8051 Instruction set & Programming

* Addressing Modes of 8051 :

Addressing modes indicates how you are addressing a given memory location.

Types of addressing modes -

1) Immediate Addressing
2) Direct addressing
3) Indirect addressing
4) Register addressing
5) External direct addressing
6) External indirect addressing

1) Immediate addressing :

If data or value are directly used as part of instruction then it is called as immediate addressing mode

symbol is used to represent the data

The source operand will be data and destination operand will be memory location or register.

e.g. MOV A, #20H

This instruction loads the immediate data byte 20H in A register

MOV DPTR, #1234H

This instruction loads DPTR with DD. 1234H.

2) Direct Addressing :

If the source and destination operand or any one can be used as memory location then it is called as direct addressing.

In this mode, internal register PC & used to access internal memory, it is not used to access external data memory

e.g. MOV A, 30H

This instruction will read the data from internal RAM address 30H and store it in the accumulator.

MOV 0AH, R1

Store the content of R1 to destination data memory location 0AH.

3) Indirect Addressing (Register indirect):

In this addressing mode, only registers R0 & R1 can be used as a pointer to the memory location where data is stored.

Using this mode, we can access internal as well as external data memory

using x in mnemonic - internal data memory accessed

using without x in mnemonic - external data memory accessed

using c in mnemonic - internal & external program memory accessed.

e.g. MOV A, @R0.

This instruction load the accumulator with the value from internal RAM location whose address in R0 Register.

4) Register Addressing -

In this addressing modes, all operand ie source and destination are used as register. R0 to R7, DPTR, B & A of microcontroller.

e.g. INC R0.

It indicates increment the value by 1, which is stored in R0 register.

MOV A, R1

In above example, source and destination's operand are register accumulator A and register R1. Copy the content of R1 into accumulator.

5) External Direct Addressing :

External memory is accessed by the using an instructions so addressing mode is called as External Direct addressing mode.

This addressing mode is used to access external memory rather than internal memory.

e.g. MOVX A, @DPTR

MOVX @DPTR, A

DPTR register is used to access address of external memory.

⑤ External Indirect Addressing Mode :

In this mode, small amount of external memory can also be accessed using indirect addressing mode.

e.g MOVX @R0, A

This external moves the content of Accumulator into external memory location stored in R0 register.

Group of Instruction Set :

Depending on operation they perform, all instructions are divided in several group :

① Arithmetic Instructions
② Branching Instructions
③ Data transfer Instructions
④ Logical Instructions
⑤ Stack operation Instructions
⑥ Boolean Operations Instructions

(1) Data Transfer Instruction Set :

i) MOV :

MOV instruction copies the content of source operand into destination operand.
Source operand will not change but destination operand get changed with source value. It is used for internal memory only.

Syntax -

```
MOV  destination,  source
     operand       operand
```

eg. MOV R0, #50H

50H data is copied into R0 register.

MOV A, @R0.

R0 register contains internal address. Content of address is copied into accumulator.

MOV R1, A

Content of accumulator copied into Register R1

ii) MOVC :

MOVC copies the content of code memory into the accumulator.

Syntax MOVC A, @A+<base-register>

The address of code memory is calculated by

adding the value of accumulator with base reg. as DPTR or Program counter (PC).
In this case program counter (PC) is first incremented by 1 before being summed with accumulator.

eg MOVC A, @A+DPTR

Content of address calculated by adding A & DPTR is moved into accumulator.

MOVC A, @A+PC

Content of address calculated by adding A & PC is moved into accumulator.

(3) MOVX :

MOVX moves the content from or to external memory into or from the accumulator.

If destination is @DPTR, the accumulator is moved to the 16 bit external memory specified in DPTR.
If source is DPTR then the byte is moved from external memory into accumulator.
If destination is @R0, or @R1 the accumulator is moved to 8 bit external memory address & vice versa.

