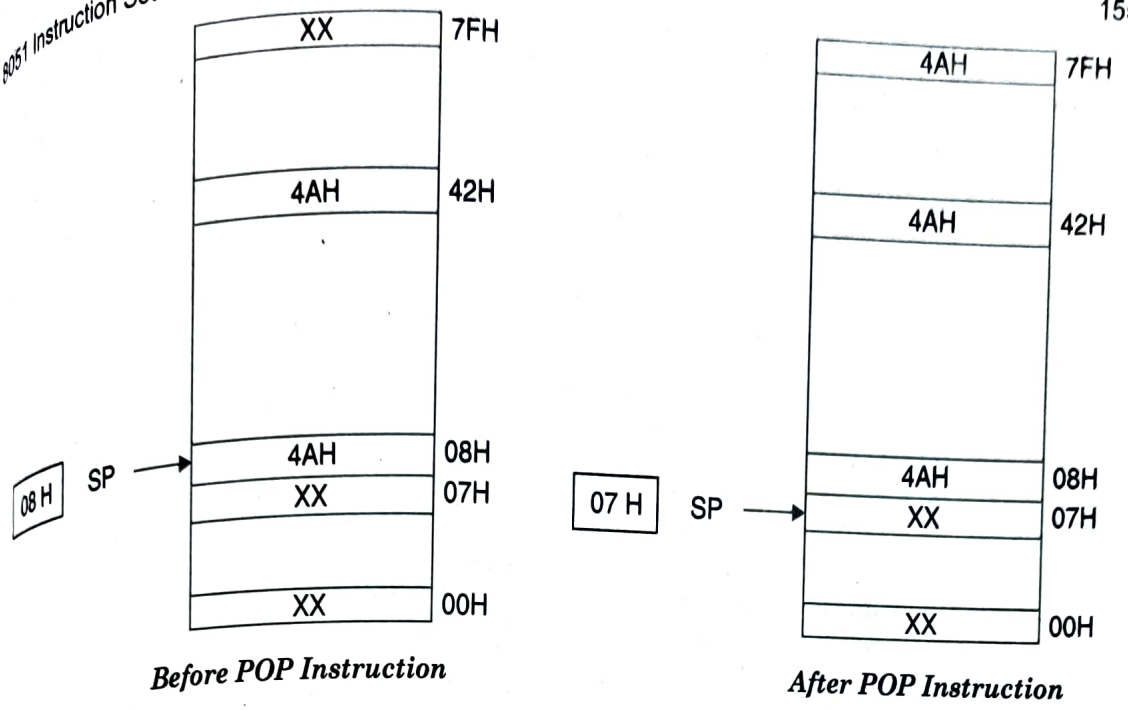


Fig. 7.4 : Pop Instruction

Exp. for POP and PUSH Instructions

Instruction	Operation
MOV 81 H, # 50H	Copy immediate data 50H to SP (Address 81H).
MOV R1, #3FH	Copy immediate data 3FH to R1.
MOV R2, #49 H	Copy immediate data 49H to R2
PUSH 01H	SP = 51 H; Mem. location 51 H contains data 3FH.
PUSH 02H	SP = 52 H; Mem. location 52 H contains data 49 H.
POP 07H	Register R7 (address 07H).
	now has data 49 H; SP = 51 H.
POP 90 H	Port 1 Latch (address 90H) now has data 3FH; SP = 50 H.

7.1.11 XCH and XCHD Instructions

XCH instruction exchanges the contents of register 'A' with a SFR or internal RAM location contents. The internal RAM location and SFR can be specified using Register Addressing Mode, Direct Addressing Mode and Register Indirect Addressing Mode.

XCH A, Rn : Exchange contents of A and register Rn (R0-R7).

XCH A, add : Exchanges contents of A and memory location whose 8-bit address is specified in the instruction.

XCH A, @ Rp : Exchanges the contents of A and memory location whose 8-bit address is specified by register Rp (R0, R1)

Exchange between A and any port (P0-P3) location copy the data on the port pins to A, whereas data in A is copied to the port latch.

Examples :

- **XCH A, R4** : Exchange bytes between A and R4, i.e. if before instruction A = 23H and R4 = 67 H then after instruction A = 67 H and R4 = 23 H.
- **XCH A, D0H :** Exchange bytes between A and register PSW (address DOH).
- **XCH A, @ R1 :** Exchanges bytes between A and the memory location/SFR whose address is in R1.

XCHD instruction exchanges only the lower nibble (4 bites) of A with the lower nibble of the RAM location/SFR specified in the instruction without changing the upper nibbles of both operands. RAM location/SFR can be specified using only Register-Indirect addressing mode.

XCHD A, @ Rp : Exchange the lower nibble of A and the location whose address is in Rp (R0, R1). Upper nibbles do not changes.

Example :

Instruction	Operation
MOV R3, #58 H	Copy immediate data 58 H to R3.
MOV A, #73 H	Copy immediate data 73 H to A
MOV R0, #03 H	Copy immediate data 03 H (address of R3) to R0.
XCHD A, @ R0	Exchanges the lower nibbles of A and memory location whose address is in R0 i.e. A = 78 H and R3 = 53 H after the instruction.

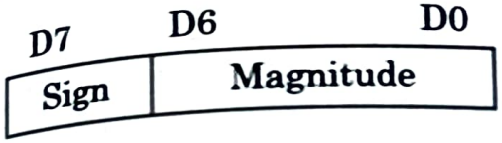
7.2 ARITHMETIC INSTRUCTIONS

8051 has arithmetic instructions for addition, subtraction, increment, decrement, multiply, divide, and decimal adjust for BCD operation. In total there are eight different types of arithmetic instructions. Most of these instructions affect flags (carry, overflow and Auxillary Carry) to depict the status of result of arithmetic operation.

7.2.1 Signed and Unsigned Numbers

The programmer may decide that the number used in the program are to be unsigned numbers or signed number. Unsigned numbers are eight bit positive

binary numbers ranging from 00H to FFH. Signed numbers uses bit 7 (MSB) as sign bit and bits 0-6 for magnitude. If MSB bit is 1 then the number is negative and if it is 0 then the number is positive. All negative numbers are in 2's complement form. A signed 8-bit numbers can range between –128 to +127. While working with unsigned numbers, only carry flag is taken care but while working with signed numbers, overflow flag is also to be taken care.



