Module 2

Classification of 8051 Instructions

Instruction set can be classified as

(a) Data transfer instructions (c) Logical instructions

(b) Arithmetic instructions (d) Bit manipulation instructions

(e) Banch control transfer instructions.

The instructions further can be classified as:

* Single byte instructions * Two byte instructions

* Three byte instructions.

Data transfer instructions:

These instructions are used to copy data from source operand to destinations operand. There are 28 distinct mnemonics to copy data from a source to a distination. Three main groups of data transfer are

1. MOV destination, source
2. PUSH SOURCE (or) POP DESTINATION
3. XCH destination, source.

The MOV opcodes involve data transfers within 8051 memory. This memory is divided into

(a) Internal RAM 00h - 7Fh

(b) Internal SFR's 80h - FFh

(c) External RAM Anywhere from 0000h - FFFFh

(d) Internal & External ROM;

(0000h - 0FFFh) { 1000h-FFFFh if $\overline{EA}$ = Vcc
0000h-FFFFh if $\overline{EA}$ = GND

Five types of Opcodes are used to move data:

1. MOV
2. MOVX
3. MOVC
4. PUSH & POP
5. XCH

* Data transfer instructions do not affect PSW

Internal Data MOV's:

Immediate Addressing:

MOV A,#n
MOV Rn,#n
MOV DPTR,#nn

Ex:
MOV A,#75h
MOV R2,#0F5h
MOV DPTR,#2ABAh

Register Addressing:

MOV A,Rn
MOV Rn,A

Ex: MOV A,R1
MOV R7,A

Direct Addressing:

MOV A,address
MOV address,A
MOV Rn,address
MOV address,Rn
MOV address,#n
MOV add1,add2

Indirect addressing:

MOV @RP,#n
MOV @RP,address
MOV @RP,A
MOV address,@RP
MOV A,@RP

MOV @R0,#55h
MOV @R1,30h
MOV @R0,A
MOV 40h,@R1
MOV A,@R1

External Data MOV's:

External data move take place using indirect addressing mode (or) indexed address mode

External RAM - 64KB
External ROM - 64KB

* R0 & R1 are limited to external RAM address range 00-FFH
* DPTR External RAM address space 0000H-0FFFH
* X refers to external.

MOVX A, @Rp

MOVX A, @DPTR

MOVX @Rp, A

MOVX @DPTR, A

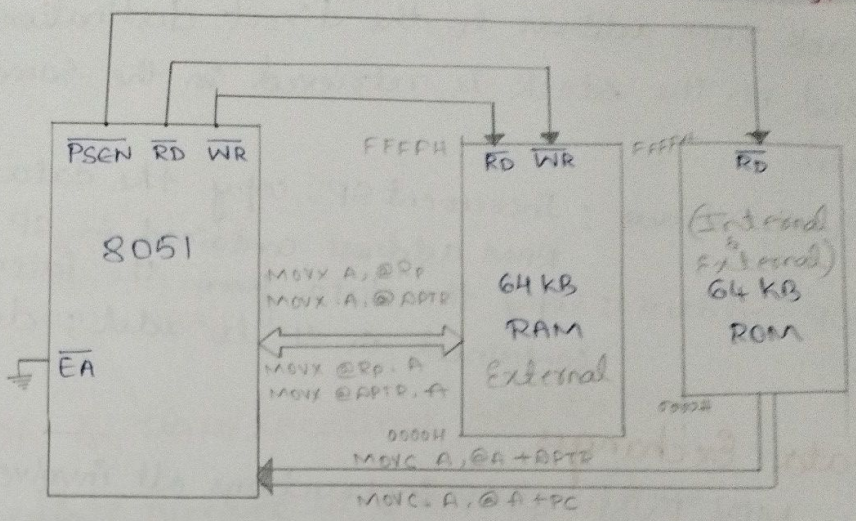


Fig. External Addressing Using MOVX & MOVC

Code memory Read Only Data Moves :

C → Code memory

MOVC A, @A+DPTR

MOVC A, @A+PC

MOV DPTR, #1000h

MOV A, #50h

MOVC A, @A+DPTR

MOVC A, @A+PC

[1050h] = 60h

[A] = 60h

[Pc] = 4000h

[1000h] = 0AFh

Push & Pop Operations :

The PUSH & POP opcodes specify the direct address of the data. The data moves blw an area of internal RAM (i.e., Stack) & the direct address.