Syntax: MOVX destination operand, source operand

e.g. MOVX, A, @R1
MOVX @R0, A

first example, explains data moved from external memory to accumulator & second explain from accumulator to external memory

④ XCH :

This instruction exchanges the content of accumulator & source ~~and~~ with destination operand. The operand contain may be register R0 to R7, of internal RAM, accumulator & internal address.

syntax -

XCH destination operand, source operand.

e.g

XCH A, R1

Exchange the content of R1 register ~~and~~ with accumulator A

XCH A, 50H.

Exchange the content of memory 50H with accumulator A.

⑤ XCHD :

This ~~external~~ instruction exchanges the bit's 0 to 3 of accumulator with the 0 to 3 bit of internal RAM address.

Syntax -

XCHD destination operand , source Operand

e.g. XCHD A, @R0

Internal RAM location 20H contain; R0 register 0 to 3 bit's of 20H @ memory is exchanged with ~~of~~ 0 to 3 bit's of accumulator.

II) Arithmetic Instructions :

These instructions performs operations like addition, subtraction, division, multiplication.

▷ ADD : (Addition)

This instruction is used to add source operand with destination operand. Result is stored in destination operand.

During addition operation carry & auxillary carry flags are affected if carry generate from bit 7 or bit 3 respectively.

Syntax -

ADD destination operand , source operand

e.g. ADD A, R0

Content of R0 register is added with

accumulator and result is stored in accumulator.

```
ADD A, @R1
ADD A, #25H.
```

2) ADDC : (Addition with carry flag)

ADDC is used to add the ~~byt~~ destination operand with source operand with carry flag and store result in destination operand.

This instruction is used to add the carry flag into result of addition.

Syntax - ADDC destination operand, source operand

e.g.

```
ADDC A, R0
```

Content of R0 register is added with accumulator A, it also add the carry flag into the result.

```
ADDC A, 30H
ADDC A, @R1
ADDC A, #20H
```

3) SUBB : (subtraction with borrow) :

SUBB is used to subtracts the source operand from destination operand. It also subtract carry/borrow flag from result.

This instruction set carry flag if borrow is is generated.

Syntax- SUBB destination , source
operand operand

e.g. SUBB A, R0

It subtract the content of R0 register from content of accumulator also subtract carry flag if borrow generated. Result is stored in accumulator.

SUBB A, #30H
SUBB A, 25H
SUBB A, @R1

4) MUL : (multiply) A with B.

MUL instruction is used to multiply content of accumulator with register B.

If result is greater than 8 bit or 16 bit then it is stored in accumulator and high order byte in B register.

e.g MUL AB

Accumulator content is multiplied with content of B & result is stored in accumulator. If result is 8 bit. If product is 16 bit then lower order 8 bit stored in accumulator and higher order 8 bit stored in register B.

5) DIV : (division) : divide A by B

DIV instruction is used to divide the content of accumulator by the content of register B. After division operation, accumulator contains quotient and register B contains remainder.

e.g. DIV AB

6) INC : Increment by 1

INC instruction is used to increment the content of destination operand by 1.

Syntax - INC destination operand

e.g. INC R0

This instruction increment of content of R0 register by 1.

INC @R0
INC A
INC DPTR

This instruction increment the data pointer (16bit) by 1.

7) DEC: decrement by 1

DEC instruction is used to decrement the content of destination by 1.

Syntax - DEC destination operand

e.g. DEC A

Above instruction decrement the content of accumulator by 1.

```
DEC R0
DEC @R2
DEC 20H
```

8] DAA : (decimal adjust accumulator after addition)

This instruction adjust the content of accumulator correspond to a BCD no. after the addition of two BCD no's.

This instruction adds the 6 into result if the result of addition is greater than 9.

e.g.
```
ADDC A, R3
DAA
```

ADDC instruction adds the content of R3 register into accumulator. DAA instruction converts invalid BCD into valid BCD by adding 6.