Unsigned Addition

$$\begin{array}{rcl} 95\,d & = & 01011111b \\ 189\,d & = & 10111101b \\ \hline 284\,d & = & 100011100b = 284\,d \end{array}$$

↑ To carry flag

Signed Addition

$$\begin{array}{rcll} +100\,d & = & 01100100b & \\ +50\,d & = & 00110010b & \\ +150\,d & = & 10010110b & = -106\,d \end{array}$$

In above signed addition, the result should be +150 but as this exceeds the limit, the result is wrongly shown – 106.

In this case overflow flag will be set which indicates that the result has exceeded the range and hence incorrect.

7.2.2 ADD Instruction

ADD instruction adds Accumulator (A) contents with a 8 bit operand specified in instruction and places the result in Accumulator (A). All the three math flags i.e. CF, AF and OF are affected. The second operand, which is to be added with A, can be specified using Immediate Addressing Mode, Register addressing mode, Direct addressing mode, and register-indirect addressing mode and can be in internal RAM or in SFR.

- ADD A, #data : Adds A with immediate data and stores the result in A.
- ADD A, Rn : Adds A with register Rn (R0-R7) and stores the result in A.

- ADD A, address : Adds A with location contents whose 8-bit address is given in instruction.
- ADD A, @ Rp : Adds A with location contents whose 8-bit address is in register Rp (R0,R1).

In addition of signed numbers, special attention should be given to the overflow flag (OF) since it indicates any error in addition result. The overflow flag is set to 1 if:

(*i*) there is a carry from D6 to D7 and no carry from D7 out.

(*ii*) there is a carry from D7 out and no carry from D6 to D7.

Example :

Instruction	Operation
MOV A, #63 H	Copy A with immediate data 63 H
MOV R3, #27 H	Copy R3 with immediate data 27 H
ADD A, R3	Adds A and R3 and places result in A. Flag affected : After the instruction A = 8AH, R3 = 27H, CF = 0, OF = 1, AF = 0

7.2.3 ADDC Instruction

ADDC (Add with carry) instruction adds 'A' with an 8 bit operand specified in instruction and carry flag bit (before the execution of the instruction). Result is placed in A and all the three math flags are affected. Second operand can be specified using direct, immediate, register and register-indirect addressing modes like in ADD instruction. This instruction is normally used to add multibyte data.

Example :

To add 4385H and 2193H, ADDC instruction is used as shown below :

```
CLR C          ;  Clear Carry Flage CF = 0
MOV A, #85 H   ;  A = 85 H
ADDC A, #93 H  ;  A = 85 H + 93 H + 0 = 18 H, CF = 1
MOV R5, A      ;  R5 = A = 18 H
MOV A, #43 H   ;  A = 43 H
ADDC A, #21 H  ;  A = 43H + 21H + 1 = 65H, CF = 0
```

The result of addition is 6518H. Lower byte in R5 and upper byte in A.

7.2.4 SUBB A, Source byte Instruction

SUBB (Subtract with borrow) instruction subtracts source byte and carry flag from accumulator (A) and places the result in A. The steps for subtraction performed by internal hardware are :

(i) Take 2's complement of source byte plus carry bit.
(ii) Add this to A
(iii) Invert the carry

SUBB instruction affects all the three math flages. Source byte can be specified using immediate, register, direct, and register-indirect addressing mode.

```
SUBB A, #data     ;  A = A – data – CF
SUBB A, Rn        ;  A = A – Rn (R0 – R7) – CF
SUBB A, address   ;  A = A – Mem. (Address) – CF
SUBB A, @ Rp      ;  A = A – Mem (Address = Rp) – CF
```

Example :

```
SETB C            ;  Set carry CF = 1
MOV A, #89H       ;  A = 89 H
MOV R5, # 26H     ;  R5 = 26H
SUBB A, R5        ;  A = 89H – 26H – 1 = 62 H, CF = 0
```

7.2.5 INC byte Instruction

INC (increment) adds 1 to the SFR or RAM location specified by the operand. No flag is affected. The operand can be specified by register, direct and register-indirect addressing modes.

```
INC    A          ;  A = A + 1
INC    R4         ;  R4 = R4 + 1
INC    DPTR       ;  DPTR = DPTR + 1
INC    35H        ;  Contents RAM location 35H increment by one
INC    @ R0       ;  Contents of RAM location, whose address is in R0,
                     incremented by one.
```

7.2.6 DEC Byte Instruction

DEC (Decrement) instruction decrease the contents of specified SFR or RAM location by one. No flag is affected. The operands can be specified by register, direct and register-indirect addressing modes.

```
DEC    A          ;  A = A – 1
DEC    R2         ;  R2 = R2 – 1
DEC    30 H       ;  Decrease the contents of memory location 30H by
                     one.
```

```
DEC @  R1      ;  Decrease the contents of memory location, whose
                  address is in R1, by one.
```

7.2.7 MUL AB Instruction

MUL (Multiply) instruction multiplies an unsigned data byte in A with an unsigned data byte in B. 16-bit result is placed in A and B registers with lower byte in A and upper byte in B. Overflow and carry flags are affected. This instruction always clear the carry flag. Overflow flag is changed according to the product. If the product is greater then FFH then OF = 1 otherwise it is zero.

Example :

```
MOV A, #06H      ;  A = 06H
MOV F0H, #05 H   ;  B = 05 H
MUL AB           ;  B = 00 H, A = 1EH Flags OF = 0 CY = 0
```

7.2.8 DIV AB Instruction

DIV (Divide) Instruction divides accumulator (A) contents with contents of register B. Both data bytes are treated as unsigned numbers. After the division, quotient will be placed in 'A' and remainder in register B. Carry flag is always cleared by this instruction. Overflow flag is set high if division is by zero.

Example :

```
MOV A, #53 H     ;  A = 53 H (83 d)
MOV F0H, #0A H   ;  B = 0A H (10 d)
DIV AB           ;  A = 08H, B = 03 H
```

7.2.9 DAA Instruction

DA (Decimal Adjust accumulator after addition) instruction is used after addition of BCD numbers to convert the result back to BCD. The data is A is adjusted in the following two possible cases :

(*i*) It adds 6 to lower 4 bits of A if it is greater than 9 or if Auxillary carry is high.

(*ii*) It also adds 6 to the upper 4 bits of A if it is greater than 9 or if carry flag is high.

