Addressing modes of 8051

Addressing modes is the manner in which operands are given in the instruction.

1) Immediate addressing mode : the data is given in the instruction itself.

E.g, i) MOV A, #35H (data whatever comes after # should be a number)

ii) MOV DPTR, #2000H

i) MOV is equivalent to 'copy'
⇒ copying 35H to accumulator.

ii) 20H goes to DPH and 00H goes to DPL

whatever follows # should be a number,

MOV A, #FFH ~~ear~~ is not a valid instruction.

∴ to make it valid, MOV A, #0FFH (zero)

DPTR is a 16 bit register divided into 2 8 bit registers — DPL and DPH.

'H' in the data signifies that the data is in hexadecimal number system.

if MOV DPTR, #0A134H,

then A134H. (each char is 4 bit).
2 char = 8 bit.

∴ DPH : A1H
DPL : 34H.

if command is MOV A, #35, (NO H, ⇒ decimal value)

~~A will contain~~ A → | 0011 | 0101 |

when its 35H, A will contain binary eq of

but when its just 35, its first converted to hex which is 23H and then converted to binary.

2. Registes addressing mode

Data is given by a Registes in the instruction. The permitted registers are A, R7, ... R0 of each memory bank (no use of #).

dest ← source.

1) MOV A, R_0 ; A gets whats stored in R_0

2) MOV R_0, A ; R_0 gets " " in A.

3) MOV R_0, R_1 ; R_0 ~~gets~~ should get ~~whats~~ stored to R_1 (invalid) but this is not possible since both are bank registers.

Snippet

```
ORG 000H
MOV A, #0AH
MOV R0, #10H
MOV A, R0
MOV R1, A
END
```

ORG 000H and END } assembler directives.

Assembler decodes the instruc and then sends it to the CP and assembler directive helps the assembler to decode the statement

A ← 0AH
R_0 ← 10H
A ← R_0 ← 10H
A ← 10H (overwrit)
R_1 ← 10H.

ORG : origin
000H : implies that the program starts from 000H.

3. Direct addressing mode

The address of the operand is given in the instruction. Only Internal RAM addresses (00H..7FH) and SFR addresses (from 80H to FFH) allowed.

Code.

```
MOV R2, #7FH   ;  R2 ← 7FH
MOV A, R2      ;  A ← 7FH
MOV R1, A      ;  R1 ← 7FH.
```

code

```
MOV A 02H ; A ← 02H
```
(02H ← memory location; 02H corresponds to R_2 of Bank 0.)

```
MOV A, 35H   ; A ← [35H]   A ← 20H
MOV A, 80H   ; A ← [80H]
MOV 20H, 30H ; 20H ← 30H
```

4. Indirect addressing mode: Here the address of the operand is given in a register. Internal RAM and External RAM can be accessed using this mode.

The advantage of giving an address using a register is that we can increment the address in a loop, by simply incrementing the register, and hence access a series of location.

→ MOV A, @ R_0

A gets the value pointed by R_0 ie 15H.
@: whatever is there in memory location 15H goes to A ie 20H

eg,

```
MOV R0, #15H ; R0 ← 15H
MOV 15H, #20H; 15H ← 20H
MOV A, @R0   ; A ← [R0]
               A ← @15H
```

```
MOV 0FH, #0AAH  ; [0FH] ← 0AAH
MOV R1 , #55H   ; R1 ← 55H
MOV R0, #0FH    ; R0 ← 0FH
MOV A, R0       ; A ← 0FH
MOV R7, A       ; R7 ← 0FH
MOV A, @R1      ; A ← [R1] ← 0FFH
MOV A, @R0      ; A ← [R0] ← 0AAH.
```

memory location.

- Internal RAM (8 bit address given by R_0 or R_1 only)
- External RAM (16 bit addresses given by DPTR)
- External RAM (8 bit " " " R_0 or R_1)

⇒ MOVX A,@R0 — (1)

X: tells the assembler that external RAM is to be used without presence of X, value is fetched from internal RAM

00 — FF : 8 bit values.

0000 — FFFF : 16 bit values.