Suppose A = 06 = 0000 0110
R3 = 05 = 0000 0101

A + R3 = 0000 0110
+ 0000 0101
0000 1011 → 11>9

= 0001 0001
= 11

After DAA instruction
+ 0110
0001 0001

5) Logical Instructions:

These instructions are used to perform logical operations like ANDing, ORing etc.

i) ANL (Logical AND):

This instruction performs AND operation between source and destination operand bit by bit. Result is stored in destination operand.

e.g ANL A, R0

ANL performs logical AND operation in between A & R0 & result will be in accumulator A.

```
ANL A, 40H
ANL A, @R0
ANL A, #25H
```

ii) ORL (Logical OR):

The ORL instruction performs OR operation between source & destination operand bit by bit. Result is stored in destination operand.

e.g. OR A, 30H

It performs logical OR operation in between source and destination operand. Result will be in destination operand.

```
OR A, #25H
OR A, B
OR A, @R2
```

iii) XRL (logical EX-OR):

It is used to perform logical Ex-OR operation in between source and destination operand. Result will be stored in destination operand.

eg. XRL A, B

It performs logical Ex-OR operation in bitwise accumulator A & content of B & result will be in the accumulator A.

XRL A, #20H
XRL A, 30H
XRL A, @R0

iv) CLR A ; (clear accumulator):

It is used to clear the content of accumulator i.e all bits set to 0.

e.g. CLR A

If accumulator contains 35H data before operation then after execution of this instruction accumulator content will be 00H.

iv) CPL A (complement A):

It is used to find complement of the accumulator A.

e.g. CPL A.

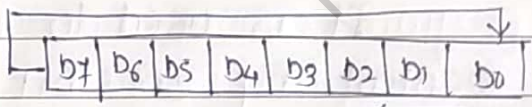
Suppose before operation it contains 00001111B then after execution of complement operation accumulator A contains 11110000 data.

iv) RL A (Rotate Accumulator to left):

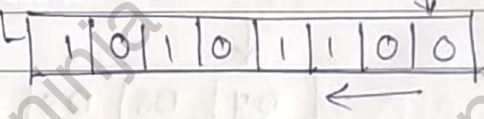
Rotate the accumulator content by left by one bit.

After execution of this instruction content of D7 bit goes to D0 bit and all bits shifted towards left side.

e.g. RL A



before operation A = 1 0 1 0 1 1 0 0

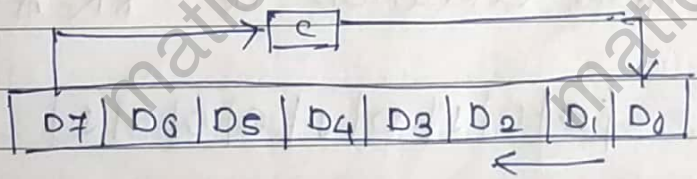


after RL A instruction.

we get A = 0 1 0 1 1 0 0 1

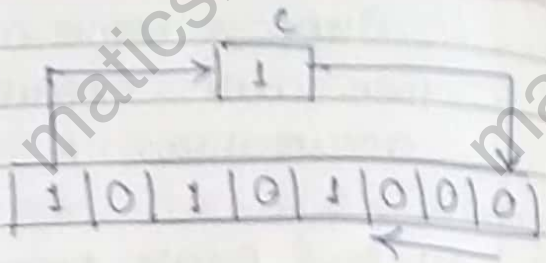
v) RLC A (Rotate accumulator left with carry):

This instruction is used to rotate the content of accumulator left by 1 bit. It uses carry flag in between its operation.