Example :

```
MOV A, #38 H     ;  A = 38H
ADD A, #25 H     ;  A = 38H + 25H = 5DH invalid BCD number
DA A             ;  A = 63 H valid BCD number.
```

7.3 LOGICAL INSTRUCTIONS

8051 has logical instructions for AND, OR, Exclusive OR, Rotate and Swapping of accumulator contents. None of the logical instruction affects flags except rotate through carry instructions which affects carry flag. Function of different logical instructions is explained below :

7.3.1 ANL destination, Source Instruction

ANL (AND-Logical) instruction, Logically AND the source byte with destination byte and places the result in destination byte. No flag is affected. Destination operand can be accumulator or direct addressed RAM location or SFR. Source operand can be specified by immediate, register, direct or register-indirect addressing mode.

ANL A, # data	;	A = A AND 8-bit immediate data.
ANL A, Rn	;	A = A AND Rn (R0 – R7)
ANL A, address	;	Performs logical AND of A and location (RAM/SFR) whose 8-bit address is given in instruction and places result in A
ANL A, @ Rp	;	Performs logical AND of A and location (RAM/SFR) whose 8-bit address is given by Rp (R0, R1) and places result in A.
ANL address, A	;	Performs logical AND of A and mem location (RAM/SFR) whose 8-bit address is given in the instruction and places result in memory location.
ANL address, #data	;	Performs logical AND of immediate data (8-bit) with memory location whose 8-bit address is given in the instruction and places the result in memory location.

Example :

MOV A, #25 H	;	A = 25H
MOV 42H, #67 H	;	Memory (Add = 42H) = 67 H
MOV R3, #1FH	;	R3 = 1FH
ANL A, #3A H	;	A = 25 H AND 3AH = 20H
MOV A, #77H	;	A = 77H
ANL A, R3	;	A = 77 H AND 1FH = 17H
ANL 42H, A	;	Memory (add = 42H) = 67 AND 17 = 07 H

7.3.2 ORL destination, source Instruction

ORL (OR-Logical) instruction logically OR the source byte with destination byte and places the result in destination byte. No flag is affected. Destination operand can be accumulator or direct addressed RAM location/SFR. Source operand can be specified by immediate, register, direct, or register-indirect addressing mode.

ORL A, #data	;	A = A OR 8 bit immediate data
ORL A, Rn	;	A = A OR Rn (R0 – R7)
ORL A, Address	;	Performs logical OR of A and RAM location/SFR whose 8-bit address is given in instruction and places result in A.
ORL A, @ Rp	;	Perform logical OR of A and RAM location/SFR whose 8-bit address is given by Rp (R0, R1) and places result in A.
ORL address, A	;	Perform logical OR of A and RAM location/SFR whose 8-bit address is given in the instruction and places the result in RAM location/SFR.
ORL address, #data	;	Performs logical OR of 8 bit immediated data with RAM location/SFR whose 8-bit address is given in the instruction and places the result in RAM location/SFR.

Example :

MOV A, #25H	;	A = 25H
MOV 42H, #67H	;	Memory (Add = 42H) = 67H
MOV R3, #1FH	;	R3 = 1FH
ORL A, #8A H	;	A = 25H OR 8AH = AFH
MOV A, #30 H	;	A = 30H
ORL A, R3	;	A = 30 H OR 1FH = 3FH
ORL 42H, A	;	Mem. location (add = 42 H) = 67 H OR 3FH = 7FH

7.3.3 XRL Destination, Source Instruction

XRL (Exclusive – OR Logical) instruction logically XOR the source byte with destination byte and places the result in destination byte. No flag is affected. Destination operand can be accumulator or direct addressed RAM location/SFR. Source operand can be specified by immediate, register, direct, or register-indirect addressing mode.

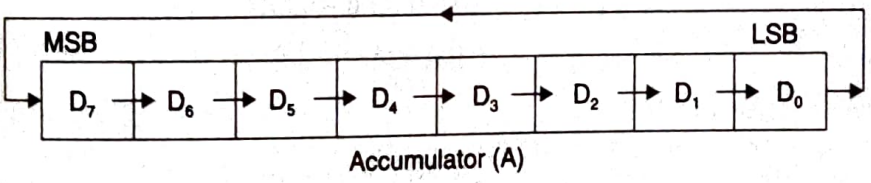
XRL A, #data	;	A = A XOR 8 bit immediate data
XRL A, Rn	;	A = A XOR Rn (R0 – R7)
XRL A, Address	;	Performs logical XOR of A and RAM location/SFR whose 8 bit address is given in instruction and places result in A.
XRL A, @ Rp	;	Performs logical XOR of A and RAM location/SFR whose 8-bit address is given by Rp (R0, R1) and places result in A.
XRL address, A	;	Perform logical XOR of A and RAM location/SFR whose 8-bit address is given in the instruction and places the result in RAM location/SFR.
XRL address, #data	;	Performs logical XOR of 8 bit immediate data with contents of RAM location/SFR whose 8-bit address is given in the instruction and places the result in the RAM location/SFR.

Example :

MOV A, #25H	;	A = 25H
MOV 42H, #67 H	;	Memory location (Add = 42H) = 67 H
MOV R3, #1FH	;	R3 = 1FH
XRL A, #0BAH	;	A = 25H XOR BAH = 9FH
XRL A, R3	;	A = 9FH XOR 1FH = 80 H
XRL 42H, A	;	Mem. location (add = 42 H)=67 H XOR 80 H = E7H

7.3.4 RR A Instruction

RR A (Rotate Right A) instruction rotates the 8-bit of contents of accumulator to right by one bit. Bit that comes out of D0 is stored in D7. No flag is affected.

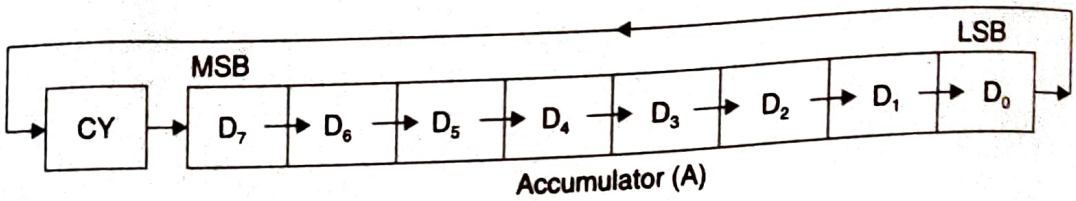


Example :

MOV A, #6A H	;	A = 01101010 b = 6AH
RR A	;	A = 00110101 b = 35H
RR A	;	A = 10011010 b = 9AH

7.3.5 RRC A Instruction

RRC A (Rotate Right Through carry the Accumulator) instruction rotates the 8-bit contents of accumulator to right by one bit through carry. Bit D0 goes into carry flag and carry bit into D7. Only carry flag is affected.