Instruction (1) is limited to only 8 bit data. If 16 bit is to be used then DPTR is used as follows;

```
MOV DPTR , #1000H
MOVX A, @DPTR
```

MOV A @DPTR will show an error. b'cos A is 8 bit. and DPTR is 16bit

```
MOV R1 , #0AAH ; R1 ← 0AAH
MOVX A , @R1   ; A ← [AAH]
```

```
MOVX A , @R7   ; error. (only R0 & R1).
```

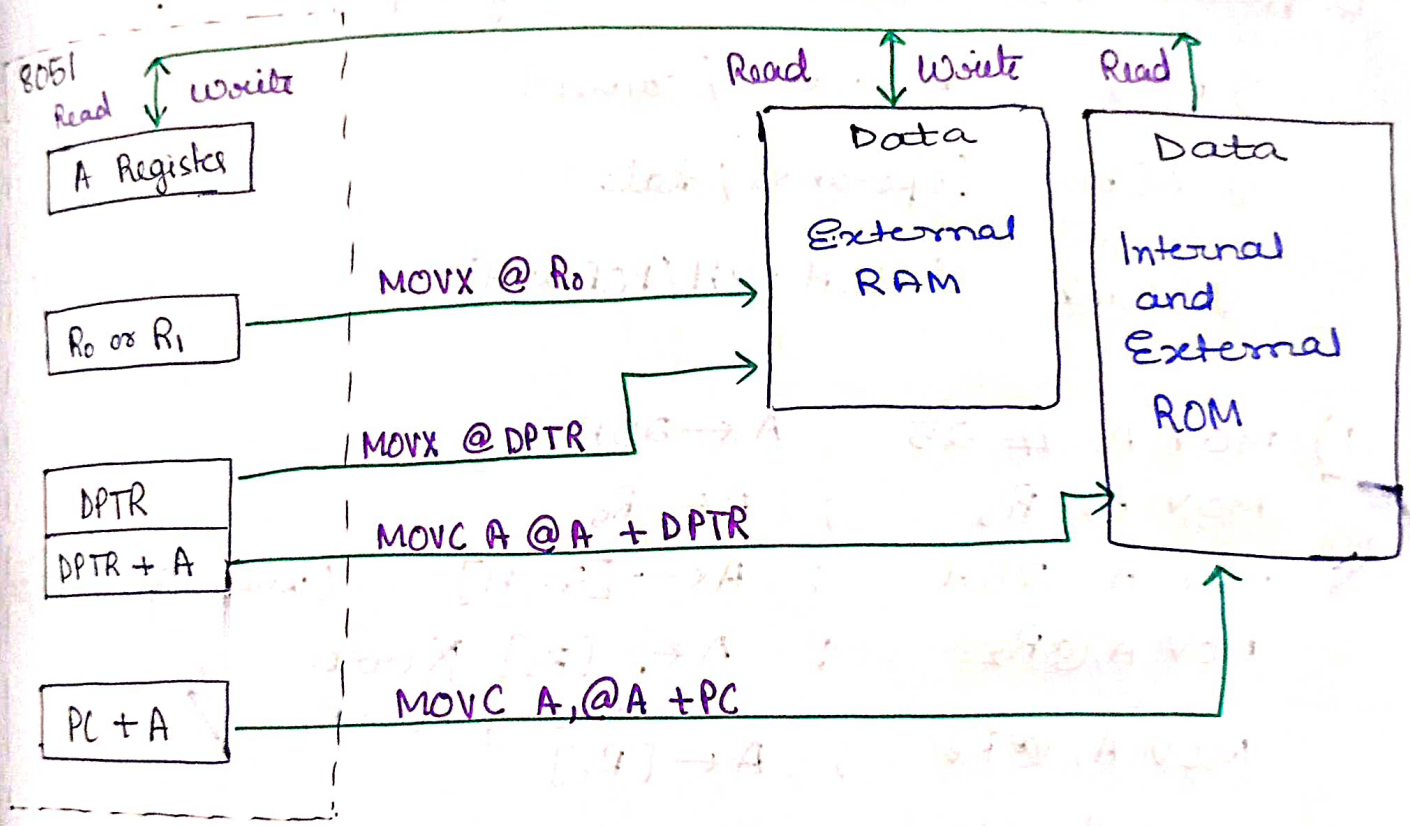
```
MOV DPTR, #2000H  ; DPTR ← 2000H
MOV A, #0AH       ; A ← 0AH
MOVX@ DPTR, A     ; [DPTR] ← 0AH
                  i.e, [2000H] ← 0AH
```

Internal or External ROM.