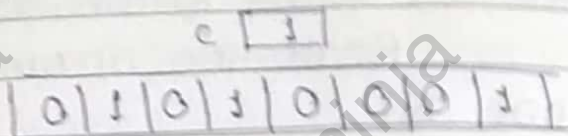


In this D7 bit loaded into carry flag, carry flag content loaded into D0 bit. All bits will be loaded in left side by 1

eg RLC A

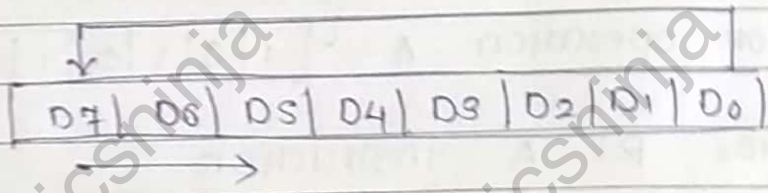


after RLC A

we get 

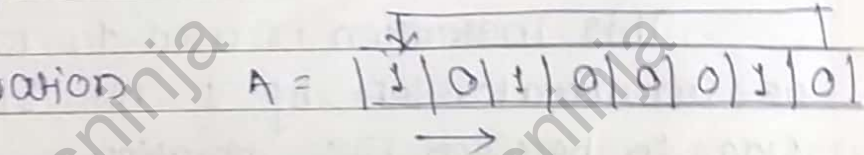
vi) RRA (Rotate Accumulator toward Right):

It is used to rotate the accumulator content by one bit to right.



In this D_0 bit loaded into D_7 bit & all bit will shifted towards right side

eg RRA A

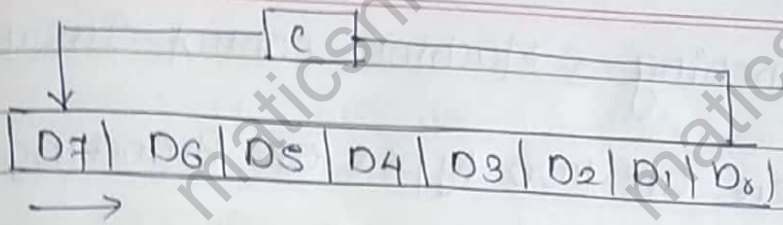


After execution of RRA

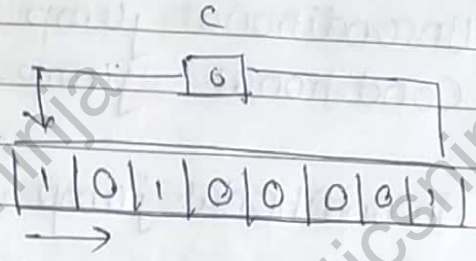
A = 0 1 0 1 0 0 0 1

vii) RRC A (Rotate Accumulator content by to right with carry)

It is used to rotate accumulator content to right by 1 bit using carry flag.

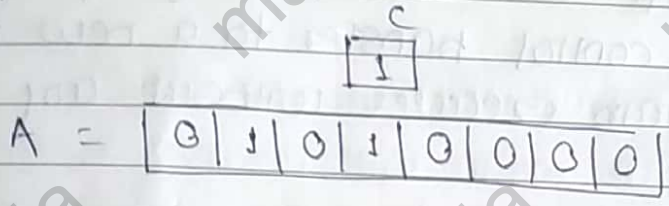


eg RRC A



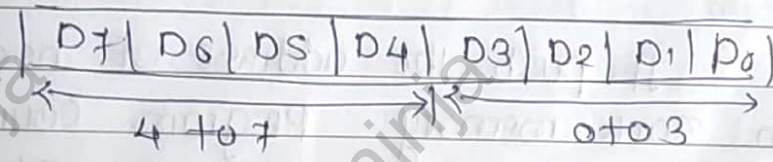
before operation

After RRC A we get



SWAP A:

This instruction is used to swap or exchange the lower nibble bits & higher nibble bits of accumulator (8 bit)



bit 0 to 3 are exchanged with bit 4 to 7.

SWAP A

Suppose A contains 25H then after execution we get 52H.

Branching & Machine control Instructions:

There are two types of Branching Instructions.

① Unconditional jump
② Conditional jump.

① Unconditional jump -

After execution of jump instruction, program control transfer to a new location & program executes without any condition.

i) ACALL (Absolute call):

ACALL instruction is used to call the subroutine unconditionally at the given 11 bit absolute code address.

It push the address of next instruction on stack memory. Program counter is then loaded with the 11 bit next address & program executes.