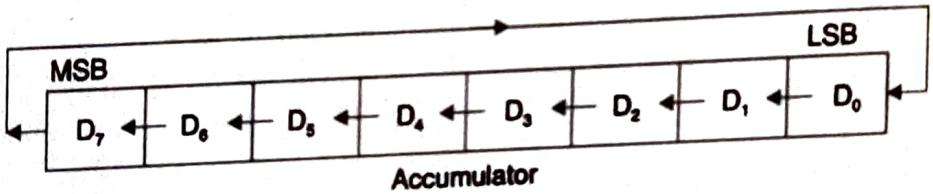


Example :

```
SETB C          ;  CY = 1
MOV A, #6A H    ;  A = 0110 1010 b = 6AH ; CY = 1
RRC A           ;  A = 10110101 b = B5H ; CY = 0
RRC A           ;  A = 01011010 b = 5AH ; CY = 1
```

7.3.6 RL A Instruction

RL A (Rotate Left Accumulator) instruction rotates the 8-bit contents of A to left by one bit. Bit D7 goes to bit D0. No flag is affected.

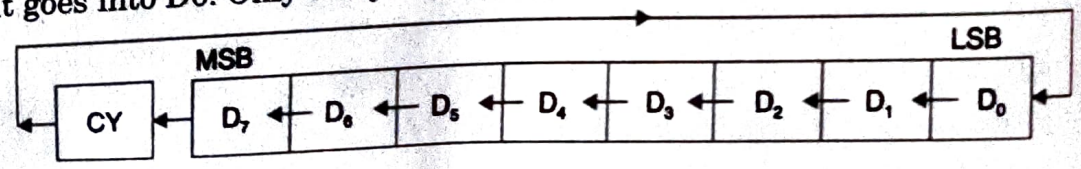


Example :

```
MOV A, # 6AH    ;  A = 01101010 b = 6AH
RL A            ;  A = 11010100 b = D4H
RL A            ;  A = 10101001 b = A9H
```

7.3.7 RLC A Instruction

RLC A (Rotate Left through carry the accumulator) instruction rotates the 8-bit contents of accumulator to left by one bit. Bit D7 goes into carry flag and carry flag bit goes into D0. Only carry flag is affected.



Example :

```
SETB C          ;  CY = 1
MOV A, #6AH     ;  A = 01101010 b = 6AH ; CY = 1
RLC A           ;  A = 11010101 b = D5H; CY = 0
RLC A           ;  A = 10101010 b = AAH; CY = 1
```

7.3.8 SWAP A Instruction

SWAP instruction interchanges lower nibble (D0 – D3) with upper nibble (D4 – D7) of accumulator A

Example :

```
MOV A, #38H     ;  A = 38 H
SWAP A          ;  A = 83 H
```

7.3.9 CLR A Instruction

CLR (Clear) instruction clears register A i.e. all bits of register A becomes zero. No flag is affected.

Example :

```
CLR A           ;  A = 00H
```

7.3.10 CPL A Instruction

CPL A (Complement accumulator) complements the contents of accumulator. No flag is affected.

Example :

```
MOV A, #38H     ;  A = 0011 1000 b = 38H
CPL A           ;  A = 11000111 b = C7H
```

7.4 BOOLEAN INSTRUCTIONS

8051 has instruction to operate on individual bits which makes it more suitable for control operations such as switching ON/OFF motors depending upon some switch positions. Carry flag works as accumulator being destination for most of boolean instructions. There are instructions that test a specified bit and transfer control to the desired location depending upon test result.

Boolean instruction work on only bit addressible RAM locations and SFRs. Of the 128-byte internal RAM, only 16 bytes are bit addressable. The RAM locations 20H to 2FH are both byte-addressable and bit addressable. Figure 7.5 shows byte address and bit address of bit addressable internal RAM. From the table, we can

Byte Address	Bit addresses							
	MSB							LSB
2FH	7F	7E	7D	7C	7B	7A	89	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
	D7	D6	D5	D4	D3	D2	D1	D0

Fig. 7.5 : Bit address of bit addressable internal RAM

find the bit address of any bit. For example, address of bit D4 of internal RAM byte address 29H is 4CH, the bit address of bit 6 of RAM address 21H is 0EH.

All the SFRs are byte-addressable but all the SFRs are not bit addressable. Only eleven SFRs are bit addressable also. The SFRs, that are also bit addressable, form the bit address by using the five most significant bits (MSB) of the byte address for that SFR together with the three least significant bits (LSB) that identify the bit position from 0 (LSB) to 7 (MSB).

For example Port 0 (P0) has byte address 80H. Bit 5 of P0 i.e. P0.5 has bit address 85H. CY bit of PSW, PSW.7 has bit address D7 H. Figure 2.6 shows bit address of all bit addressable SFRs.

Byte Address	Bit Addresses								SFR
	MSB							LSB	
F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
E0H	E7	E6	E5	E4	E3	E2	E1	E0	A
D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8H	–	–	–	BC	BB	BA	B9	B8	IP
B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8H	AF	–	–	AC	AB	AA	A9	A8	IE
A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0
	D7	D6	D5	D4	D3	D2	D1	D0	

Fig. 7.6 : Bit Addresses of bit Addressable SFRs

7.4.1 CLR C Instruction

CLR C (Clear Carry) instruction clears the carry flag i.e. after the execution of the instruction CY = 0

```
CLR    C          ;   CY = 0
```

7.4.2 CLR Bit Instruction

CLR bit (Clear Bit) instruction clears the single bit of bit addressable internal RAM or SFR specified using direct addressing mode. No flag is affected unless directly specified.

```
CLR P1.2          ;   P1.2 (Bit 2 of port 1) = 0
CLR ACC.0         ;   ACC.0 (Bit 0 of A) = 0
```

7.4.3 CPL C Instruction

CPL C (Complement carry) instruction complements the carry bit of PSW.