PUSH: PUSH opcode copies data from the source address to the stack. SP is incremented by 1 before the data is copied to the internal RAM Location.

Data of the address specified will be pushed on to the stack Excessive PUSHing can make the stack exceed 7Fh (top of internal RAM), after which PUSHed data is lost.

POP: POP opcode copies data from stack to the destination address. SP is decremented by 1 after data is copied from the

Stack RAM address to the direct destination to ensure that data placed on the stack is retrieved in the same order as it was stored.

PUSH address ; Increment SP, copy the data in add to the internal RAM address contained in SP

POP address ; Copy the data from the internal RAM address contained in SP to add ; decrement the SP

Data Exchanges

MOV, PUSH & POP instructions all involve copying the data found in the source address to the destination address ; the original data in the source is not changed.

Exchange instructions actually move data in two directions:

- from source to destination
- from destination to source

All addressing modes except immediate may be used in XCH instructions.

XCH A, Rr ; Exchange data bytes blw current register Rr & A

XCH A, add ; Exchange data bytes blw add & A

XCH A, @Rp Exchange data bytes blw A & address in Rp

XCHD A, @Rp Exchange lower nibble blw A & address in Rp

Examples:

1) Load 40h & 50h any random numbers & exchange the contents of memory locations in 40h & 50h (i) Using direct address (ii) Using Exchange instructions (XCH) (iii) Using indirect addrss

(i)
```
Org 00h
MOV 40h, #0Ah
MOV 50h, #0Fh
MOV A, 40h
MOV 40h, 50h
MOV 50h, A
END
```

(ii)
```
Org 00h
MOV 40h, #0AAh
MOV 50h, #0Fh
MOV A, 40h
MOV R0, #50h
XCH A, @R0
XCH A, 40h
END
```

(iii)
```
Org 00h
MOV 40h,#0Ah
MOV 50h,#0Fh
MOV R0,#40h
MOV R1,#50h
MOV A,@R0
MOV @R0,50h
MOV @R1,A
END
```

A = 0AH
R0 = 0FH
50H = 0AH

② Put the number 8Dh in RAM locations 30h to 34h

Method 1 : Use the immediate number to a direct address

```
MOV 30h,#8Dh ; Copy the number 8Dh to RAM address 30h
MOV 31h,#8Dh
MOV 32h,#8Dh
MOV 33h,#8Dh
MOV 34h,#8Dh
```

Method 2: Using the immediate number in each instruction uses bytes
Use a register to hold number

```
MOV A,#8Dh  ; Copy the number 8Dh to the A register
MOV 30h,A   ; Copy the contents of A to RAM location 30h
MOV 31h,A   ;
MOV 32h,A
MOV 33h,A
MOV 34h,A
```

Method 3: There must be a way to avoid naming each address; PUSH opcode can increment to each address

```
MOV 30h,#8Dh ; Copy the number 8Dh to RAM address 30h
MOV 81h,#30h ; Set the SP to 30h
PUSH 30h     ; Push the contents of 30h(=8Dh) to address 31h
PUSH 30h     ; Continue pushing to address 34h
PUSH 30h
PUSH 30h
```

③ Swap the contents of registers R7 & R6 in register bank 0

Methode 1: Use a series of register address mode MOVes:

```
MOV A,R6   ; Copy the contents of R6 to A
MOV R5,A   ; Save contents of R6 in R5
MOV A,R7   ; Copy contents of R7 to A
```

```
MOV R6, A    ; Contents of R7 now in R6
MOV A, R5    ; Retrieve contents of R6
MOV R7, A    ; Contents of R6 now in R7
```