12-36

e.g. ACALL 02CH

ii) LCALL (Long CALL):

It is used to call subroutine. LCALL instruction stores the return address.

It push the address of next instruction on stack memory. Program counter is then loaded to the 16 bit subroutine address & program executes.

e.g. LCALL 4000H.

iii) RET : (Return from subroutine)

RET instruction is used to return from a subroutine called by LCALL or ACALL instruction.

This instruction returns the program execution from subroutine to the main program.

e.g RET.

iv) RETI (Return from interrupt):

This instruction is used to return from an ISR (interrupt service routine)

It transfers the program execution control from interrupt to main program. Program execution continues the address that is copied from stack memory.

v) AJMP (Absolute Jump):

It is used to unconditional jump to the specific code address.

It uses 2kb memory

vi) LJMP (Long Jump):

This instruction is used to unconditional jump to the 16 bit address specified in the LJMP instruction.

Address may be anywhere within 64k of program memory address.

eg. LJMP 1234H

vii) JMP : (Short Jump):

SJMP instruction is used to unconditional jump to the signed 8 bit relative address specified in the JMP instruction.

Address is stored in stack memory. Program counter loads the 8 bit relative address where program executes.

eg. JMP 23H
JMP UP.

viii) JMP @A+DPTR:

This instruction is used to jump unconditional to the address calculated by adding the value of DPTR & accumulator. The resulting address stored in program counter which starts the program execution from specific address.

e.g JMP @A+DPTR.

Conditional Instructions:

i) JZ : (jump if accumulator zero):

If the content of accumulator is zero then this instruction jump to the address specified in instruction otherwise continues with next instruction.

eg. JZ UP1
DEC A
JZ UP2

Suppose A contains 1, if A first A is decremented by 1 if it is zero then address of next instruction is fetched other specific address executes.

ii) JNZ (jump if accumulator is not zero)

This instruction jump to specific address if the accumulator content is not zero. otherwise continues with next instruction.

e.g. JNZ UP.

iii) CJNE (compare & jump if not equal):

This instruction compares the source & destination operand & jump to the specific address if they are not equal. otherwise continues it's execution to the next instruction.

e.g. CJNE R0, #20H, DOWN

DOWN: MOV A, B.

CJNE R0, @R1, UP
CJNE A, 25H, UP1

iii) DJNZ (decrement jump if Not zero):

This instruction decrement the accumulator destination content by 1 and jump to specific address if destination content is not zero.

eg. DJNZ R0, UP
DJNZ 50H, DOWN

v) NOP : No operation

This instruction transfer the program execution without performing any operation. This instruction can be used to add small time delay

e.g. NOP

Stack Instructions :

i) PUSH : (push data into stack):

This instruction copies the value of specific address (RAM) into the stack memory. During this stack pointer is increment by 01 if data is 08 bit

eg. PUSH DPL
PUSH DPH

3) POP (POP from stack):

This instruction removes the content of stack memory into the internal RAM memory.

It decrement the stack pointer by 1 if data stored is 8 bit.

eg. POP DPH
POP DPL

Boolean Operation Instructions:

1) SETB (set bit):

This instruction is used to set the destination operand bit to one.

eg. SETB C
SETB P1.0

It indicates carry flag & port bit P1.0 set to the one.

2) JC (jump IF carry):

This instruction is used to jump to the address specified in the instruction if the carry flag set to 1 & it continues with next instruction.

JC UP

3) JNC (jump if not carry):

This instruction is used to jump to the specific address if carry flag reset to zero

e.g. JNC UP1

4) JB (Jump if bit set)

This instruction is used to set the to the jump bit specific address if bit is set to one.

JB P1.2, Lable.1

5) JNB (jump if bit NOT set) =0.

This instruction is used to jump to the specific address if bit is not set to 1.

