Practical 1. Perform the following Operations related to memory locations.

a. Store the data byte 32H into memory location 4000H.

```
; Initialize the HL register pair to point to memory location 4000H
LXI H, 4000H

; Move the immediate value 32H into the accumulator
MVI A, 32H

; Store the contents of the accumulator (32H) into the memory location pointed to
by HL (4000H)
MOV M, A

; Halt the program
HLT
```

b. Exchange the contents of memory locations 2000H and 4000H

```
; Initialize HL and DE register pairs to point to the respective memory locations
LXI H, 2000H   ; HL points to 2000H
LXI D, 4000H   ; DE points to 4000H

; Store the content of memory location 2000H (pointed by HL) into register B
MOV B, M

; Load the content of memory location 4000H (pointed by DE) into the
accumulator
LDAX D

; Store the content of the accumulator (originally from 4000H) into memory
location 2000H (pointed by HL)
MOV M, A

; Move the content of register B (originally from 2000H) into the accumulator
```

```
MOV A, B

; Store the content of the accumulator (originally from 2000H) into memory
location 4000H (pointed by DE)
STAX D

; Halt the program
HLT
```

Practical 2. Simple assembly language programs.

a. Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

```
LXI H, 2000H   ; HL points to 2000H
LXI D, 4001H   ; DE points to 4001H

; Load the content of 2000H into the accumulator
MOV A, M

; Subtract the content of 4001H from the accumulator
SUB M

; Store the result in 4002H
LXI D, 4002H
STAX D

HLT
```

b. Subtract two 8-bit numbers.

```
; Load the first number into the accumulator
MVI A, 80H

; Load the second number into register B
MVI B, 20H

; Subtract the second number from the first
SUB B

; Store the result in memory location 5000H
LXI H, 5000H
MOV M, A

HLT
```

c. Add the 16-bit number in memory locations 4000H and 4001H to the 16-bit number in memory locations 4002H and 4003H. The most significant eight bits of the two numbers to be added are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

```
; Initialize HL and DE register pairs to point to the respective memory locations
LXI H, 4000H   ; HL points to 4000H
LXI D, 4002H   ; DE points to 4002H

; Load the low byte of the first number into the accumulator
MOV A, M

; Add the low byte of the second number to the accumulator
ADD M

; Store the low byte of the result in 4004H
LXI D, 4004H
MOV M, A

; Load the high byte of the first number into the accumulator
INX H
MOV A, M

; Add the high byte of the second number to the accumulator with carry
ADC M

; Store the high byte of the result in 4005H
INX D
MOV M, A

HLT
```

d. Add the contents of memory locations 4000H and 4001H and place the result in the memory locations 4002H and 4003H.

```
; Initialize HL and DE register pairs to point to the respective memory locations
LXI H, 4000H   ; HL points to 4000H
LXI D, 4002H   ; DE points to 4002H

; Load the low byte of the number into the accumulator
MOV A, M

; Add 0 to the accumulator (to clear the carry flag)
ADD A

; Store the low byte of the result in 4002H
MOV M, A

; Load the high byte of the number into the accumulator
INX H
MOV A, M

; Add the carry from the previous addition
ADC A

; Store the high byte of the result in 4003H
INX D
MOV M, A

HLT
```

e. Subtract the 16-bit number in memory locations 4002H and 4003H from the 16-bit number in memory locations 4000H and 4001H. The most significant eight bits of the two numbers are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

```
; Initialize HL and DE register pairs to point to the respective memory locations
LXI H, 4000H   ; HL points to 4000H
LXI D, 4002H   ; DE points to 4002H
```

```
; Load the low byte of the minuend into the accumulator
MOV A, M

; Subtract the low byte of the subtrahend from the accumulator
SUB M

; Store the low byte of the result in 4004H
LXI D, 4004H
MOV M, A

; Load the high byte of the minuend into the accumulator
INX H
MOV A, M

; Subtract the high byte of the subtrahend from the accumulator with borrow
SBB M

; Store the high byte of the result in 4005H
INX D
MOV M, A

HLT
```

f. Find the l's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H.

```
LXI H, 4400H   ; HL points to 4400H

; Load the number into the accumulator
MOV A, M

; Complement the accumulator
CMA

; Store the complemented number at 4300H
```

```
LXI H, 4300H
MOV M, A

HLT
```

g. Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.

```
LXI H, 4200H   ; HL points to 4200H

; Load the number into the accumulator
MOV A, M

; Complement the accumulator (1's complement)
CMA

; Add 1 to the accumulator (2's complement)
ADI 1

; Store the 2's complement at 4300H
LXI H, 4300H
MOV M, A

HLT
```