Method 2: Use a series of direct address mode MOV's

```
MOV 10h, 06h   ; Copy the contents of R6 to RAM address 10h
MOV 06h, 07h   ; Copy the contents of R7 to R6
MOV 07h, 10h   ; Copy the saved contents of R6 to R7
```

Method 3: Use a series of PUSH & POP's:

```
PUSH 07h   ; Push contents of R7 on the stack
PUSH 06h   ; Push contents of R6 on the stack
POP  07h   ; POP contents of R6 to R7
POP  06h   ; POP contents of R7 to R6
```

Method 4: Use a series of XCH

```
XCH A, R6   ; Exchange contents of A & R6
XCH A, R7   ; Contents of R6 now in R7, R7 in A
XCH A, R6   ; Contents of R7 now in R6, A same.
```

(4) State the addressing mode & meaning of the following instructions with an example

(i) MOV A, #30H

(ii) MOV @R1, 40h

(iii) MOV X @DPTR, A

(iv) MOVX A, @DPTR

(v) MOV 36H, 47H

(5) What would be the contents of accumulator after executing the following instruction

(i)
```
MOV 35h, #45H
MOV 35, #47h
MOV A, 35
END
```

(ii)
```
MOV 50H, #02H
MOV A, #04H
MOV R1, #50h
XCH A, @R1
```

(iii)
```
MOV A, #56H
MOV R1, #50H
MOV 50H, #45h
XCHD A, @R1
```

⑥ Write the content of each registers, memory location, after execution of each instructions, Take contents of SP as default value.

```
MOV A, #00h
MOV R1, #0ABh
MOV R2, A
MOV A, R1
MOV R0, #0FFh
~~MOV~~ PUSH E0h
PUSH 01H
XCH A, R0
.
.
POP 02H
POP 01H
END
```

Logical Instructions

* Byte level logical operations
* Bit level logical operations

(a) Byte level logical operations

The byte-level logical operations use all four addressing modes for the source of a data byte. The A register (or) a direct address in internal RAM is the destinations of the logical operation result

The operations done using each individual bit of destination & source bytes, called byte-level Boolean operations because the entire byte is affected are as follows:

ANL A, #n	ORL A, #n	XRL A, #n	CLR A
ANL A, add	ORL A, add	XRL A, add	CPL A
ANL A, Rr	ORL A, Rr	XRL A, Rr	
ANL A, @Rp	ORL A, @Rp	XRL A, @Rp	
ANL add, A	ORL add, A	XRL add, A	
ANL add, #n	ORL add, #n	XRL add, #n	

Ex1: Write the contents of each register & RAM location of the following program

MOV A, #0FFh	A = FFh
MOV R0, #77h	R0 = 77h
ANL A, R0	A = 77h
MOV 15h, A	15h = 77h
CPL A	A = 88h
ORL 15h, #88h	15h = FFh
XRL A, 15h	A = 77h
XRL A, R0	A = 00h
ANL A, 15h	A = 00h
ORL A, R0	A = 77h
CLR A	A = 00h
XRL 15h, A	15h = FFh
XRL A, R0	A = 77h

*Note: The instructions that can use SFR port latches as destinations are ANL, ORL & XRL

Only internal RAM (&) SFR's may be logically manipulated.

(b) Bit level logical Operations

One of the most important feature of 8051 is the ability to access the registers, RAM & I/O ports in bits instead of bytes Among 128 bytes internal RAM of 8051, only 16 bytes are bit addressable, the bit addressable RAM locations are 20H to 2FH, these 16 bytes provide 128 bits (16x8) of RAM bit addressability. They are addressed as 00H-7FH.

(c) SFR Bit Addresses

All SFRs may be addressed at the byte level by using the direct address assigned to it, but not all of the SFRs are addressable at the bit level. The bit addressable SFR & the corresponding bit addresses are given in Table 1.