MOVC A, @ A+DPTR

MOVC A, @ A+PC

C: assembler knows that it has to transfer code from one memory to another



↑ External addressing using MOVC and MOVC

Data Transfer Instructions.

- Used to transfer or copy data from source to specified destination or exchange source or destination operands
- These instruction do not affect any flag except parity flag.
- Different instructions under this group are,
  1) MOV (destination/source)
  2) XCH (operands/data)
  3) Stack (PUSH xxH/POP xxH)

```
1) MOV A, #25    ; A ← 25H
   MOV A, R5     ; A ← R5
   MOV A, 25H    ; A ← [25H]      (value stored in 25H)
   MOV A,@R2     ; A ← [R2] X error.
                    ↳ only R0 or R1
   MOV A, @R0    ; A ← [R0]
   MOV R0, A     ; R0 ← A.
   MOV R5/#05H;  R5 ← 05H
   MOV R6, 05H   ; R6 ← [05H].
   MOV @R0, #20H ;[R0] ← 20H.
   MOV @ R0, @R1 ; [R0] ← [R1] error.
```

W.

```
MOVX  A, @R0          ; A ← [R0]
MOV DPTR, #5000H      ; DPTR ← 5000H
MOVX @R0,  A          ; [R0] ← A
MOVX @DPTR, A         ; [DPTR] ← A
```

② XCH → exchange.

1. XCH A, R_5 ; A ⟷ R_5 (data in A and R_5 are exchanged

Before execution (BE): A ← 25H R_5 ← FFH

After execution (AE): A ← FFH R_5 ← 25H

2. XCH A, 25H ; A ⟷ [25H]

3. XCH A, @R_0 ; A ⟷ [R_0]

4. XCHD A, @R_0 ; $A_{\text{lower nibble}}$ ⟷ $[R_0]_{\text{lower nibble}}$

D: digit (Enables gets exch) (always lower nibbles)

- BE: A ← 12H @R_0 → [R_0] ← 78H

  AE: A ← 18H @ [R_0] ← 72H.

③ PUSH / POP

PUSH 25H (stores temporarily without telling where to store.

By default, stack pointer (SP) has the value 07H in 8051. Can take value of 00 to FFH.

due to PUSH, a) SP gets incremented ie SP ← SP + 1

b) [SP] ← [25H]

for POP, sp gets decremented after data is retrieved.

```
MOV 25H, #0AH      ; [25H] ← 0AH
MOV 35H, #FFH        [35H] ← FFH
MOV 90H, #0BH        [90H] ← 0BH.
PUSH 35H ; SP ← 08H
           [08H] ← [35H] ← FFH
PUSH 90H ; SP ← 09H ; [09H] ← 0BH
PUSH 25H ; SP ← 0AH ; [0AH] ← 0AH
POP 15H  ; [15H] ← [SP] ← [0AH] ← 0AH.  2) SP ← 09H.
POP 30H  ; [30H] ← 0BH  and SP ← 08H
```

PUSH 15H ; sp←09H and [09H]←[15H]← 0AH

POP 35H ; [35H]←0AH and sp←08H

Arithmetic Instructions

(Addⁿ and subtraction operati... only on accumulator ie A

1) ADD (Addition).

* ADD A, #25H ; A ← A + 25H
  ↑ Dest ↑ Source

Eg. MOV A, #1...
ADD A, #05H
↳ A = 10H + 05H = 15H

① A = 10H
② A = 15H

a) ADD A, R_7 ; A ← A + R_7

3) ADD A, 25H ; A ← A + [25H]
(value stored in 25H location)

4) ADD A, @R_0 ; A ← A + [R_0]

If value of R_0 is 25H, then 3 and 4 is same.