JNB P1.2, Lable.1

6) ~~JBC (jump if bit is set 1.~~

6) CLR → clear bit

e.g. CLR ~~20H~~ C

7) CPL - complement bit e.g. CPL C

CPL P1.2

8) ANL - Logical AND → e.g. ANL A, B

ANL A, P1.2

ANL C, P1.2

Program Development steps :

1) Defining the problem
2) Algorithm
3) flowchart
4) Initialization checklist
5) choosing Instruction
6) converting ~~ALP~~ Algorithm into ALP.

1) Defining the problem :

This is first step of writing program. problem statement must be clear and think carefully how to solve that problem.

2) Algorithm:

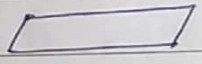
The sequence of tasks performed by program written in general english is called as algorithm.

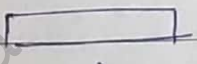
3) Flowchart :

The graphical representation of program steps is called as flowchart.

In this specific operation p represented by symbol like

 - start/stop

 - Input/output

 - operation

 - condition

↓↑ - flow of program

Keil & SPJ have it's own cross compiler.

4) Linker :

Linker combines the more than one separately assembled module into one executable program. It generate executable file having extension .exe.

Linker links the different module stored in different files together into single final program.

5) Locator :

Locator is used to assign specific addresses for program code.

Keil & SPJ have an its own locater with linker.

6) Compiler :

A compiler is a computer program which executes the executable code and finds the errors from executable code.

Assembler Directives:

Directives are used to define symbol, values, reserve & initialize storage space for variables.

① ORG:

ORG directive is used to define location in program memory where program is placed.

```
e.g.  ORG  1000H
      ORG  0000H.
```

② DB: (define byte):

It is used to define 8 bit data
D used for decimal, B - for binary,
H for hexadecimal value.

```
eg.  NUM  DB  28   ; decimal no 28
     NUM1 DB  34H  ; Hexadecimal no. 34H.
```

③ EQU: (equal):

It is used to define constant to which memory location is not allocated by assembler. It is used begining in program.

```
e.g.  PORT EQU 40H.
```

④ CODE:

Using this directive, an address in code memory is assigned as a symbol. code memory size is 64K. The address should be within range 0 to 65535.

e.g. START CODE 0 ; memory location 0 is called START.

5) DATA :

Using DATA directive, an address within internal data RAM is assigned to a symbol & address must be in the range of 0 to 255.

e.g. TEMP DATA 32 ; register at address 32 is named as TEMP.

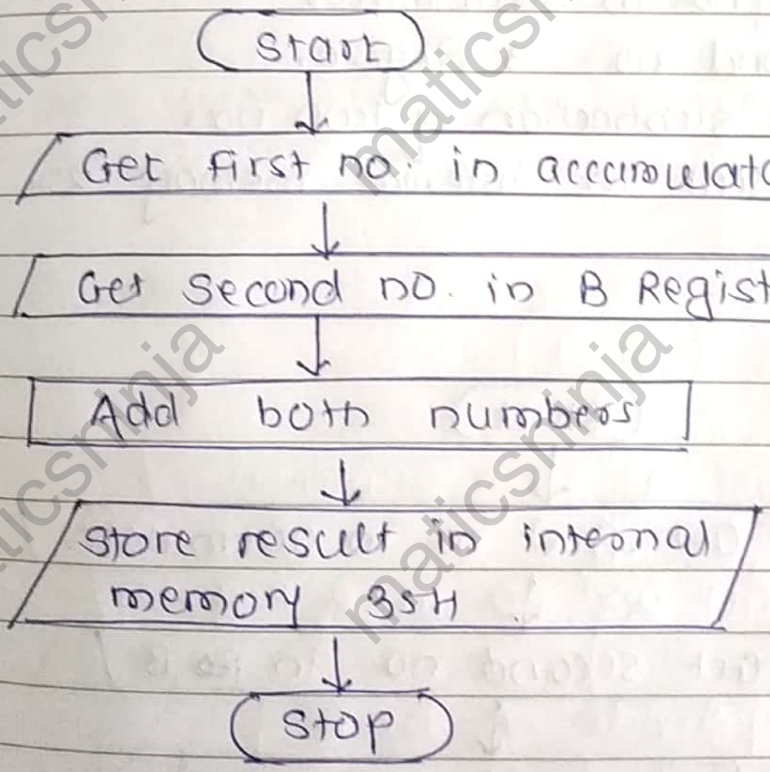
Assembly Language Programming:

Write and execute a program to add data 40H & 30H and store result in 35H internal memory

(I) Algorithm:

1. Get first no. in accumulator
2. Get second no. in B register
3. Add both numbers
4. store result in internal memory location 35H
5. Stop.