Practical 3. Packing and unpacking operations.

a. Pack the two unpacked BCD numbers stored in memory locations 4200H and 4201H and store result in memory location 4300H. Assume the least significant digit is stored at 4200H.

```
LXI H, 4200H ; HL points to the least significant digit

; Load the least significant digit into the accumulator
MOV A, M

; Shift the accumulator left by 4 bits
RLC
RLC
RLC
RLC

; Load the most significant digit into the accumulator
INX H
MOV A, M

; OR the most significant digit with the shifted least significant digit
ORA M

; Store the packed BCD number at 4300H
LXI H, 4300H
MOV M, A

HLT
```

b. Two digit BCD number is stored in memory location 4200H. Unpack the BCD number and store the two digits in memory locations 4300H and 4301H such that memory location 4300H will have lower BCD digit.

```
LXI H, 4200H ; HL points to the packed BCD number
```

```
; Load the packed BCD number into the accumulator
MOV A, M

; AND with 0FH to extract the lower digit
ANI 0FH

; Store the lower digit at 4300H
LXI H, 4300H
MOV M, A

; Load the packed BCD number again into the accumulator
LXI H, 4200H
MOV A, M

; Shift right 4 times to isolate the higher digit
RRC
RRC
RRC
RRC

; AND with 0FH to mask the higher digit
ANI 0FH

; Store the higher digit at 4301H
INX H
MOV M, A

HLT
```

Practical 4. Register Operations

a. Write a program to shift an eight bit data four bits right. Assume that data is in register C.

```
MVI C, 80H ; Load the data into register C

RRC ; Rotate right through carry 4 times
RRC
RRC
RRC

HLT ; Halt the program
```

b. Program to shift a 16-bit data 1 bit left. Assume data is in the HL register pair

```
LXI H, 1234H ; Load the 16-bit data into HL

DAD H ; Add HL to itself, effectively shifting left by 1 bit

HLT ; Halt the program
```

c. Write a set of instructions to alter the contents of flag register in 8085.

```
; Set the carry flag
CMC

; Set the sign flag
MVI A, 80H ; Load a negative number
ORA A ; Set the sign flag

; Clear the zero flag
MVI A, 01H ; Load a non-zero number
ORA A ; Clear the zero flag
```

```
; Set the parity flag
MVI A, 0AH ; Load a number with odd parity
ANA A ; Set the parity flag

; Clear the auxiliary carry flag
MVI A, 0FH ; Load a number that won't cause auxiliary carry
ADD A ; Clear the auxiliary carry flag
```

d. Write a program to count number of I's in the contents of D register and store the count in the B register.

```
MVI B, 00H ; Initialize the count to 0
MVI C, 08H ; Initialize the counter for 8 iterations

LOOP:
   RLC ; Rotate left through carry
   JNC SKIP ; If carry is 0, skip incrementing the count
   INR B ; Increment the count
SKIP:
   DCR C ; Decrement the counter
   JNZ LOOP ; Jump back to LOOP if counter is not zero

HLT ; Halt the program
```

Practical 5. Multiple memory locations.

a. Calculate the sum of series of numbers. The length of the series is in memory location 4200H and the series begins from memory location 4201H. a. Consider the sum to be 8 bit number. So, ignore carries. Store the sum at memory location 4300H.

```
LXI H, 4200H ; HL points to the length of the series

; Load the length into register B
MOV B, M

; Initialize the sum in register A to 0
MVI A, 00H

; Initialize HL to point to the first number in the series
LXI H, 4201H

LOOP:
    ; Add the current number to the sum
    ADD M

    ; Increment HL to point to the next number
    INX H

    ; Decrement the counter
    DCR B

    ; Jump back to the loop if the counter is not zero
    JNZ LOOP

; Store the sum at memory location 4300H
LXI H, 4300H
MOV M, A

HLT
```

Extra (

b. Consider the sum to be a 16 bit number. Store the sum at memory locations 4300H and 4301H

```
LXI H, 4200H ; HL points to the length of the series
MOV B, M ; Load the length into register B

MVI A, 00H ; Initialize the low byte of the sum to 0
MVI H, 4300H ; HL points to the low byte of the result

LOOP:
   ADD M ; Add the current number to the low byte of the sum
   DAA ; Adjust the result to BCD
   MOV M, A ; Store the low byte of the result

   INX H ; Increment HL to point to the high byte of the result
   MOV A, H ; Load the high byte of the result into the accumulator
   ADC A ; Add the carry from the low byte addition
   DAA ; Adjust the result to BCD
   MOV M, A ; Store the high byte of the result

   INX H ; Increment HL to point to the next number in the series
   DCR B ; Decrement the counter
   JNZ LOOP ; Jump back to the loop if the counter is not zero

HLT
```