Example :

```
MOV D0H, #OFFH    ;   PSW (add = D0H) = FF, CY = 1
CPL C             ;   CY = 0
```

7.4.4 CPL bit Instruction

CPL bit (Complement bit) instruction complements the single bit specified in the instruction using direct addressing. Flags are not affected.

Example :

```
SETB P1.0        ;  P1.0 = 1
CPL P1.0         ;  P1.0 = 0
```

7.4.5 SETB C Instruction

SETB C (set bit) instruction sets high the carry bit

```
SETB C           ;  CY = 1
```

7.4.6 SETB bit instruction

SETB bit (set bit) instruction sets high the single bit of bit addressable internal RAM or SFR specified in the instruction using direct addressing mode.

```
SETB P2.4        ;  P2.4 (Bit 4 of Port 2 SFR latch) = 1
```

7.4.7 ANL C, bit Instruction

ANL C, bit (AND logical for bit variable) instruction 'AND' the carry bit with directly specified bit and places result in CY. Only CY flag affected.

```
MOV A, #01H      ;  A = 01H i.e. ACC.0 = 1, ACC.1 = 0
SETB C           ;  CY = 1
ANL C, ACC. 0    ;  CY = '1' AND '1' = 1
ANL C, ACC. 1    ;  CY = 1 AND 0 = 0
```

7.4.8 ANL C,/bit Instruction

This instruction 'AND' the carry bit with complement of the bit specified in the instruction and places result in CY. The source bit can be specified using direct addressing mode. Only CY flag is affected.

Example :

```
MOV A, #01H      ;  A = 01 H, i.e. ACC.0 = 1, ACC.1 = 0
ANL C,/ACC.0     ;  CY = CY AND 0 = 0
SETB C           ;  CY = 1
ANL C,/ACC.1     ;  CY = 1 AND 1 = 1
```

7.4.9 ORL C, bit Instruction

ORL C, bit (OR logical for bit variable) instruction 'OR' the carry bit with the source bit specified in the instruction and places result in CY flag bit. The source bit

can be any bit addressible RAM/SFR location specified using direct addressing mode. Only CY flag is affected.

Example :

```
MOV A, #01H      ;  A = 01H i.e. ACC. 0 = 1, ACC. 1 = 0
CLR C            ;  CY = 0
ORL C, ACC. 1    ;  CY = 0 OR 0 = 0
ORL C, ACC. 0    ;  CY = 0 OR 1 = 1
```

7.4.10 ORL C, /bit Instruction

This instruction 'OR' the carry bit with the complement of the source bit specified in the instruction and places result in the CY bit. The source bit can be any bit addressable RAM/SFR location specified using direct addresssing mode. Only CY bit is affected.

Example :

```
MOV A, #01H      ;  A = 01H, i.e. ACC. 0 = 1, ACC. 1 = 0
CLR C            ;  CY = 0
ORL C, /ACC. 0   ;  CY = 0 OR 0 = 0
ORL C,/ACC. 1    ;  CY = 0 OR 1 = 1
```

7.4.11 MOV destination bit, source bit Instruction

This MOV instruction copies the source bit to the destination bit. One of the operand must be carry flag bit and the second operand can be any bit of the bit addressable RAM location/SFR specified using direct addressing mode.

```
MOV C, ACC. 5    ;  CY = ACC.5 (bit 5 of A)
MOV P0.3, C      ;  P0.3 (Bit 3 of P0) = CY
```

Instructions JC, JNC, JB, JNB, JBC are boolean instruction used for conditional program branching (relative) depending upon the value of bit under test. These instructions will be discussed along with the program branching instructions.

7.5 PROGRAM BRANCHING INSTRUCTIONS

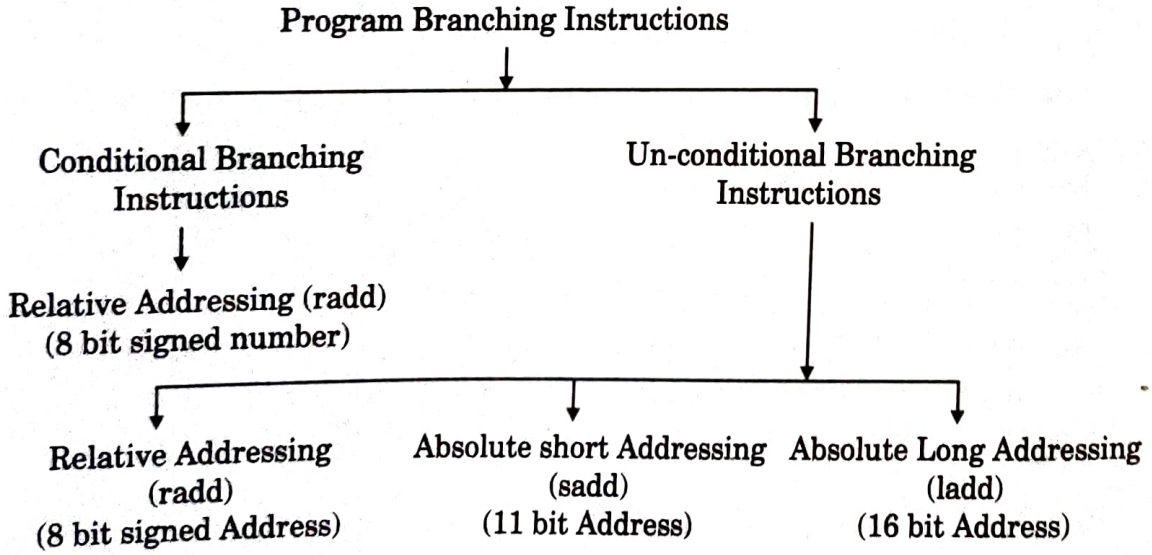
Normally the program instructions are executed sequentially *i.e.* after first instruction, second instruction is executed. But many times it is necessary to stop sequential execution and jump to a particular instruction depending upon some condition or programming need. For this purpose, program branching instructions are used. Both conditional and unconditional branching instructions are available in 8051. Jump or call instructions may have one of the three ranges :

- Relative range
- Absolute short range
- Long range

A **relative jump** transfers control within 256 byte range, that is from –128 to +127 bytes relative to the first byte of the following instruction (instruction after the jump instruction). Relative address is specified by 8-bit signed number (negative in 2's complement form). **All the conditional jump instructions are relative short jumps. (SJMP).**

Absolute short range jump (AJMP) transfers control within 2K byte page of program memory of the next instructions (after the jump instruction). Absolute jumps allows 11-bit address to the specified in the instruction. Complete 16 bit address of the new instruction is formed by taking five MSB bits (page number) of next instruction address and attaching the 11 bit absolute address specified in the instruction as LSB.