(II) Flow chart:



(III) ALP:

```
ORG 0000H
MOV A, #40H
MOV B, #30H
ADD A, B
MOV 35H, A
END
```

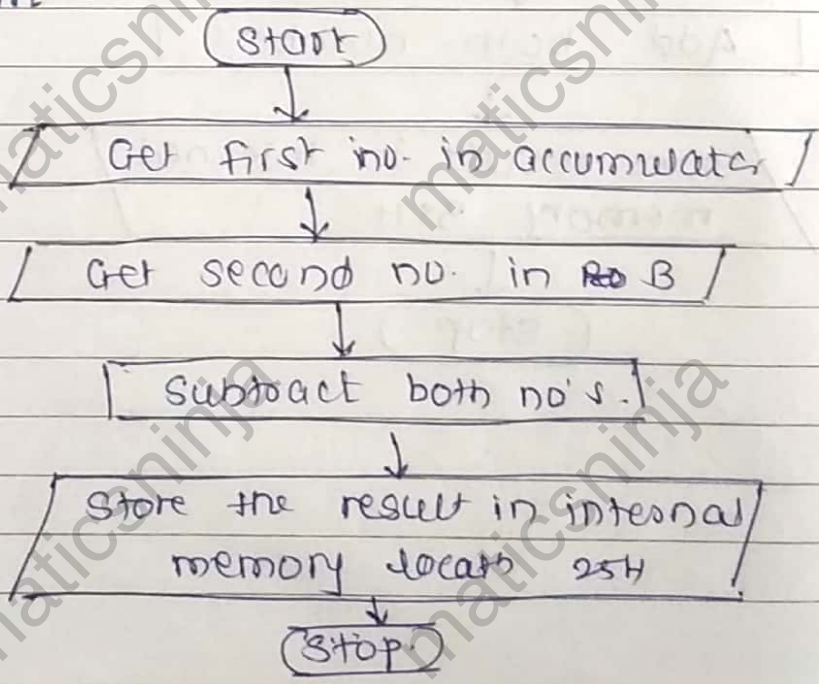
② Write a program to subtract data 40H & 30H & store result in 25H internal memory.

→

(I) Algorithm:

① Get first no. in accumulator
② Get second no. B Register
③ perform subtraction of two no's
④ Store result in internal memory 25H
⑤ Stop

(II) flowchart:



(II) ALP:

```
ORG 0000H
MOV A, #40H
MOV B, #30H
SUBB A, B
MOV 25H, A
END.
```

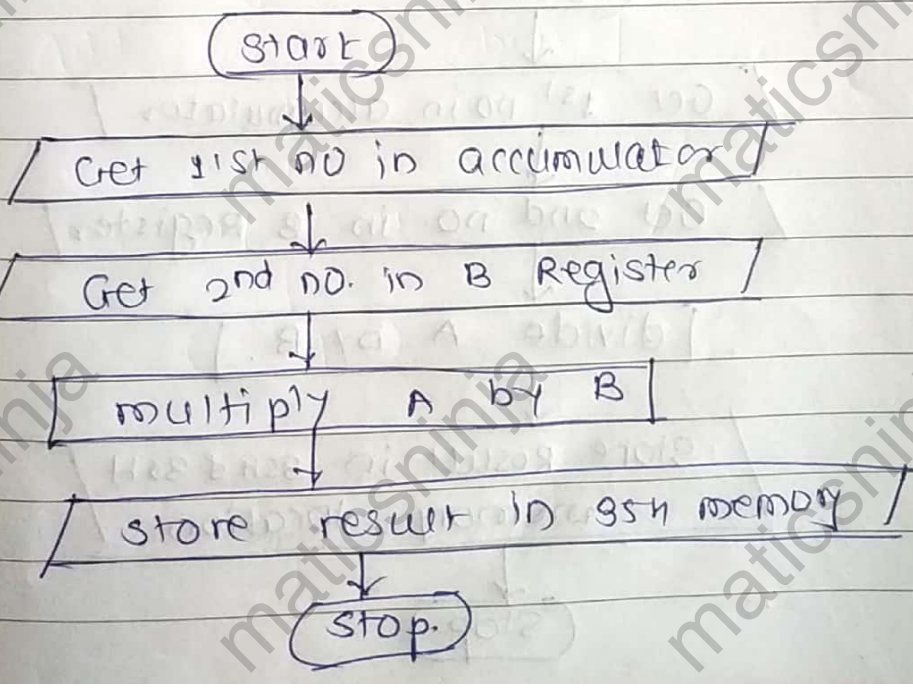
(3) Write an ALP to multiply data 03H & 02H & store result in 35H internal memory.

→

(I) Algorithm :

1. Get 1st no. in accumulator
2. Get 2nd no. in B Register
3. multiply A by B.
4. Store result in 35H memory.
5. Stop.

(II) flowchart :



(III) ALP

```
ORG 0000H
MOV A, #03H
MOV B, #02H
MUL AB
MOV 35H, A
END.
```

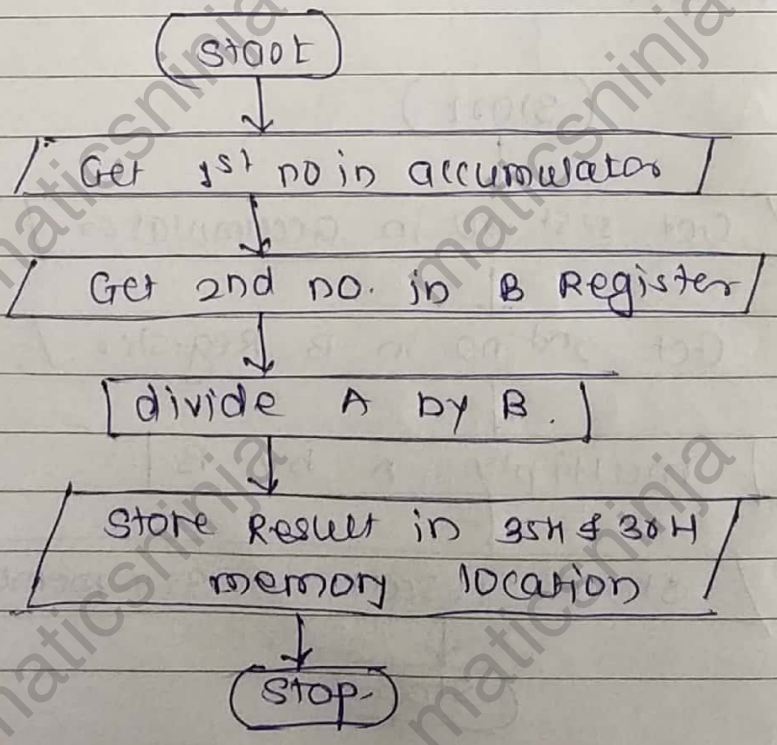
(4) Write ALP to perform division data 40H & 02H.

→

(I) Algorithm:

① Get 1st no. of accumulator
② Get 2nd no. in B Register
③ ~~multiply~~ divide A by B.
④ Store result in 35H (Quotient) & 36H (Remainder)
⑤ Stop.

(II) flowchart:



(iii) ALP :

```
ORG 0000H
MOV A, #40H
MOV B, #02H
DIV AB
MOV 35H, A
MOV 36H, B
END
```