Addressing Modes

* CPU can access data in various ways.
* the data could be in register or in memory or can be provided as a immediate value.
* Various ways of accessing data are called as addressing mode.

There are 5 addressing modes in 8051.

1. Immediate addressing mode.
2. Register addressing mode.
3. Direct addressing mode.
4. Register indirect addressing mode
5. Indexed addressing mode.

* Immediate addressing mode:-

* the Source operad is constant.
  the data is preceeded by '#' Symbol.

Mov des, Src → Src.

Eg Mov A, #FFh → immediate data
↑ ↑ ↳ Symbol
Instruction Dst

this instruction can be used to load information into any register, including DPTR register and 8051 ports.

Eg MOV R4, #0Ah
MOV B, #10h.
MOV DPTR, #1234h.

* Register addressing mode:-

- this involves the use of register to hold the data to be manipulated.

- the registers A, DPTR, R0 to R7 may be used as Both Source and destination.

Eg MOV A, R0
↑ ↑ ↳ Source
Instruction Destination

MOV R_2, A

ADD, A, R_5

MOV DPTR, #1234h.

MOV R7, DPH.

MOV R6, DPL.

* We can move data between accumulator and Register but movement of data blw R0 to R7 is not allowed.

Eg MOV R0, R1 → invalid.

* Direct addressing mode:-

- RAM locations 00-1Fh - assigned to register bank and stack.
- RAM location 20-2Fh are bit addressable space to save single bit data.
- RAM location 30-7Fh are available as a place to save byte size data.
- there by entire 128 byte of RAM can be accessed using direct addressing mode. 30h-7Fh are most often used.

- Here data is in RAM memory location whose address is known, & this is given as a part of instruction.

& MOV R0, 40h → address of RAM location where some data is present
↑ Register where data need to be sent.

MOV 56h, A ; Content of Accumulator will be moved to address 56h of RAM location.

* Register Indirect addressing mode:-
- Register is used to hold the address of data.
- this instruction uses opcode along with register R0/R1 which will hold the RAM address ranging from 00h to 7Fh.
- at @ symbol is used to indicate indirect addressing mode.
- MOV A, @R0 → Content present at the RAM add pointed by R0 will be moved to accumulator.
MOV @R1, B.

Limitation:-
* R0 and R1 are the only register that can be used for pointers in register indirect

* Indexed addressing mode:-
Widely used in accessing data elements in look up table located in program ROM space.

& MOVC A, @A+DPTR.

Add the content of accumulator with the content of DPTR to form the prog code memory location and then moves the content of external memory address to accumulator.

* only program memory can be accessed in this kind of addressing mode.

* Either DPTR or PC can be used.

* Instruction Sets:-

Based on the operation performed, the instruction set of 8051 are classified as.

1. Data transfer instruction.
2. Arithmetic instruction.
3. Logical instruction.
4. Boolean instruction.
5. program branching or machine control instruction.

Each instruction has two parts.
operation code & operands.

* Data transfer instruction:-

Instruction in this group are MOV, PUSH, POP, XCH.

MOV:- It copies data from one location to another location.

Syntax: MOV operand 1, operand 2

format: MOV, Destination, Source;

copy the content from Source to Destination.

1. MOV A, Rn. –

Byte/Size – 1 , Cycle – 1

Status flag affected: none.

operation: MOV

(A) ⟵ (Rn)

- moves the content of Rn to accumulator. data in Rn is not affected.
- Rn may be R0 to R7 register.

2. MOV A, direct (direct address)

Byte: 2 Cycle: 1.

operation: (A) ⟵ (direct address)

moves the content of address to the accumulator.

flags: none.

Eg MOV A, 40h.

A ← xx A ← FF

40h ← FF 40h ← FF

* Stack refers to an area of internal RAM used by the CPU to store & retrieve data quickly.

* register used to access is Stack pointer (SP)

*PUSH direct address:-

function: push the data onto stack.

Description: the Stack pointer is incremented by 1 the content of the indicated variable is the copied

into internal RAM location addressed by the stack pointer.

flag: None , Byte: 2 , cycle: 2.

operation: (SP) ← (SP)+1

* Supports only direct addressing mode.

& Push OEOh

(SP) ← (SP)+1 ; increment the SP

(SP) ← (OEOh Content) ; push the content present at EOh which is the RAM address of Accumulator to stack pointer.

2. push 03h → RAM address of R3 of Bank 0.

(SP) ← (SP+1)

(SP) ← (03h Content)

Post is RESET, SP is set to 07h.

Stack pointer is incremented before storing data so that the stack grows up as data is stored.

* POP direct address :-

function : Pop from stack.

Description : Copies the byte pointed to by Stack pointer (SP) to the location where direct address is indicated and decrements SP by 1.

flag : None. , Byte : 2 , Cycle : 2

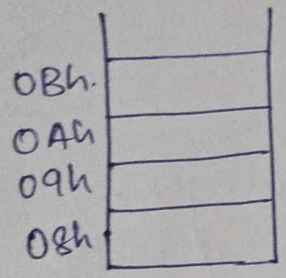
operation : (SP) ← (SP-1)

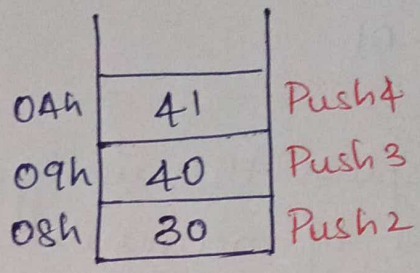
* Supports only direct addressing mode.
* pop A, pop Rn — illegal instruction.
* pop 0E0h, pop 03h.

Eg mov R2, #30h ; mov R3, #40h, mov R4, #41h
push2 ; push3 ; push4

initially Bank 0 is Selected SP = 07h

B.E:-, R2=0, R3=0, R4=0 A.E: R2 = 30h
R3 = 40h, R4=41h.





Eg Pop 4 ; pop 3 ; pop 2 ; pop Stack into R4, R3, R2 of Bank 0.

* Arithmetic group of Instruction :-

1. ADD A, Source.

Syntax : Add A, operand.

function : Addition.

Adds the content of Source with the content of accumulator and result is stored in accumulator.

flags affected : C, AC & OV

OV is set if there is a overflow.

2. ADDC A, Rn.

Byte : 1 cycle : 1

operation : (A) ← (A) + (C) + Rn

Simultaneously adds the content of Rn register, Accumulator, carry flag and result is stored in accumulator.

flag affected : OV, CY, AC.

& ADDC A, R7

A=01 A=03
C=01 C=01
R7=01 R7=01

ADDC A, 80h (direct)

A=02 A=05
C=01 C=01
80h=02 80h=02

ADDC A, @R0

A=01 A=03
C=01 C=01
R0=70h R0=70h
70h=01 70h=01

ADDC A, # data.

ADDC A, # 01h.

A=01h. A=03
C=01 C=01

2. MUL AB: multiplies the unsigned 8bit integer. the content of accumulator is multiplied with the content of B register.

operation A(7-0) X B(15-8)

A X B.

flag affected : C, OV

& MOV A, #5
MOV B, #7.
MUL AB.

7X5 = 35 = 23h.
A = 23h.
B = 00

50h X A0h = 3200h.

A=50h A=00h (0-7) lower byte
B=A0h B=32h (15-8) higher byte.

100 X 200 = 20,000 = 4E20h.

A=20h, B=4E.