)

b. Multiply two 8-bit numbers stored in memory locations 2200H and 2201H by repetitive addition and store the result in memory locations 2300H and 2301H.

```
LXI H, 2200H ; HL points to the multiplicand
MOV B, M ; Load the multiplicand into register B

LXI H, 2201H ; HL points to the multiplier
MOV C, M ; Load the multiplier into register C
```

```
MVI A, 00H ; Initialize the product to 0
MVI D, 00H ; Initialize the counter to 0

LOOP:
   ADD B ; Add the multiplicand to the product
   DAA ; Adjust the product to BCD
   MOV D, A ; Store the low byte of the product

   INX H ; Increment HL to point to the high byte of the product
   MOV A, H
   ADC A ; Add the carry from the low byte addition
   DAA ; Adjust the high byte to BCD
   MOV H, A ; Store the high byte of the product

   DCR C ; Decrement the multiplier
   JNZ LOOP ; Jump back to the loop if the multiplier is not zero

LXI H, 2300H ; HL points to the low byte of the result
MOV M, D ; Store the low byte of the product

INX H
MOV M, H ; Store the high byte of the product

HLT
```

c. Divide the 16 bit number stored in memory locations 2200H and 2201H by the 8 bit number stored at memory location 2202H. Store the quotient in memory locations 2300H and 2301H and remainder in memory locations 2302H and 2303H

```
; Initialize registers
LXI H, 2200H ; HL points to the 16-bit dividend
MOV B, M ; Load the low byte of the dividend into B
INX H
MOV C, M ; Load the high byte of the dividend into C
```

```
LXI H, 2202H ; HL points to the 8-bit divisor
MOV D, M ; Load the divisor into D

MVI A, 00H ; Initialize the quotient to 0
MVI E, 00H ; Initialize the remainder to 0

; Division loop
DIVLOOP:
   MOV A, E ; Move the current remainder to the accumulator
   MOV H, C ; Move the high byte of the dividend to H
   SUB D ; Subtract the divisor from the high byte of the remainder
   DAA ; Adjust the result to BCD
   MOV C, A ; Store the result back to the high byte of the remainder

   MOV A, B ; Move the low byte of the dividend to the accumulator
    SBB D ; Subtract the divisor from the low byte of the remainder, considering
the borrow
   DAA ; Adjust the result to BCD
   MOV B, A ; Store the result back to the low byte of the remainder

       JC NOT_SUBTRACTED ; If subtraction was successful, increment the
quotient
   INR A ; Increment the quotient
   NOT_SUBTRACTED:

   DCR C ; Decrement the high byte of the remainder
   JNZ DIVLOOP ; Continue the loop if the high byte of the remainder is not zero

; Store the quotient and remainder
LXI H, 2300H ; HL points to the quotient
MOV M, A ; Store the low byte of the quotient
INX H
MOV M, E ; Store the high byte of the quotient

LXI H, 2302H ; HL points to the remainder
MOV M, B ; Store the low byte of the remainder
```

```
INX H
MOV M, C ; Store the high byte of the remainder

HLT
```

d. Find the number of negative elements (most significant bit 1) in a block of data. The length of the block is in memory location 2200H and the block itself begins in memory location 2201H. Store the number of negative elements in memory location 2300H

e. Find the largest number in a block of data. The length of the block is in memory location 2200H and the block itself starts from memory location 2201H. Store the maximum number in memory location 2300H. Assume that the numbers in the block are all 8 bit unsigned binary numbers.

```
LXI H, 2200H ; HL points to the length of the block
MOV B, M ; Load the length into register B

LXI H, 2201H ; HL points to the first number in the block
MOV A, M ; Load the first number into the accumulator
MOV D, A ; Initialize the maximum number to the first number

LOOP:
    INX H ; Point to the next number
    MOV A, M ; Load the next number into the accumulator
    CMP D ; Compare with the current maximum
    JC SKIP ; If the current number is smaller, skip
    MOV D, A ; Update the maximum number
SKIP:
    DCR B ; Decrement the counter
    JNZ LOOP ; Continue the loop if the counter is not zero

LXI H, 2300H ; HL points to the memory location to store the maximum
MOV M, D ; Store the maximum number
```