Absolute long range (or simply Long Range) jump (LJMP) transfers control to anywhere in the entire program memory i.e. from 0000H to FFFFH. 16 bit absolute address is specified in the instruction. Long range addresses required three byte instruction : one byte for opcode and two bytes for absolute address.



7.5.1 JC radd Instruction

JC (Jump if carry) instruction jumps to relative addressed location if carry flag is high otherwise next instruction will be executed. Relative address is 8-bit signed number which is added to PC, which is presently pointing to the instruction next to JC instruction, to get the address of the jumped location.

Example :

```
004E MOV A, #33H  ;  A = 33H
0050 JC 02 H      ;  Jump relative if carry = 1 i.e. PC = 0052 + 02H if
                     carry = 1 otherwise PC = 0052 H
0052 MOV A, #55H  ;  A = 55H
0054 MOV 90H, A   ;  P1 (add = 90 H) = 33H or 55H depending upon
                     whether jump takes place (CY = 1) or not.
```

7.5.2 JNC radd Instruction

JNC (Jump if no carry) instruction causes jump to relative addressed location if there is no carry (CY = 0). Relative address (radd) is 8 bit signed number. Address of jumped location is radd plus address of the instruction following the JNC instruction.

Example :

```
0050 ADD A, #01H  ;  A = A + 1
0052 JNC FCH      ;  If CY = 0, PC = 0054 – 4 (FC in 2's complement
                     form) = 0050 H otherwise PC = 0054 H
0054 NOP
```

7.5.3 JB bit, radd Instruction

JB (Jump if bit set) instruction causes jump to relative addressed location if the given bit is high. The bit can be any of the bit addressable RAM location/SFR and can be specified using direct addressing mode only.

```
JB P1.3, NEXT  ;  Jumps to relative addressed location with lable
                  NEXT if bit 3 of P1 is high.
```

7.5.4 JNB bit, radd Instruction

JNB (Jump if no bit *i.e.* bit not set) instruction causes jump to relative addressed location if the given bit is low (0). The bit can be any of the bit addressable RAM location/SFR and can be specified using direct addressing mode only.

```
JNB P2.5, NEXT  ;  Jumps to relative addressed location with label
                   NEXT if bit 5 of P2 is low.
```

Example

```
HERE : JNB P1.2, HERE  ;  Relative address by HERE will have value
                          FE (–2H)
```

The program will execute this instruction repeatidly as long as bit 2 of P1 is zero.

Note : In JB and JNB, if the bit specified is port bit, then port pin data is used.

7.5.5 JBC bit, radd Instruction

JBC (Jump if bit is set and clear bit) instruction causes jump to relative addressed location if the specified bit is high and at the same time the bit is cleared to zero.

```
JBC P1.3, NEXT   ;  Jumps to relative addressed location with label
                    NEXT if bit 3 of P1 is high and then P1.3 = 0
```

Note : In JBC, if the bit specified is port bit then port SFR latch bit data is used and altered.

7.5.6 JZ radd Instruction

JZ (Jump if zero) instruction causes jump to relative addressed (radd) location if the accumulator contents are zero.

```
JZ NEXT1         ;  Jumps to relative addressed location with label
                    NEXT1 if A = 0.
```

7.5.7 JNZ radd Instruction

JNZ (Jump if not zero) instruction causes jump to relative addressed (radd) location if the accumulator has a value other than zero.

```
JNZ NEXT 2       ;  Jumps to relative addressed location with label
                    NEXT2 if A ≠ 0.
```

7.5.8 CJNE destination, source, radd Instruction

CJNE (Compare and jump if not equal) instruction compares the source byte with destination byte and if they are not equal then it causes jump to relative addressed location (radd + address of next instruction). CY flag is affected. CY flag is set if contents of destination are less than contents of source, otherwise, CY is rest. Source byte can be direct addressed or immediate addressed. Destination byte can be A, R0 – R7, or register-indirect addressed location :

```
CJNE A, address, radd  ;  Compares A with direct addressed locations
                          contents and if not equal then jumps to relative
                          addressed location (PC = radd + address of Next
                          Instruction)
```

CJNE A, #n, radd ; Compares A with immediate data given in the instruction and if not equal then jumps to relative addressed location.

CJNE Rn, #n, radd ; Compares register Rn (R0 – R7) with immediate data and if not equal then jumps to relative addressed location.

CJNE @ Rp, #n, radd ; Compares internal RAM/SFR location, whose 8bit address is given by Rp (R0, R1), with immediate data and if not equal then jumps to relative addressed location.

Example :

CJNE A, 35H, NEXT ; Jumps to relative addressed location with label NEXT if A is not equal to RAM contents of location 35H.

CJNE @R1, #65H, NEW ; Jumps to relative addressed location with label NEW if the contents of memory location, whose address is in R1, is not 65H.

7.5.9 DJNZ destination, radd Instruction

DJNZ (Decrement and jump if not zero) instruction decrement the destination byte by one, and if result is not zero, it will jump to the relative addressed location. Destination byte can be register R0 – R7 or direct addressed internal RAM location or SFR. In case of ports, port SFR is decremented and tested.

DJNZ Rn, radd : Decrement register Rn (R0 – R7) by one and jumps to relative addressed location if result is not zero.

DJNZ address, radd : Decrement the contents of direct addressed RAM location/SFR and jumps to relative addressed location if result is not zero.

