Expt-1: Write an ALP to perform arithmetic operations using 8051

```
; ADDITION
MOV A, #15H       ; Load immediate value 15H into Accumulator (A)
MOV B, #10H       ; Load immediate value 10H into register B
ADD A, B          ; Add contents of B to A &  A = 15H + 10H = 25H
MOV 40H, A        ; Store the result of addition (25H) into internal RAM location 40H
; SUBTRACTION
MOV A, #15H       ; Load immediate value 15H into A
MOV B, #10H       ; Load immediate value 10H into B
SUBB A, B         ; Subtract B from A with borrow ,  A = 15H - 10H = 05H
MOV 41H, A        ; Store result (05H) into internal RAM location 41H
;  MULTIPLICATION
MOV A, #05H       ; Load 05H into A
MOV B, #07H       ; Load 15H into B
MUL AB            ; Multiply A × B & result = 05H × 07H = 23H (Low byte in A, High byte in B)
MOV 42H, A        ; Store lower byte of result into 42H
MOV 43H, B        ; Store higher byte of result into 43H
;  DIVISION
MOV A, #24H       ; Load 24H (36 in decimal) into A
MOV B, #05H       ; Load 05H (5 in decimal) into B
DIV AB            ; Divide A by B, Quotient in A, Remainder in B
MOV 44H, A        ; Store quotient into 44H
MOV 45H, B        ; Store remainder into 45H
END               ; End of program
```

Expt-2: Write an ALP to perform logical operations using 8051

```
; AND OPERATION
MOV A, #35H        ; Load immediate value 35H into Accumulator (A)
MOV R0, #25H       ; Load immediate value 25H into register R0
ANL A, R0          ; Logical AND,  A = 35H AND 25H
                   ; 35H = 0011 0101 (binary)  ; 25H = 0010 0101 ; AND = 0010 0101 = 25H
MOV R3, A          ; Move result (25H) into register R3
MOV 40H, R3        ; Store result into RAM location 40H
;  OR OPERATION
MOV A, #35H        ; Reload 35H into A
MOV R0, #25H       ; Reload 25H into R0
ORL A, R0          ; Logical OR ,  A = 35H OR 25H
                   ; 35H = 0011 0101
                   ; 25H = 0010 0101
                   ; OR  = 0011 0101 = 35H
MOV R3, A          ; Move result (35H) into R3
MOV 41H, R3        ; Store result into RAM location 41H
; XOR OPERATION
MOV A, #35H        ; Reload 35H into A
MOV R0, #25H       ; Reload 25H into R0
XRL A, R0          ; Logical XOR , A = 35H XOR 25H
                   ; 35H = 0011 0101 ; 25H = 0010 0101  ; XOR = 0001 0000 = 10H
MOV R3, A          ; Move result (10H) into R3
MOV 42H, R3        ; Store result into RAM location 42H
;  NOT OPERATION
MOV A, #35H        ; Load 35H into A
CPL A          ; Complement (NOT),  A = NOT(35H)
               ; 35H = 0011 0101
               ; CPL = 1100 1010 = CAH
MOV 43H, A         ; Store complemented value (CAH) into RAM location 43H
END                ; End of program
```

Expt-3: Write an ALP to move a block of n bytes of data from the source (20h) to the destination (40h) using Internal RAM.

```
ORG 00H          ; Origin - program starts at memory location 0000H
MOV R0, #20H     ; Load R0 with 20H → Source starting address in RAM
MOV R1, #40H     ; Load R1 with 40H → Destination starting address in RAM
MOV R2, #0AH     ; Load R2 with 0AH (10 decimal) → Counter (number of bytes to move)
UP: MOV A, @R0   ; Move the data from source address (pointed by R0) into Accumulator
 MOV @R1, A      ; Store the data from Accumulator into destination address (pointed by R1)
 INC R0          ; Increment source pointer → next source address
 INC R1          ; Increment destination pointer → next destination address
DJNZ R2, UP      ; Decrement R2 and repeat until R2 = 0 (copies 10 bytes total)
EXIT: SJMP EXIT  ; Infinite loop → Stop program here
END              ; End of program
```

Expt-4: Write an ALP To exchange the source block starting with address 20h (Internal RAM) containing N (05) bytes of data with the destination block starting with address 40h (Internal RAM).

```
   ORG 0000H
   MOV R0, #20H      ; R0  points to Source block (20H)
   MOV R1, #40H      ; R1  points to Destination block (40H)
   MOV R2, #05H      ; Load R2 with Counter = 5 bytes
EXCHANGE:
  MOV A, @R0    ; Move data into A whose address is in internal RAM
   MOV B, @R1    ; Move data into A whose address is in internal RAM
   MOV @R0, B        ; Put Destination data into Source
   MOV @R1, A        ; Put Source data into Destination
   INC R0           ; Point to next Source byte
   INC R1           ; Point to next Destination byte
   DJNZ R2, EXCHANGE  ; Repeat for all 5 bytes
   EXIT: SJMP EXIT           ; Stop (infinite loop)
   END
```