HLT

Practical 6. Calculations with respect to memory locations.

a. Write a program to sort given 10 numbers from memory location 2200H in the ascending order.

```
; Initialize registers
LXI H, 2200H ; HL points to the first number
MVI B, 09H ; Counter for outer loop (9 passes)
MVI C, 09H ; Counter for inner loop (9 comparisons per pass)

OUTER_LOOP:
   MVI D, 00H ; Flag to check if any swap occurred
   LXI H, 2200H ; Reset HL to the beginning of the array

INNER_LOOP:
   MOV A, M ; Load the first number
   INX H
   CMP M ; Compare with the next number
   JC SKIP_SWAP ; If A >= M, no need to swap

   ; Swap the numbers
   MOV E, M
   MOV M, A
   INX H
   MOV M, E

   ; Set the swap flag
   MVI D, 01H

SKIP_SWAP:
   DCR C
   JNZ INNER_LOOP

   DCR B
   JNZ OUTER_LOOP

HLT
```

b. Calculate the sum of a series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be an 8 bit number so you can ignore carries and store the sum at memory location.

```
; Initialize registers
LXI H, 2200H ; HL points to the length of the list
MOV B, M ; Load the length into register B

LXI H, 2201H ; HL points to the first number in the list
MVI A, 00H ; Initialize the sum to 0

LOOP:
   MOV A, M ; Load the current number into the accumulator
   ANI 01H ; Check if the number is even (LSB is 0)
   JZ EVEN_NUMBER ; If LSB is 0, it's even

   ; If the number is odd, skip to the next number
   INX H
   DCR B
   JNZ LOOP

EVEN_NUMBER:
   ADD M ; Add the even number to the sum
   INX H
   DCR B
   JNZ LOOP

; Store the sum at memory location 2300H
LXI H, 2300H
MOV M, A
HLT
```

c. Calculate the sum of a series of odd numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself

begins from memory location 2201H. Assume the sum to be 16-bit. Store the sum at memory locations 2300H and 2301H.

```
; Initialize registers
LXI H, 2200H ; HL points to the length of the list
MOV B, M ; Load the length into register B

LXI H, 2201H ; HL points to the first number in the list
MVI A, 00H ; Initialize the low byte of the sum to 0
MVI D, 00H ; Initialize the high byte of the sum to 0

LOOP:
   MOV A, M ; Load the current number into the accumulator
   ANI 01H ; Check if the number is odd (LSB is 1)
   JZ EVEN_NUMBER ; If LSB is 0, it's even

   ; If the number is odd, add it to the sum
   ADD A ; Add the odd number to the low byte of the sum
   DAA ; Adjust the result to BCD
   MOV D, A ; Store the low byte of the sum

   INX H ; Point to the high byte of the sum
   MOV A, D ; Load the high byte of the sum
   ADC A ; Add the carry from the low byte addition
   DAA ; Adjust the result to BCD
   MOV D, A ; Store the high byte of the sum

EVEN_NUMBER:
   INX H ; Point to the next number
   DCR B ; Decrement the counter
   JNZ LOOP

; Store the sum at memory locations 2300H and 2301H
LXI H, 2300H
MOV M, D ; Store the low byte of the sum
INX H
MOV M, D ; Store the high byte of the sum
```

```
HLT
```

d. Find the square of the given numbers from memory location 6100H and store the result from memory location 7000H

```
; Initialize registers
LXI H, 6100H ; HL points to the number to be squared
MOV B, M ; Load the number into register B

MVI A, 00H ; Initialize the square to 0
MVI C, B ; Copy the number to another register for loop counter

LOOP:
   ADD B ; Add the number to the square
   DCR C ; Decrement the counter
   JNZ LOOP

LXI H, 7000H ; HL points to the memory location to store the square
MOV M, A ; Store the low byte of the square
INX H
MOV M, H ; Store the high byte of the square

HLT
```

e. Search the given byte in the list of 50 numbers stored in the consecutive memory locations and store the address of memory location in the memory locations 2200H and 2201H. Assume the byte is in the C register and the starting address of the list is 2000H. If byte is not found, store 00 at 2200H and 2201H

```
; Initialize registers
LXI H, 2000H ; HL points to the start of the list
MVI B, 50H ; Counter for the loop
```

```
SEARCH_LOOP:
    MOV A, M ; Load the current number into the accumulator
    CMP C ; Compare with the byte in register C
    JZ FOUND ; If equal, jump to FOUND

    INX H ; Increment HL to point to the next number
    DCR B ; Decrement the counter
    JNZ SEARCH_LOOP

; If not found, store 00 in 2200H and 2201H
LXI H, 2200H
MVI M, 00H
INX H
MVI M, 00H
HLT

FOUND:
    ; Store the address of the found byte
    LXI H, 2200H
    MOV M, H
    INX H
    MOV M, L
    HLT
```