Example : Count from 1 to 50 and sends the count to P0.

```
CLR A          ;  A = 00H
MOV R1, #32H   ;  R1 = 50 (32H)
REP : INC A    ;  A = A + 1
MOV P0, A      ;  P0 = A
DJNZ R1, REP   ;  R1 = R1 – 1, Repeat if R1 ≠0.
```

7.5.10 JMP @ A + DPTR Instruction

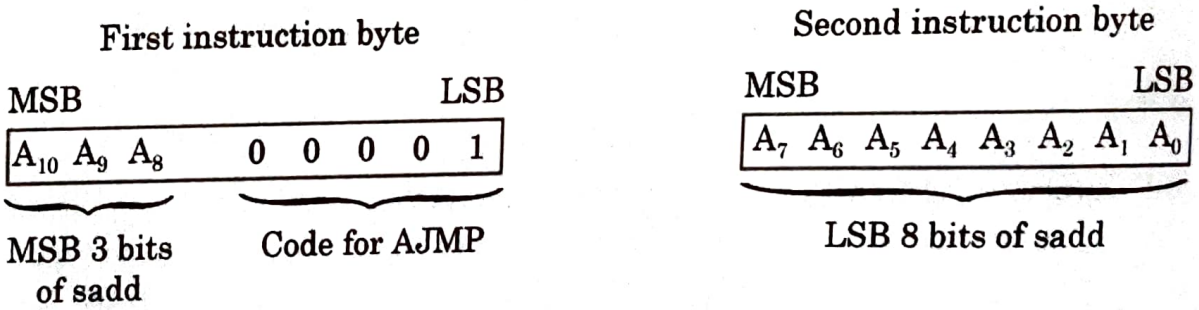
JMP (Jump) instruction is an unconditional branch instruction. Whenever this instruction is executed, it transfers control to program memory location whose address is sum of register A and register DPTR.

JMP @A + DPTR ; PC = A + DPTR

7.5.11 AJMP sadd Instruction

AJMP (Absolute Jump) instruction is an unconditional jump instruction. Whenever this instruction is executed, it transfers control to absolute short range location specified by sadd. Absolute short address (sadd) is of 11 bit. This instruction is 2 byte long. First opcode byte's bits D7, D6 and D5 contains MSB 3 bits of sadd. Rest of eight LSB of sadd are specified by second byte of the instruction. Address of the jumped location (where control is to be transfered) is formed by taking MSB five bits from the address of the next instruction and attaching the 11 bit absolute short address with it.

Opcode formation for AJMP :



Example :

Memory address	Opcode	Instruction
4030	A1	AJMP 583H
4031	83	
4032	EB	MOV A, R3

When instruction at program location 4030, 4031 i.e. AJMP 583 H is executed, control is transferred to program mem. location 4583 H instead of executing next instruction at location 4032 H i.e. MOV A, R3. Address 4583H is formed by taking MSB five bits (01000) from address 4032H of next instruction and attaching 583H 11-bit sadd to it.

7.5.12 LJMP Ladd Instruction

LJMP (Long Jump) is an unconditional jump instruction. Whenever this instruction is executed, the program control is transferred to program memory location whose 16 bit address (ladd) is given in the instruction. It is a three byte instruction. The first byte is opcode and next two bytes are the destination address (ladd). LJMP instruction covers the whole 64K byte program memory location.

```
LJMP 236AH      ;  Program jumps to memory location 236A H i.e. PC =
                   236AH.
```

7.5.13 SJMP radd Instruction

SJMP (short jump) is an unconditional jump instruction. Whenever this instruction is executed, the program control is transferred to relatively addressed program memory location. Relative address (radd) is 8-bit signed number (negative number in 2's complement form). This is a two-byte instruction. First byte is the opcode for SJMP and second byte is radd. Relative address (radd) is added to PC (Program Counter), pointing the instruction next to SJMP instruction, to get the address of destination location. In this instruction, the destination address (where program control jumps) must be within –128 to +127 bytes from the instruction next to SJMP instruction.

Example Problem 2.12 : The program ON/OFF the LED connected at port 1-3 pin and repeats the execution infinitely.

Solution : The program can be written as :

LOC	OBJ	LABEL	Mnemonics	
1020	D2, 93	AGAIN :	SETB P1.3	
1022	78,FF		MOV R0, #FFH	
1024	00	REP1 :	NOP	
1025	00		NOP	
1026	D8, FC		DJNZ R0, REP1	Delay loop1
1028	C2, 93		CLR P1.3	
102A	78, FF		MOV R0, #FFH	
102C	00	REP2 :	NOP	
102D	00		NOP	
102E	D8, FC		DJNZ R0, REP2	Delay loop 2
1030	80, EE		SJMP AGAIN	

First instruction sets bit P1.3. 2nd instruction copy immediate data FFH (according to delay time) into R0. Next two instruction are No operation instructions and are used to increase the delay time of delay loop. Next instruction decrements R0 by one and jump back to instruction at location 1024 with label 'REP1'. Here relative addressing is used and address for label REP1 is calculated by adding 1028 (Address of Next instruction) + FC (8 bit signed number) using 2's complement addition (FFFC for FC) = 1024 H. Next five instructions do the some by clearing bit P1.3. Last instruction jumps back to first instruction. Here adding 1032 (Address of Next Instruction) + (FF) EE gives 1020 H.

7.5.14 ACALL sadd (11 bit) Instruction

ACALL (Absolute Call) instruction is an unconditional call instruction to call a subroutine. It works just like AJMP instruction but with the difference that this instruction pushes the address (two bytes) of the next instruction (Insturction written after ACALL instruction) onto the stack which is used to come back. The subroutine should be within 2K bytes from the current program counter position. This instruction is 2 bytes long. First opcode byte's bits D7, D6 and D5 contains MSB 3 bits of sadd and rest of eight LSB bits are specified by second byte of the instruction. Address of the first instruction of the subroutine is formed by taking MSB five bits from the address of the next instruction after ACALL and attaching 11 bit absoulute short address (sadd) with it.

7.5.15 LCALL ladd (16 bit) Instruction

LCALL (Long CALL) Instruction is an unconditional call instructioin to call a subroutine anywhere in the 64K byte program memory. It is like LJMP instruction but with the difference that this instruction pushes the 16 bit address of the next instruction onto the stack which is used at the end of the subroutine to go back to the calling program with the help of RET instruction. It is a three byte instruction. The first byte is opcode and next two bytes are the destination address (ladd).

7.5.16 RET Instruction

RET (Return from subroutine) instruction is used to return from end of a subroutine to the calling program which had earlier called the subroutine using LCALL or ACALL instructions. The top two bytes of the stack are popped into the program counter (PC) and program execution continues at this new address. After popping the top two bytes of the stack into the program counter, the stack pointer (SP) is decremented by 2. Figure 7.7 shows use of stack during subroutine call.