Table 1:

SFR	Direct Address (hex)	Bit addresses (hex)
A	0E0	0E0-0E7
B	0F0	0F0-0F7
IE	0A8	0A8-0AF
IP	0B8	0B8-0BF
P0	80	80-87
P1	90	90-97
P2	0A0	0A0-0A7
P3	0B0	0B0-0B7
PSW	0D0	0D0-0D7
TCON	88	88-8F
SCON	98	98-9F

* I/O port [P0, P1, P2, P3] bit address can access either the entire 8 bits or any single bit without altering the other.
PX.Y (X → Port number & Y → bit number)

(d) Bit Level Boolean Operations

The bit level Boolean logical opcodes operate on any addressable RAM or SFR bit. The CARRY flag (C) in PSW SFR is the destination for most of opcodes because flag can be tested. The boolean bit level operations are as follows

ANL C,b	AND C & addressed bit, put result in C
ANL C,/b	AND C & the complement of addressed bit; put the result in C
ORL C,b	
ORL C,/b	
CPL C	Complement the C flag
CPL b	Complement the addressed bit
CLR C	Clear the C flag to 0
CLR b	Clear the addressed bit to 0
MOV C,b	Copy the addressed bit to C flag
MOV b,C	Copy the C flag to the addressed bit
SETB C	Set the C flag to 1
SETB b	Set the addressed bit to 1

Note: The bit instructions that can use a SFR latch bit are CLR, CPL, MOV & SETB

(e) Rotate and Swap Operations

The ability to rotate data is useful for inspecting bits of a byte without using individual bit opcodes. The reg. A can be rotated one bit position to the left (or) right with (or) without including C flag in rotation

If the C flag is not included, then rotation involves eight bits of reg. A & if C flag is included, then nine bits are involved in the rotation.

SWAP instruction can be thought of as a rotation of nibbles in reg. A

```
RL A    ; Rotate the A reg. one bit position to left
RLC A   ; Rotate reg. A & carry flag, as a 9th bit, one bit to left
RR A    ; Rotate A reg one bit position to right
RRC A   ; Rotate A reg & carry flag as 9th bit, one bit to right
SWAP A  ; Interchange the nibbles of reg. A ; put high nibble in low
          nibble position & low nibble in high nibble position
```

Ex: ① Write the contents of bits in the following program

SETB 00h	Bit 0 of RAM byte 20h = 1
MOV C,00h	C = 1
MOV 7Fh, C	Bit 7 of RAM byte 2Fh = 1
ANL C,/00h	C = 0 ; bit 0 of RAM byte 20h = 1
ORL C,00h	C = 1
CPL 7Fh	Bit 7 of RAM byte 2Fh = 0
CLR C	C = 0
ORL C,/7Fh	C = 1 , bit 7 of RAM byte 2Fh = 0
END	

Ex: ②

MOV A, #0A5h	A = 10100101b = A5h
RR A	A = 11010010b = D2h
RR A	A = 01101001b = 69h
RR A	A = 10110100b = B4h
RR A	A = 01011010b = 5Ah
SWAP A	A = 10100101b = A5h

```
CLR C        C=0; A=10100101b = A5h
RRC A        C=1; A=01010010b = 52h
RRC A        C=0; A=10101001b = A9h
RL A         A = 01010011b = 53h
RL A         A = 10100110b = A6h
SWAP A       C=0; A = 01101010b = 6Ah
RLC A        C=0; A = 11010100b = D4h
RLC A        C=1; A = 10101000b = A8h
SWAP A       C=1; A = 10001010b = 8Ah
```

Ex.③ Write instructions to use the registers of Bank 3 & load the same value 05H in the registers R0 to R3

```
ORG 00h
SETB PSW.4        PS1 = 1
SETB PSW.3        PS0 = 1   } Bank 3
MOV R0, #05H
MOV R1, #05H
MOV R2, #05H
MOV R3, #05H
END
```

④ Set Port 0 - bits 1, 3, 5, 7 to 1 & Rst the rest to 0

```
MOV 80h, #00h
MOV A, 80h
SETB 81h
SETB 83h
SETB 85h
SETB 87h
END
```