f. Two decimal numbers six digits each, are stored in BCD package form. Each number occupies a sequence of bytes in the memory. The starting address of the first number is 6000H. Write an assembly language program that adds these two numbers and stores the sum in the same format starting from memory location 6200H

```
; Initialize pointers
LXI H, 6000H ; HL points to the first number
LXI D, 6200H ; DE points to the result

; Add the digits, starting from the least significant digit
ADD_LOOP:
    MOV A, M ; Load the current digit from the first number
    ADD M ; Add the corresponding digit from the second number
    DAA ; Adjust the result to BCD
    MOV M, A ; Store the result in the corresponding location in the result

    INX H ; Increment both pointers
    INX D

    DCR H ; Decrement the HL pair to point to the next digit of the first number
    DCR D ; Decrement the DE pair to point to the next digit of the result

    MOV A, H ; Check if we've reached the end of the numbers
    ORA L
    JNZ ADD_LOOP

HLT
```

g. Add 2 arrays having ten 8-bit numbers each and generate a third array of results. It is necessary to add the first element of array 1 with the first element of array-2 and so on. The starting addresses of array l, array2 and array3 are 2200H, 2300H and 2400H, respectively.

```
; Initialize registers
LXI H, 2200H ; HL points to the first element of array 1
LXI D, 2300H ; DE points to the first element of array 2
LXI B, 0AH ; Counter for 10 iterations

; Loop to add corresponding elements
ADD_LOOP:
    MOV A, M ; Load the current element from array 1
    ADD M ; Add the corresponding element from array 2
    DAA ; Adjust the result to BCD (optional, if needed)
    MOV M, A ; Store the result in the corresponding element of array 3

    INX H ; Increment HL to point to the next element of array 1
    INX D ; Increment DE to point to the next element of array 2
    INX H ; Increment HL to point to the next element of array 3

    DCR B ; Decrement the counter
    JNZ ADD_LOOP

HLT
```

Practical 7. Assembly programs on memory locations.

a. Write an assembly language program to separate even numbers from the given list of 50 numbers and store them in another list starting from 2300H. Assume the starting address of the 50 number list is 2200H

```
; Initialize registers
LXI H, 2200H ; HL points to the first number in the original list
LXI D, 2300H ; DE points to the first location in the new list
MVI B, 50H ; Counter for the loop

; Loop through the list
LOOP:
    MOV A, M ; Load the current number into the accumulator
    ANI 01H ; Check if the number is even (LSB is 0)
    JZ EVEN_NUMBER ; If even, store it in the new list

    ; If odd, increment HL to point to the next number
    INX H
    DCR B ; Decrement the counter
    JNZ LOOP

EVEN_NUMBER:
    MOV M, A ; Store the even number in the new list
    INX H ; Increment HL to point to the next number in the original list
    INX D ; Increment DE to point to the next location in the new list
    DCR B ; Decrement the counter
    JNZ LOOP

HLT
```

b. Write assembly language program with proper comments for the following:

- **c. A block of data consisting of 256 bytes is stored in memory starting at 3000H. This block is to be shifted (relocated) in memory**

from 3050H onwards. Do not shift the block or part of the block anywhere else in the memory.

```
; Initialize registers
LXI H, 3000H ; HL points to the source block
LXI D, 3050H ; DE points to the destination block
MVI B, 100H ; Counter for 100 iterations (256 bytes = 100 words)

; Loop to copy each byte from source to destination
COPY_LOOP:
   MOV A, M ; Load the byte from the source
   MOV M, A ; Store the byte at the destination
   INX H ; Increment HL to point to the next source byte
   INX D ; Increment DE to point to the next destination byte
   DCR B ; Decrement the counter
   JNZ COPY_LOOP

HLT
```