Expt 5: Write an ALP to add the byte in the RAM at 34h and 35h, and store the result in the register R5 (LSB) and R6 (MSB), using Indirect Addressing Mode.

```
ORG 0000H
        MOV R0, #34H      ; Point R0 to 34H
        MOV A, @R0        ; A = value at 34H
        MOV R1, #35H      ; Point R1 to 35H

        ADD A, @R1        ; A = (34H + 35H), Carry in CY
        MOV R5, A         ; Store LSB of result in R5
        MOV R6, #00H      ; Clear R6 before adding carry
        JNC NEXT          ; If no carry, skip increment
        INC R6            ; If carry, MSB = 1
NEXT: SJMP NEXT
        END
```

Expt 6: Write an ALP to subtract the bytes in Internal RAM 34h & 35h and store the result in register R5 (LSB) & R6 (MSB).

```
ORG 0000H
        MOV R0, #34H      ; R0 points to 34H
        MOV A, @R0        ; A = value at 34H
        MOV R1, #35H      ; R1 points to 35H
        CLR C             ; Clear carry before SUBB
        SUBB A, @R1       ; A = (34H - 35H), CY=borrow
        MOV R5, A         ; Store LSB in R5
        MOV R6, #00H      ; Clear MSB
        JNC NEXT          ; If no borrow, skip
        INC R6            ; If borrow, MSB = 1
NEXT: SJMP NEXT           ; Stay here
        END
```

Expt 7: Write an ALP to multiply two 8-bit numbers stored at 30h and 31h and store16- bit results in 32h and 33h of Internal RAM.

```
ORG 0000H
    MOV R0, #30H    ; R0 = 30H (address of first number)
    MOV A, @R0      ; A = [30H]
    INC R0          ; R0 = 31H
    MOV B, @R0      ; B = [31H]
    MUL AB          ; A = low byte, B = high byte

    INC R0          ; R0 = 32H
    MOV @R0, A      ; Store LSB at 33H
    INC R0          ; R0 = 33H
    MOV @R0, B      ; Store MSB at 34H

    END
```

Expt-8: Write an ALP to perform a division operation on the 8-bit number by an 8-bit number.

```
    MOV A, 30H    ; Load dividend into Accumulator (A)

    MOV B, 31H    ; Load divisor into B register

    DIV AB        ; Divide A by B → Quotient in A, Remainder in B

    MOV 32H, A    ; Store quotient at 32H

    MOV 33H, B    ; Store remainder at 33H

    END
```

Expt 9: Write an ALP to separate positive and negative in a given array.

a) Write an ALP to separate Positive numbers in a given array.

```
org 0000h
mov r0, #30h       ; starting address of array data
mov r1, #50h       ; address for storing Positive numbers after separation
mov R2, #05        ; array size 5
back: mov a, @r0   ; take data from address present in R0
    JB acc.7, skip ; if MSB bit of accumulator is set ? negative, skip
    mov @r1, a     ; if MSB bit of acc is not set ? positive, store in 50H
    inc r1         ; increment R1 by 1
skip: inc r0       ; increment R0 by 1
    djnz r2, back ; decrement R2, if not zero jump to label back
here: sjmp here    ; jump to this line again (infinite loop)
end
```

b) Write an ALP to separate Negetive numbers in a given array.

```
org 000h
mov r0, #30h       ; starting address of array datas
mov r1, #50h       ; address for storing Positive numbers after separation
mov R2, #05        ; array size 5
back: mov a, @r0   ; take data from address present in R0
    JNB acc.7, skip ; jump if bit =0 negative, skip
    mov @r1, a     ;
    inc r1         ; increment R1 by 1
skip: inc r0       ; increment R0 by 1
    djnz r2, back ; decrement R2, if not zero jump to label back
here: sjmp here    ; jump to this line again (infinite loop)
end
```

Expt 10 : Write an assembly language program to separate even or odd elements in a given array

a) Separate Even numbers:

```
ORG 0000H
   MOV R0, #30H     ; R0 points to start of array (input)
   MOV R1, #50H     ; R1 points to storage location
   MOV R2, #05H     ; Counter = 5 elements
BACK: MOV A, @R0     ; Get data from array
     JB ACC.0, SKIP ; If LSB = 1, skip storing
     MOV @R1, A     ; If LSB = 0, store in new array
     INC R1
SKIP: INC R0         ; Next element
     DJNZ R2, BACK   ; Repeat until counter = 0
HERE: SJMP HERE      ; Infinite loop
END
```

b) Separate Odd numbers:

```
ORG 0000H
   MOV R0, #30H     ; R0 points to start of array (input)
   MOV R1, #50H     ; R1 points to storage location
   MOV R2, #05H     ; Counter = 5 elements
BACK: MOV A, @R0     ; Get data from array
     JNB ACC.0, SKIP ; If LSB = 0, skip storing
     MOV @R1, A     ; If LSB = 1, store in new array
     INC R1
SKIP: INC R0         ; Next element
     DJNZ R2, BACK   ; Repeat until counter = 0
HERE: SJMP HERE      ; Infinite loop
END
```