Bit address of P0

```
80h  P0.0
81h  P0.1
82h  P0.2
83h  P0.3
84h  P0.4
85h  P0.5
86h  P0.6
87h  P0.7
```

⑤ Clear bit 3 of RAM location 22h without affecting any other bit

```
MOV 22H, #88h
CLR C
MOV 13H, C
END
```

⑥ Invert the data on the port 0 pins & write the data to port 1

```
MOV 50h, #55h
MOV A, 50h
CPL A
MOV 90h, A
END
```

⑦ Swap the nibbles of R0 & R1 so that low nibble of R0 swaps with high nibble of R1 & high nibble of R0 swaps with the low nibble of R1

```
MOV R1, #0ABh
MOV R0, #69h
MOV A, R0          A = 69h
SWAP A             A = 96h
MOV R2, A          R2 = 96h
MOV A, R1          A = 0ABh
SWAP A             A = BAh
MOV R0, A          R0 = BAh
MOV R1, R2         R1 = 96h
END
```

⑧ Complement the lower nibble of RAM location 50h

```
MOV 2Ah, #55h
CPL 50h
CPL 51h
CPL 52h
CPL 53h
END
```

⑨ Make low nibble of R5 the complement of high nibble of R6

```
MOV R6, #0ABh
MOV R5, #76h
MOV A, R6
SWAP A
CPL A
ANL A, #0F0h
```

```
ANL  R5, #0F0h
ORL  A, R5
END
```

(10) Make the high nibble of R5 the complement of the low nibble of R6

[R6] = ABh
[R5] = 76h

```
MOV R6, #0ABh
MOV R5, #76h
MOV A, R6
CPL A
SWAP A
ANL A, #0F0h
ANL R5, #0F0h
ORL A, R5
END
```

(11) Move bit 6 of R0 to bit 3 of port 3

[R0] = 7Ah P3 → 0B0

D1 B6 B5 B4 B3 B2 B1 B0
07 ... 00
8 ch

0111 1010
7A

```
MOV R0, #7Ah
MOV 20h, R0
MOV C, 06h
MOV 0B3h, C
END
```

(12) Move bit 4 of RAM location 30h to bit 2 of A

A = 0E0

```
MOV 30h, #77h
MOV 20h, 30h
MOV C, 04h
MOV 0E2h, C
END
```

(13) XOR a number with whatever is in A, so that the result is FFh

```
MOV A, #77h
MOV R1, A
CPL A
XRL A, R1
END
```

(14) Store the most significant nibble of A in both nibbles of register R5. For example if A = B3h, then R5 = BBh

```
MOV A,#0B6h
MOV R5,A
ANL A,#0F0h
SWAP A
ANL R5h,#0F0h
ORL A,R5
MOV R5,A
END
```

(15) Store the least significant nibble of A in both nibbles of RAM address 3Ch for example if A = 36h, then 3C = 66h

```
MOV A,#36h
MOV 3Ch,A
ANL A,#0F0h
SWAP A
ANL 3Ch,#0F0h
ORL 3Ch,A
END
```

(16) Set the carry flag to 1 if the number in A is even

(17) Set the carry flag to 0 if the number in A is odd

```
MOV A,#44h
MOV R0,A
RRC A
CPL C
MOV A,R0
END
```

```
MOV A,#45h
MOV R0,A
RRC A
CPL C
MOV A,R0
END
```

(18) Treat registers R0 & R1 as 16-bit registers & rotate them one place to the left; bit 7 of R0 becomes bit 0 of R1, bit 7 of R1 becomes bit 0 of R0, & so on

```
MOV R0,#0F0h
MOV R1,#0F0h
CLR C
MOV A,R0
```

```
RLC   A
MOV   R0, A
MOV   A, R1
RLC   A
MOV   R1, A
MOV   A, R0
MOV   0E0, C
END
```

Arithmetic Operations

Arithmetic operations those performed by MC are addition, subtraction, increment, decrement, multiplication, division & decimal adjust accumulation