- **d. Add even parity to a string of 7-bit ASCII characters. The length of the string is in memory location 2040H and the string itself begins in memory location 2041H. Place even parity in the most significant bit of each character.**

```
; Initialize registers
LXI H, 2040H ; HL points to the length of the string
MOV B, M ; Load the length into register B

LXI H, 2041H ; HL points to the first character of the string

LOOP:
   MOV A, M ; Load the current character into the accumulator
   RLC ; Rotate left through carry once to shift bits left
   RRC ; Rotate right through carry 7 times to count 1s
   RRC
   RRC
```

```
        RRC
        RRC
        RRC
        RRC
        JNC ODD_PARITY ; If carry is 0, parity is odd

        ; Even parity, set the most significant bit
        ORA 80H

    EVEN_PARITY:
        MOV M, A ; Store the character with parity bit
        INX H ; Increment HL to point to the next character
        DCR B ; Decrement the counter
        JNZ LOOP

    HLT
```

- **e. A list of 50 numbers is stored in memory, starting at 6000H. Find the number of negative, zero and positive numbers from this list and store these results in memory locations 7000H, 7001H, and 7002H respectively**

```
; Initialize registers
LXI H, 6000H ; HL points to the first number
MVI B, 50H ; Counter for 50 iterations
MVI C, 00H ; Initialize the negative count
MVI D, 00H ; Initialize the zero count
MVI E, 00H ; Initialize the positive count

; Loop through the list
LOOP:
    MOV A, M ; Load the current number into the accumulator
    ORA A ; Set the zero flag if the number is zero
    JZ ZERO_COUNT

    ; Check the sign flag to determine the number's sign
    JP POSITIVE_COUNT ; If sign flag is set, the number is positive
    INR C ; Increment the negative count

POSITIVE_COUNT:
    INR E ; Increment the positive count

    INX H ; Increment HL to point to the next number
    DCR B ; Decrement the counter
    JNZ LOOP

; Store the counts in memory
LXI H, 7000H
MOV M, C ; Store the negative count
INX H
MOV M, D ; Store the zero count
INX H
MOV M, E ; Store the positive count

HLT
```

- **f. Write an assembly language program to generate Fibonacci numbers.**

```
; Initialize registers
MVI A, 00H ; First Fibonacci number
MVI B, 01H ; Second Fibonacci number
MVI C, 10H ; Number of Fibonacci numbers to generate

LOOP:
    ADD B ; Add the second number to the first
    MOV D, A ; Store the sum in D
    MOV A, B ; Move the second number to the accumulator
    MOV B, D ; Move the sum to the second number

    ; Store the current Fibonacci number in memory (adjust the address as
needed)
```

```
   LXI H, 2000H ; Example: Store the first 10 Fibonacci numbers starting
from 2000H
  MOV M, B
  INX H

  DCR C ; Decrement the counter
  JNZ LOOP

HLT
```

- **g. Program to calculate the factorial of a number between 0 to 8.**

```
; Initialize registers
MVI A, 05H ; Number to calculate factorial (adjust as needed)
MVI B, 01H ; Initialize the factorial to 1

LOOP:
  MUL B ; Multiply the factorial by the current number
  DCR A ; Decrement the number
  JNZ LOOP

HLT
```

Practical 8. String operations in assembly programs.

a. Write an 8085 assembly language program to insert a string of four characters from the tenth location in the given array of 50 characters.

```
; Initialize registers
LXI H, 2000H ; HL points to the start of the 50-character array
MVI B, 48H ; Counter for the loop (50 - 4 = 46 iterations)

; Shift the existing characters to make space for the new string
SHIFT_LOOP:
    INX H ; Point to the next character
    MOV A, M ; Load the character
    INX H ; Point to the next destination location
    MOV M, A ; Store the character at the new location
    DCR B ; Decrement the counter
    JNZ SHIFT_LOOP

; Insert the new string
LXI H, 2009H ; HL points to the 10th location in the array
LXI D, 2040H ; DE points to the new string (starting address)

INSERT_LOOP:
    MOV A, M ; Load the character from the new string
    MOV M, A ; Store the character at the 10th location
    INX H
    INX D
    DCR B ; Decrement the counter (4 iterations for 4 characters)
    JNZ INSERT_LOOP

HLT
```

b. Write an 8085 assembly language program to delete a string of 4 characters from the tenth location in the given array of 50 characters.

```
; Initialize registers
```

```
LXI H, 2009H ; HL points to the 10th location in the array
MVI B, 44H ; Counter for the loop (50 - 4 = 46 iterations)

; Shift the characters to the left to delete the string
SHIFT_LOOP:
   INX H ; Point to the next character to be shifted
   MOV A, M ; Load the character
   MOV M, A ; Store the character at the current location
   DCR H ; Point to the previous location (where the deleted character was)
   DCR B ; Decrement the counter
   JNZ SHIFT_LOOP

HLT
```