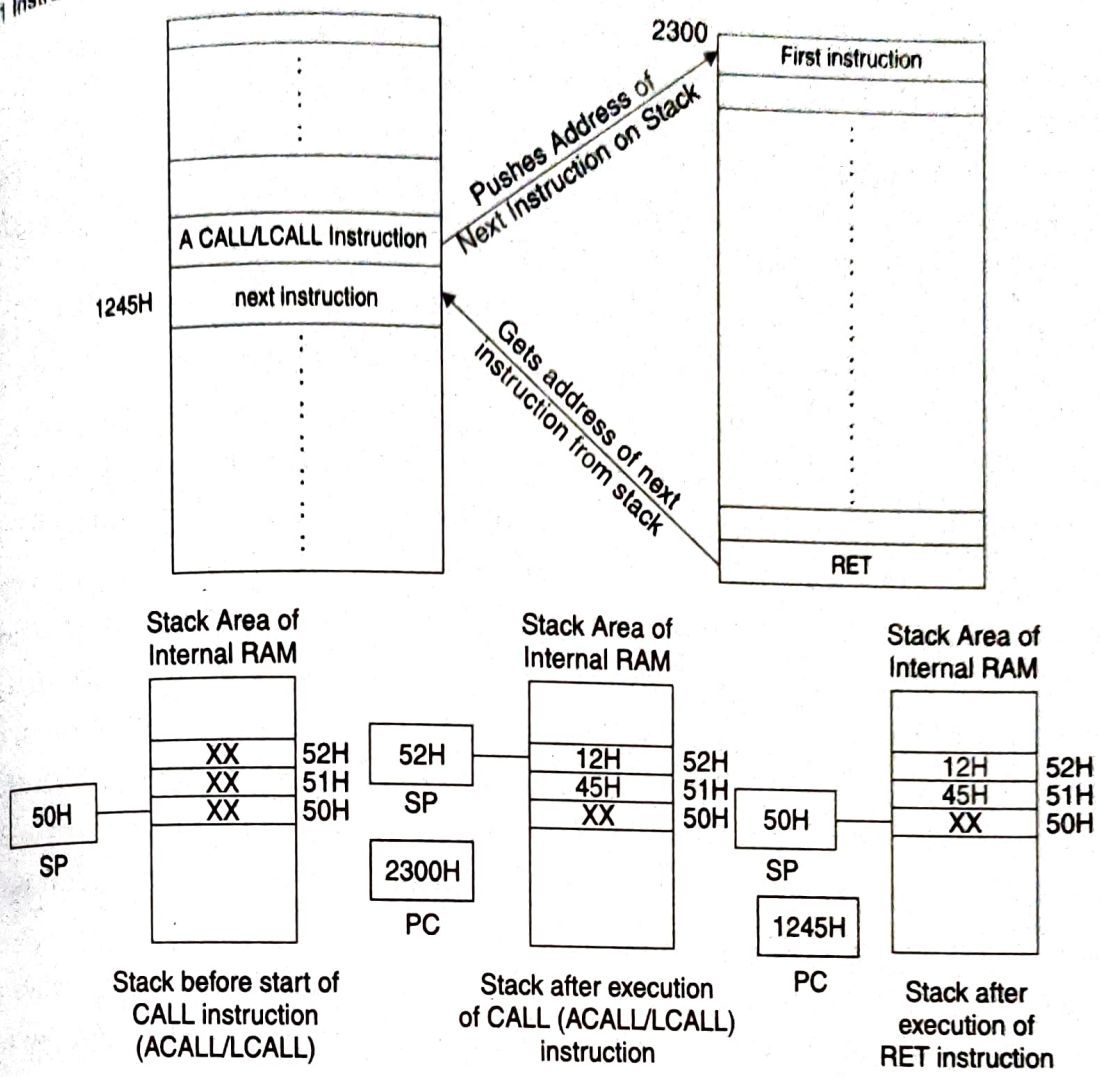


Fig. 7.7 : Stack and PC Contents during Subroutine Call

7.5.17 NOP Instruction

NOP (No operation) instruction performs no operation and execution continues with the next instruction. It is used for timing delays to waste clock cycles.

7.5.18 RETI Instruction

RETI (Return from Interrupt) is used at the end of the an interrupt service routine to go back to the main program. The top two bytes of the stack are popped into PC and program execution continues at this new address. After popping the top two bytes of the stack onto the PC, the SP is decremented by 2 and **resets the interrupt enable** flag. When an ISS (Interrupt Service Subroutine) execution starts, hardware interrupt disable flag is set to prevent another interrupt of same priority level to take place untill RETI is executed.

SUMMARY

8051 has one to three bytes long, 111 different types of instruction. These instructions can be divided into five categories : Data transfer, Arithmetic, Logical, Boolean and Program Branching Instruction.

Data transfer instructions are used to transfer (Copy) data from one location to another. MOVX is the only instruction to transfer data between external RAM and A. MOVC instruction is used to transfer data bytes from program memory to A. MOV, PUSH, POP, XCH and XCHD data transfer instructions work with only internal RAM and SFRs. Data transfer instructions do not affect flags unless PSW is directly specified. ADD, ADDC, SUBB, DIV, MUL, INC, DEC, DA A are arithmetic instruction. Accumulator A normally stores the result and math flags are affected to tell the result condition. ANL, ORL, XRL, RRA, RRC A, RL A, RLC A, SWAP, CLR and CPL are logical instructions. Arithmetic and logical instructions work on one operand or maximum two operans of eight bit each. CLR bit, CPL bit, SETB, ANL bit, ORL bit, MOV bit are instruction which work on single bit and are very useful for control design. These instructions work with only 16 bytes of bit addressable internal RAM and bit addressable SFRs.

There are different ways to specify operand of an instruction. These ways are called addressing modes. Large the addressing modes, more flexible and easy is to write a complicated program 8051 supports five addressing modes : Register, Immediate, Direct, Register-Indirect and indexed. In register addressing mode, registers A, DPTR and R0 – R7 of current register bank can be specified in the instruction holding the operand. In immediate addressing mode, operand is a part of the instruction. ln direct addressing mode, address of the location where operand lies is specified in the instruction. In register-indirect addressing mode, register DPTR, R0 or R1 is specified in the instruction which holds the address of the location where operand lies. In indexed addressing mode, contents of register A and DPTR/PC are added to form the address of program memory location where operand lies and useful for look-up tables. External data memory can be addressed using only register-indirect addressing mode.