ADD group of instruction

```
ADD   A, #n
ADD   A, Rn
ADD   A, addr
ADD   A, @Rp
```

```
ADDC  A, #n
ADDC  A, Rn
ADDC  A, addr
ADDC  A, @Rp
```

* CY, AC & OV flags are affected

Subtraction

```
SUBB  A, #n
SUBB  A, Rn
SUBB  A, addr
SUBB  A, @Rp
```

* CY, AC & OV flags are affected

Incrementing

```
INC   A
INC   Rn
INC   addr
INC   @Rp
INC   DPTR
```

Decrementing

```
DEC   A
DEC   Rn
DEC   addr
DEC   @Rp
```

* MUL AB ; Multiply contents of reg. A & B
DIV AB ; Divide contents of reg. A by contents of reg. B
DA A ; Decimal adjust the A register

→ MUL AB (multiplier, multiplicand) ⇒ Result is stored in AB
Low byte in A & High byte in B.

→ DIV AB (dividend, divisor) → Result is stored in AB
Quotient in A & remainder in B

* The C, AC & OV flags are arithmetic flags. They are set to 1 (or) cleared to 0 automatically, depending on the outcomes of the instructions shown in Table 1

Instruction Mnemonic	Flags affected
ADD	C AC OV
ADDC	C AC OV
ANL C,direct	C
CJNE	C
CLR C	C = 0
CPL C	C = $\bar{C}$
DA A	C
DIV	C = 0 OV
MOV C,direct	C
MUL	C = 0 OV
ORL C,direct	C
RLC	C
RRC	C
SETB C	C = 1
SUBB	C AC OV

Unsigned & Signed Addition:

* Signed numbers use bit 7 as a sign bit in the most significant byte (MSB) of group of bytes chosen to represent largest number to be needed by program.
* Signed numbers use a 1 in bit position 7 of the MSB as a negative sign & 0 as a +ve sign. Further, all negative numbers are not in true form, but are in two's complement form.

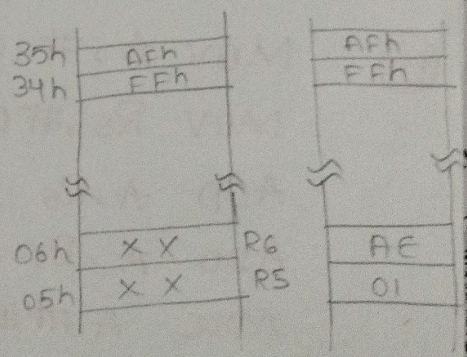
* Adding (or) subtracting unsigned numbers may generate a Carry flag when sum exceeds FFh (or) a Borrow Flag when the minuend is less than the subtrahend.
* OV flag is not used for unsigned addition & subtraction.

① Add the unsigned numbers found in internal RAM locations 25h, 26h & 27h together & put the result in RAM locations 31h (MSB) & 30h (LSB)

```
MOV  31h, #00h   ; Clear the MSB of result to 0
MOV  A, 25h      ; Get the first byte to be added from location 25h
ADD  A, 26h      ; Add 2nd byte found in RAM location 26h
MOV  R0, A       ; Save the sum of first two bytes in R0
MOV  A, #00h     ; Clear A to 00
ADDC A, 31h      ; Add carry to MSB; carry=0 after this operation
MOV  31h, A      ; Store MSB
MOV  A, R0       ; Get partial sum back
ADD  A, 27h      ; Form final LSM Sum
MOV  30h, A      ; Store LSB
MOV  A, #00h     ; Clear A for MSB addition
ADDC A, 31h      ; Form final MSB
MOV  31h, A      ; Store final MSB
```

② Add the bytes in RAM locations 34h & 35h, put the result in registers R5 (LSB) & R6 (MSB)

```
MOV R5, 34h
MOV A, 35h
ADD A, R5
MOV R5, A   ; Save LSB of result
CLR A
MOV R6, A   ; Clear R6
ADDC A, R6
END
```