---

5) ADDC A, #25H ; A ← 25H + A + Carry (1 bit)

(ADDC can be used everytime for a addition purpose whenever carry is generated due to addition)

⇒ whenever two digit nos are added, max ans is in 3 digits.

if, A ← FF
R_2 ← FF
ADD A, R_7 ; A ← A + R_7 ← 1FE

But 1FE is more than 8 bit ∴ A ← FE and 1 is carry which goes to the carry flag in the PSW register.

6) ADDC A, R_7 ; A ← A + R_7 + C_y

7) ADDC A, 25H ; A ← A + [25H] + C_y

8) ADDC A, @R ; A ← A + [R_8] + C_y

ADD R_1, R_2 ← not valid
ADD 20H, 30H ← " , ADD A, @R_7 X

⇒ After every ADD operation check carry flag.

SUBB.

9) SUBB A, #10H; A ← A - 10H - C_y

10) SUBB A, R_5 ; A ← A - R_5 - C_y

11) SUBB A, 10H ; A ← A - [10H] - C_y

12) SUBB. A, @R_0 ; A ← A - [R_0] - C_y

- for first subtraction operation, carry flag must always be zero. C_y=0.

- FF05
  - 1018

  2nd step. 1st step

---

13) INC A ; A ← A + 1 (Increment)

14) INC R_7 ; R_7 ← R_7 + 1 (R_0 to R_7 can be used)

15) INC 25H ; [25H] ← [25H] + 1 (Value gets plus 1)

16) INC @R_1 ; [R_1] ← [R_1] + 1

17) INC DPTR ; DPTR ← DPTR + 1

---

18) DEC A ; A ← A - 1

if A ← 00H then DEC A will A ← FF. (rolling back from 00 - FF)

19) DEC R_5 ; R_5 ← R_5 - 1

20) DEC 20H ; [20H] ← [20H] - 1

21) DEC @R_1 ; [@R_1] ← [@R_1] - 1

- DEC DPTR is not valid.

  however long method is possible.

  e.g if 1300H was to be decremented by 1,

  13 00
  ↓ ↓ -1
  12 FF

  it would be ⇒ 12FFH.

- DEC @R_7 is not valid. (Only R_0 & R_1).

---

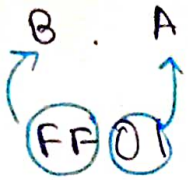
22) MUL AB (only multiplication variant)

- mult. of 2 8 bit nos can give a max of 16 bit no

  FF × FF = FE01

∴ The answer is stored in both A and B.
lower bit is stored in A
higher bit in B

The B register is especially reserved for mult and div operations.



23) DIV AB ; = A/B (always)
↳ load dividend to A
divisor to B.

A → stores Quotient
B → stores Remainder

- If B=0 then DIV AB, is not valid; but the compiler doesn't throw an error as its syntax is correct.

∴ if B=0, flag = 1 ⇒ illegal op

overflow flag that tells the prog if div is valid.

If B≠0
flag = 0.

⇒ Check overflow flag after every division.

24) DA A ; Decimal adjust Accumulator / Dec adjust after ADD.

* Used after ADD of 2 decimal numbers (BCD) (not hex)
* Does not convert hex to dec. DA just adjust the answer to get eq. decimal.

DA A ; A:

↓ works only after and ADD.

① if LN > 9 (ie, A B C ... F)
OR, AC = 1 (AC · aux carry)
(Beyond F)
⇒ 06H is added to A

② if ~~HN > 9~~ HN > 9, or Cy = 1
then ADD 60H to A

③ both 1 & 2 is true then ADD 66H

---

1) MOV A, #25H ; A ← 25H
ADD A, #25H ; A ← ~~50H~~ 4AH {expected is 50H ∴ use DA}
DA A ; A ← 50H

4	A

LN > 9 ∴
4A
06
50H

28H
28H
ADD 50H
DA 06H
56H → final ans.

Aux carry = 1

50H
50H
ADD → A0H
DA. + 60H
↳ 100H
↓ carry.