c. Multiply the 8-bit unsigned number in memory location 2200H by the 8-bit unsigned number in memory location 2201H. Store the 8 least significant bits of the result in memory location 2300H and the 8 most significant bits in memory location 2301H.

```
; Initialize registers
LXI H, 2200H ; HL points to the first number
MOV B, M ; Load the first number into B
INX H
MOV C, M ; Load the second number into C

MVI A, 00H ; Initialize the accumulator to 0
MVI D, 00H ; Initialize the carry register to 0

; Multiplication loop
LOOP:
   ADD B ; Add the first number to the accumulator
   DAA ; Adjust for BCD (optional, if needed)
   MOV D, A ; Store the low byte of the result in D

   INX H ; Point to the high byte of the result
   MOV A, H
```

```
    ADC A ; Add the carry from the previous addition
    DAA ; Adjust for BCD (optional, if needed)
    MOV H, A ; Store the high byte of the result

    DCR C ; Decrement the counter
    JNZ LOOP

; Store the result
LXI H, 2300H
MOV M, D ; Store the low byte of the result
INX H
MOV M, H ; Store the high byte of the result

HLT
```

d. Divide the 16-bit unsigned number in memory locations 2200H and 2201H (most significant bits in 2201H) by the B-bit unsigned number in memory location 2300H store the quotient in memory location 2400H and remainder in 2401H.

```
; Initialize registers
LXI H, 2200H ; HL points to the low byte of the dividend
MOV B, M ; Load the low byte of the dividend into B
INX H
MOV C, M ; Load the high byte of the dividend into C

LXI H, 2300H ; HL points to the divisor
MOV D, M ; Load the divisor into D

MVI A, 00H ; Initialize the quotient to 0
MVI E, 00H ; Initialize the remainder to 0

; Division loop
DIV_LOOP:
    MOV A, C ; Move the high byte of the remainder to the accumulator
    SUB D ; Subtract the divisor
    DAA ; Adjust the result to BCD (optional, if needed)
    MOV C, A ; Store the result back to the high byte of the remainder

    MOV A, B ; Move the low byte of the remainder to the accumulator
    SBB D ; Subtract the divisor with borrow
    DAA ; Adjust the result to BCD (optional, if needed)
    MOV B, A ; Store the result back to the low byte of the remainder

    JC NOT_SUBTRACTED ; If subtraction was successful, increment the quotient
    INR A ; Increment the quotient
    NOT_SUBTRACTED:

    DCR C ; Decrement the high byte of the remainder
    JNZ DIV_LOOP ; Continue the loop if the high byte of the remainder is not zero

; Store the quotient and remainder
LXI H, 2400H
MOV M, A ; Store the low byte of the quotient
INX H
MOV M, E ; Store the high byte of the quotient

INX H
MOV M, B ; Store the low byte of the remainder
INX H
MOV M, C ; Store the high byte of the remainder

HLT
```

e. DAA instruction is not present. Write a subroutine which will perform the same task as DAA.

We need to implement a subroutine that performs the same function as the DAA instruction, adjusting a binary number to a BCD number. This is necessary when working with BCD arithmetic, especially when the DAA instruction is not available.

```
; Subroutine to perform DAA-like adjustment
DAA_SUBROUTINE:
   ; Check if the lower nibble is greater than 9
   ANI 0FH
   CMP 0AH
   JC NO_ADJUST_LOW

   ; Adjust the lower nibble
   ADI 06H

NO_ADJUST_LOW:
    ; Check if the higher nibble is greater than 9 or if a carry occurred from the lower nibble
   ANI 0F0H
   RRC ; Rotate right to check the carry flag
   JC ADJUST_HIGH
   CMP 090H
   JNC NO_ADJUST_HIGH

   ; Adjust the higher nibble
   ADI 060H

NO_ADJUST_HIGH:
   RET ; Return from the subroutine
```

To use this subroutine in your main program:

```
; ... (Your main program code)

; ... (Part of your code where you want to use DAA)
ADD B ; Add two numbers
CALL DAA_SUBROUTINE ; Call the DAA subroutine to adjust the result
; ... (Rest of your code)
```

Practical 9. Calculations on memory locations.