③ Add the number 84h to the contents of RAM locations 17h (LSB) & 18h (MSM

```
MOV 17h, #0ABh
MOV 18h, #06h
MOV A, #84h
```

```
ADD   A, 17h
MOV   R1, A    ; Low byte of result
CLR   A
ADDC  A, 18h
MOV   R2, A    ; High byte of result
END
```

④ write the contents of registers & RAM location

```
MOV   A, #3Ah      ;  A = 3Ah
DEC   A            ;  A = 39h
MOV   R0, #15h     ;  R0 = 15h
MOV   15h, #12h    ;  [15h] = 12h
INC   @R0          ;  [15h] = 13h
DEC   15h          ;  [15h] = 12h
INC   R0           ;  R0 = 16h
MOV   16h, A       ;  [16h] = 39h
INC   @R0          ;  [16h] = 3Ah
MOV   DPTR, #12FFh ;  DPTR = 12FFh
INC   DPTR         ;  DPTR = 1300h
DEC   83h          ;  DPTR = 1200h  (SFR 83h is DPH byte)
```

⑤

```
MOV   A, #1Ch      ;  A = 1Ch
MOV   R5, #0A1h    ;  R5 = A1h
ADD   A, R5        ;  A = BDh   C = 0, OV = 0
ADD   A, R5        ;  A = 5Eh   C = 1, OV = 1
ADDC  A, #10h      ;  A = 6Fh   C = 0, OV = 0
ADDC  A, #10h      ;  A = 7Fh   C = 0, OV = 0
```

⑥

```
MOV  0D0h,#00h      ; C = 0
MOV  A,#3Ah         ; A = 3Ah
MOV  45h,#13h       ; [45h] = 13h
SUBB A,45h          ; A = 27h   C = 0, OV = 0
SUBB A,45h          ; A = 14h   C = 0, OV = 0
SUBB A,#80h         ; A = 94h   C = 1, OV = 1
SUBB A,#22h         ; A = 71h   C = 0, OV = 0
SUBB A,#0FFh        ; A = 72h   C = 1, OV = 0
```

⑦

```
MOV  A,#7Bh         ; A = 7Bh
MOV  0F0h,#02h      ; B = 02h
MUL  AB             ; A = F6h & B = 00h ; OV = 0
MOV  B,#0FEh        ; B = FEh
MUL  AB             ; A = 14h & B = FAh ; OV = 1
END
```

⑧

```
MOV  A,#0FFh        ; A = FFh (255d)
MOV  0F0h,#2Ch      ; B = 2C (44d)
DIV  AB             ; A = 05h, B = 23h [255d = (5 X 44) + 35]
DIV  AB             ; A = 00h & B = 05h [05d = (0 X 35) + 5]
DIV  AB             ; A = 00h & B = 00h [00d = (0 X 5) + 0]
DIV  AB             ; A = ?? & B = ?? ; OV = 1
```

⑨

```
MOV  A,#42h         ; A = 42 BCD
ADD  A,#13h         ; A = 55h ; C = 0
DA   A              ; A = 55h  C = 0
ADD  A,#17h         A = 6Ch , C = 0
DA   A              A = 72 BCD , C = 0
ADDC A,#34h         A = A6h ; C = 0
DA   A              A = 06 BCD ; C = 1
ADDC A,#11h         A = 18 BCD , C = 0
DA   A              A = 18 BCD , C = 0
END
```

⑩ Convert hexadecimal to decimal

```
MOV  A, #9CH          ; Hexa
MOV  B, #64h
DIV  AB
MOV  30h, A           ; Digit 3 in 30h
MOV  A, B
MOV  B, #0Ah
DIV  AB
MOV  31h, A           ; Digit 2 in 30h
MOV  32h, B           ; Digit 1 in 32h
END
```

DA – Decimal Adjust

↳ It can operate only with Accumulator & to ADD & ADDC instructions

↳ After the execution of ADD & ADDC instructions,

a. If the lower nibble is greater than 9 (or) if AC=1, add 0110 to lower nibble

b. If the higher nibble is greater than 9 (or) if CY=1, add 0110 to higher nibble