80H
80H
100H
160H

Logical and branching instructions

- Used to clear bits of any registes (And logic)

1. ANL A, #25H ; A ← A & 25H
2. ANL A, R_5 ; A ← A & R_5
3. ANL A, 30H ; A ← A & [30H]
4. ANL A, @R_0 ; A ← A & [R_0]
5. ANL 30H, A ; [30H] ← [30H] & A
6. ANL 30H, #00H ; [30H] ← [30H] & 00H

↓ and op^n with 00 helps clear certain bit

- Used to compliment bits of registes. (Xor logic)

XRL A, #55H
XRL A, R_6
XRL A, R_2
XRL A, @R_1
XRL 50H, A
XRL 50H, #0FFH ⟶ used to complement the bit.

Say, we want to complement only the LN of A.

MOV A, #95H
XRL A, #

- Used to set bits of any registes (Or log

ORL A, #10H ; A ← A or [illegible]
ORL A, R_3 ;
ORL A, R_5 ;
ORL A, @R_1 ;
ORL 30H, A ;
ORL 30H, #0FFH ;

↓

[30H] ← [30H] OR [illegible]

↓ will remain [illegible]

① RL A ; (Rotate left by 1 position)

② RR A ; (" right " ")

③ RLC A ; (Rotate left " " with carry)

---

① CPL A
↳ Compliment

A ← 55H, 01010101
A ← AAH 10101010

② CLR A
↳ clear.

A ← 00H

Insha Manowar
19090618 4.

Roll 16.

2. $y'' - 2x^2 y' + 2y = 0$

$y(0) + y'(0) = 5$

$y(1) = 0$

$h = 1/2.$

$y_0 + y_0' = 5$

$y_0 + \frac{y_1 - y_{-1}}{1} = 5$

$y_0 + y_1 - y_{-1} = 5$

$y_0 + y_0' = 5$

y_0 $\quad$ $y_1 = 0$ $\quad$ $y_2 = 2$

$x_0 = 0$ $\quad$ $x_1 = 0.5$ $\quad$ $x_2 = 1$

$$\left(\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \right) - 2x_i^2 y_i' + 2y_i = 0$$

4).

$$y_i'' - 2x_i^2 y_i' + 2y_i = 0$$

$$y_0 + y_0' = 5$$

$y_0 = ?$	$y_1 = ?$	$y_2 = 0$
$x_0 = 0$	$x_1 = 1/2$	$x_2 = 1$

$$\Rightarrow \left(\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}\right) + 2x_i^2 \left(\frac{y_{i+1} - y_{i-1}}{2h}\right) + 2y_i = 0 \quad \text{— (1)}$$

we check for $i = 0$ and 1.

$\Rightarrow$ when $i = 0$,

eq (1),

$$\frac{y_1 - 2y_0 + y_{-1}}{(0.5)^2} + 2x_0^2 \left(\frac{y_1 - y_{-1}}{(0.5}\right) + 2y_0 = 0$$

$$4y_1 - 8y_0 + 4y_{-1} + 2y_0 = 0$$

$$4y_1 - 6y_0 + 4y_{-1} = 0$$

$$y_0' = y_1 - y_{-1}$$

$$5 - y_0 = y_1 - y_{-1}$$

$$y_{-1} = y_1 + y_0 - 5$$

$$4y_1 - 6y_0 + 4y_1 + 4y_0 - 20 = 0$$

$$\Rightarrow 4y_1 - y_0 - 10 = 0 \quad \text{— (II)}$$

When $i = 1$,

$$\frac{y_2 - 2y_1 + y_0}{1/4} - 2x_1^2\left(\frac{y_2 - y_0}{l}\right) + 2y_1 = 0$$