a. To test RAM by writing '1' and reading it back and later writing '0' (zero) and reading it back. RAM addresses to be checked are 40FFH to 40FFH. In case of any error, it is indicated by writing 01H at port 10.

```
; Initialize the RAM address
LXI H, 40FFH

; Write '1' to the RAM location
MVI A, 01H
MOV M, A

; Read the value and check if it's '1'
MOV A, M
CPI 01H
JNZ ERROR

; Write '0' to the RAM location
MVI A, 00H
MOV M, A

; Read the value and check if it's '0'
MOV A, M
CPI 00H
JNZ ERROR

; If no errors, exit normally
HLT

; Error handling
ERROR:
   MVI A, 01H
   OUT 0AH ; Write 01H to port 10 to indicate error
   HLT
```

b. Arrange an array of 8 bit unsigned no in descending order.

```
; Initialize registers
LXI H, 2000H ; HL points to the start of the array
MVI B, 07H ; Counter for outer loop (7 passes)
MVI C, 07H ; Counter for inner loop (7 comparisons per pass)

OUTER_LOOP:
   MVI D, 00H ; Flag to check if any swap occurred
   LXI H, 2000H ; Reset HL to the beginning of the array

INNER_LOOP:
   MOV A, M ; Load the first number
   INX H
   CMP M ; Compare with the next number
   JC SKIP_SWAP ; If A >= M, no need to swap

   ; Swap the numbers
   MOV E, M
   MOV M, A
   INX H
   MOV M, E

   ; Set the swap flag
   MVI D, 01H

SKIP_SWAP:
   DCR C
   JNZ INNER_LOOP

   DCR B
   JNZ OUTER_LOOP

HLT
```

c. Transfer ten bytes of data from one memory to another memory block. Source memory block starts from memory location 2200H whereas destination memory block starts from memory location 2300H.

```
; Initialize registers
LXI H, 2200H ; HL points to the source block
LXI D, 2300H ; DE points to the destination block
MVI B, 10H ; Counter for 10 iterations

; Loop to copy each byte from source to destination
COPY_LOOP:
   MOV A, M ; Load the byte from the source
   MOV M, A ; Store the byte at the destination
   INX H ; Increment HL to point to the next source byte
   INX D ; Increment DE to point to the next destination byte
   DCR B ; Decrement the counter
   JNZ COPY_LOOP

HLT
```

d. Write a program to find the Square Root of an 8 bit binary number. The binary number is stored in memory location 4200H and the square root in 4201H.

```
; Initialize registers
LXI H, 4200H ; HL points to the number
MOV A, M ; Load the number into the accumulator

MVI B, 0 ; Initialize the square root to 0
MVI C, 01H ; Initialize the increment value

LOOP:
   ADD B ; Add the increment to the square root
   MOV D, B ; Store the new square root in D
   MUL D ; Square the new square root
   CMP A ; Compare the square with the original number
```

```
    JC LOOP ; If the square is less, continue the loop

    ; The square root is in D
    DCR D ; Adjust the square root (optional, depending on the desired precision)
    LXI H, 4201H ; HL points to the memory location to store the result
    MOV M, D ; Store the square root
    HLT
```

e. Write a simple program to Split a HEX data into two nibbles and store it in memory.

```
; Initialize registers
LXI H, 4200H ; HL points to the number
MOV A, M ; Load the number into the accumulator

MVI B, 00H ; Initialize the square root to 0
MVI C, 01H ; Initialize a counter

LOOP:
    ADD B ; Add the current square root to itself
    DCR C ; Decrement the counter
    JNZ LOOP

; Store the square root
LXI H, 4201H
MOV M, B

HLT
```

Practical 10. Operations on BCD numbers.

a. Add two 4 digit BCD numbers in HL and DE register pairs and store the result in memory locations, 2300H and 2301H. Ignore carry after 16-bit.

```
; Add the two BCD numbers
DAD D ; Add DE to HL

; Adjust the result to BCD format
DAA
MOV M, A ; Store the low byte of the result in 2300H
INX H
MOV M, H ; Store the high byte of the result in 2301H

HLT
```

b. Subtract the BCD number stored in E register from the number stored in the D register.

```
; Subtract the BCD numbers
SUB E ; Subtract E from D
DAA ; Adjust the result to BCD format

; Store the result in the accumulator
MOV A, D

HLT
```

c. Write an assembly language program to multiply 2 BCD numbers.

```
; Initialize registers
MVI A, 05H ; First BCD number
MVI B, 03H ; Second BCD number
MVI C, 00H ; Initialize the result to 0
```

```
; Multiplication loop
LOOP:
   ADD A ; Add the first number to the result
   DAA ; Adjust the result to BCD format
   DCR B ; Decrement the second number
   JNZ LOOP

; Store the result (assuming a 2-digit BCD result)
LXI H, 2000H ; HL points to the memory location to store the result
MOV M, A

HLT
```