$$-3y_1 + 4y_0 - y_0/2 + 2y_1 = 0$$

$$-6y_1 + 3.5y_0 = 0 \quad \text{(III)}$$

Solving (II) and (III),

$$y_1 = 4.375$$

$$y_0 = 7.5.$$

1. $y_{n+2} - 5y_{n+1} + 6y_n = 4^n(n^2 - n + 5) + 3^n$

auxiliary eqn: $(E^2 - 5E + 6)\, y_n = 0$

$$E = 3, 2$$

$$\therefore \quad y_{nc} = C_1(3)^n + C_2(2)^n$$

$$n^2 - n + 5 = [n]^2 + 5$$

$$y_{np} = \frac{4^n([n]^2 + 5)}{E^2 - 5E + 6} + \frac{3^n}{E^2 - 5E + 6}$$

$$= \frac{4^n([n]^2 + 5)}{\Delta^2 - 3\Delta + 2} + \frac{1}{(3^1 - 2)} \cdot \frac{3^{n-1}(n)^2}{2!}$$

$$= \frac{-1}{2}\left(1 - \frac{\Delta^2 - 3\Delta}{2}\right) \cdot 4^n([n]^n + 5) + 3^{n-2}\frac{[n]^3}{2!}$$

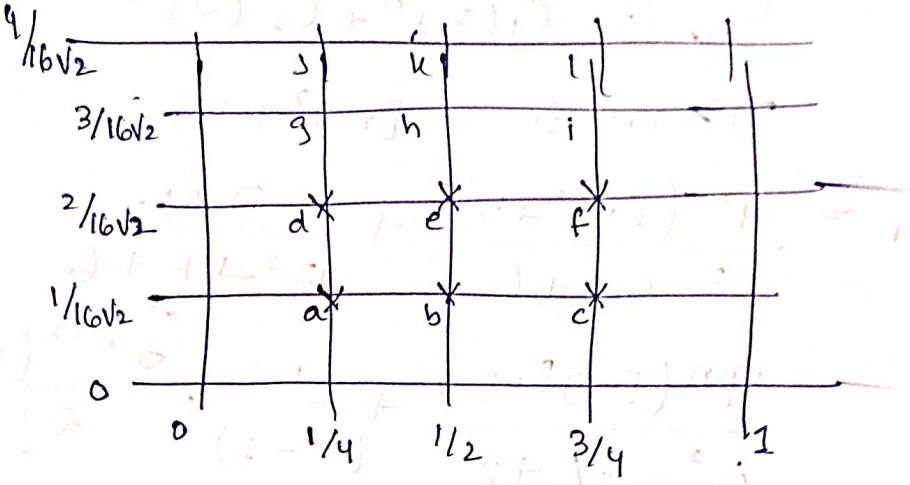
$$= \frac{-1}{2}\left[1 + \left(\frac{\Delta^2 - 3\Delta}{2}\right) + \left(\frac{\Delta^2 - 3\Delta}{2}\right)^2 \cdots\right) \{4^n[n]^2 + 5\} + 3^{n-2}\frac{[n]^3}{2!}$$

$$= \frac{-1}{2}\left[4^n[n]^2 + 5 \cdot 4^n + 4^n - \frac{3}{2} \cdot 2[n] \cdot 4^n + 3^{n-2}\frac{[n]^2}{2!}\right.$$

3) $\frac{\partial^2 u}{\partial t^2} = \frac{1}{32}\frac{\partial^2 u}{\partial x^2}$, $0<x<1$, $t>0$

$u(x,0) = 100\sin \pi x$, $\frac{\partial u(x,0)}{\partial t} = 1$

$u(0,t) = u(1,t) = 0$ $\quad h = 1/4$.



$k = \frac{h}{c} = \frac{1}{4\sqrt{32}} = \frac{1}{16\sqrt{2}}$

$u_{i,j+1} = u_{i+1,j} + u_{i-1,j} - u_{i,j-1}$

$u_{i,1} = \frac{f_{i+1} + f_{i-1}}{2} + k g_i$

$d = 37.454 + 0 - 100/\sqrt{2} = 150.044$

$e = 50.044 + 50.044 - 100$

$= 150.044$

$f = 0 + 37.454 - 4.11$

$= 33.344$

$g = 150.044 + 0 - 50.044$

$= 100$

$h = 1004.054$

$i = 100$ $\quad k = 49.956$

$j = 4.11$ $\quad L = 90.71$