* STACK AND SUBROUTINES: (12)

DEFINITIONS:

The stack is a group of memory locations in the R/W memory that is used for temporary storage of binary information during the execution of a program.

Subroutine is a group of instructions that performs a subtask (eg:- time delay or arithmetic operation) of repeated occurrence.

A large software project is usually divided into subtasks called modules.

STACK:

The stack in an 8085 microcomputer system can be described as a set of memory locations in the R/W memory, specified by a programmer in a main program. These memory locations are used to store binary information (bytes) temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP, which loads a 16-bit memory address in the stack pointer register of the microprocessor. Once the stack location is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register. For example, if the stack pointer register is loaded with the memory address 2099 H (LXI SP, 2099 H), the storing of data bytes begins at 2098 H and continues in reversed numerical order (decreasing memory addresses such as 2098 H, 2097 H, etc.). The size of the stack is limited only by the available memory.

Data bytes in the register pairs of the microprocessor can be stored on the stack (two at a time) in reverse order (decreasing memory address) by

using the Instruction PUSH. Data bytes can be transferred from the stack to respective registers by using the Instruction pop.

The programmer can store and retrieve the contents of a register pair by using PUSH and POP Instructions. The instructions necessary for using the stack are explained below.

INSTRUCTIONS

OPCODE	OPERAND	
LXI	SP, 16-bit	; Load stack pointer ; Load the stack pointer register with a ; 16-bit address.
PUSH	RP	; Store Register pair on stack ; This is a 1-byte instruction. ; It copies the contents of the specified ; register pair on the stack as described ; below. ; The stack pointer register is decremented ; and the contents of the high-order ; register (eg., register B) are copied to ; the location shown by the stack pointer ; register. ; The stack pointer register is again ; decremented, and the contents of the ; low-order register (eg; register C) ; copied in the location. ; The operands B, D and H represent ; register pairs BC, DE & HL respectively ; The operand PSW represents program ; status word, meaning the contents ; of the accumulator and the flag
PUSH	B	
PUSH	D	
PUSH	H	
PUSH	PSW	

OPCODE	OPERAND	
POP	Rp	; Retrieve Register pair from Stack.
		; This is a 1-byte instruction.
POP	B	; It copies the contents of the top two memory
POP	D	; locations of the stack into the specified
POP	H	; register pair.
POP	PSW	; First, the contents of the memory location
		; indicated by the stack pointer register
		; are copied into the low-order register
		; (eg., register L), and then the stack
		; pointer register is incremented by 1.
		; The contents of the next memory location are
		; copied into the high-order register
		; (eg. register H), and the stack pointer
		; register is again incremented by 1.

All three of these instructions belong to the data transfer (copy) group;

SUBROUTINE:

A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program.

The 8085 microprocessor has two instructions to implement subroutines: CALL (call a subroutine), and RET (return to main program from a subroutine). The CALL instruction is used in the main program to call a subroutine and the RET instruction is used at the end of the subroutine to return to the main program.

POP PSW Program status word.

INSTRUCTIONS :

OPCODE	OPERAND	
CALL	16-bit memory address of a Subroutine	Call subroutine Unconditionally. • ; This is a 3-byte instruction that ; transfers the program sequence to ; subroutine address • ; save the contents of the program counter ; (the address of the next instruction) ; on the stack. • ; Decrements the stack pointer register by ; two • ; Jumps unconditionally to the memory ; location specified by the second and ; third bytes. The second byte specifies ; line number and the third byte ; specifies a page number. • ; This instruction is accompanied by ; return instruction in the subroutine
RET		Return from Subroutine Unconditionally • ; This is a 1-byte instruction. • ; Inserts the two bytes from the top of ; the stack into the program counter and ; increments the stack pointer register by two • ; Unconditionally returns from a subroutine

EXAMPLE:

Illustrate the exchange of information between the stack and the program counter for the following program if the available user memory ranges from 2000H to 23FFH.

MEMORY ADDRESS		
2000	LXI SP, 2400 H	; Initialize the stack pointer at
		; 2400H.
2040	CALL 2070 H	; Call the subroutine located at 2070H
2041		; This is a 3-byte instruction.
2042		
2043	NEXT INSTRUCTION	; The address of the next instruction
		; following the CALL instruction.
205F	HLT	; End of the main program
2070	First subroutine Instruction	; Begining of the subroutine
207F	RET	; End of the subroutine
2080	other subroutines	
2898	Empty Space	
28FF		
2400		; The stack is initialized at
		; 2400 H.

Solution After reviewing the above program note the following points:

1. The available user memory is from 2000H to 28FFH (1024 to 1K bytes); however the stack pointer register is initialized at 2400H, one location beyond the user memory. This allows maximum use of the memory because the actual stack begins at 28FFH. The stack can expand up to the location 2898H

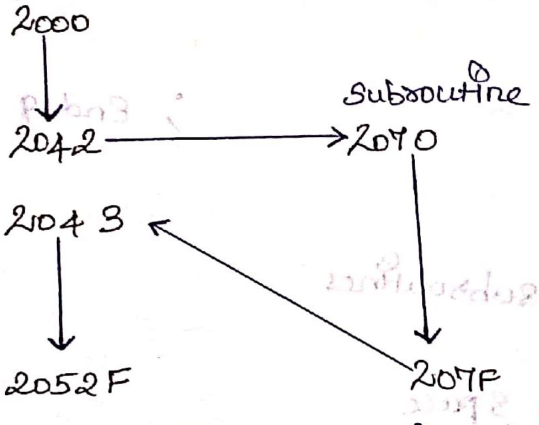
without overlapping with the program.

2. The main program is stored at memory locations from 2000H to 205FH.
3. The CALL instruction is located at 2040H to 2042H (3-byte instruction). The next instruction is at 2043H.
4. The subroutine begins at the address 2070H and ends at 207FH.

PROGRAM EXECUTION:

The sequence of the program execution and the events in the execution of the CALL and subroutine are shown.

SUBROUTINE CALL AND PROGRAM TRANSFER



The program execution begins at 2000_H, continues until the end of CALL 2042H, and transfers to the subroutine at 2070H. At the end of the subroutine, after executing the RET instruction, it comes back to the main program at 2043H and continues.

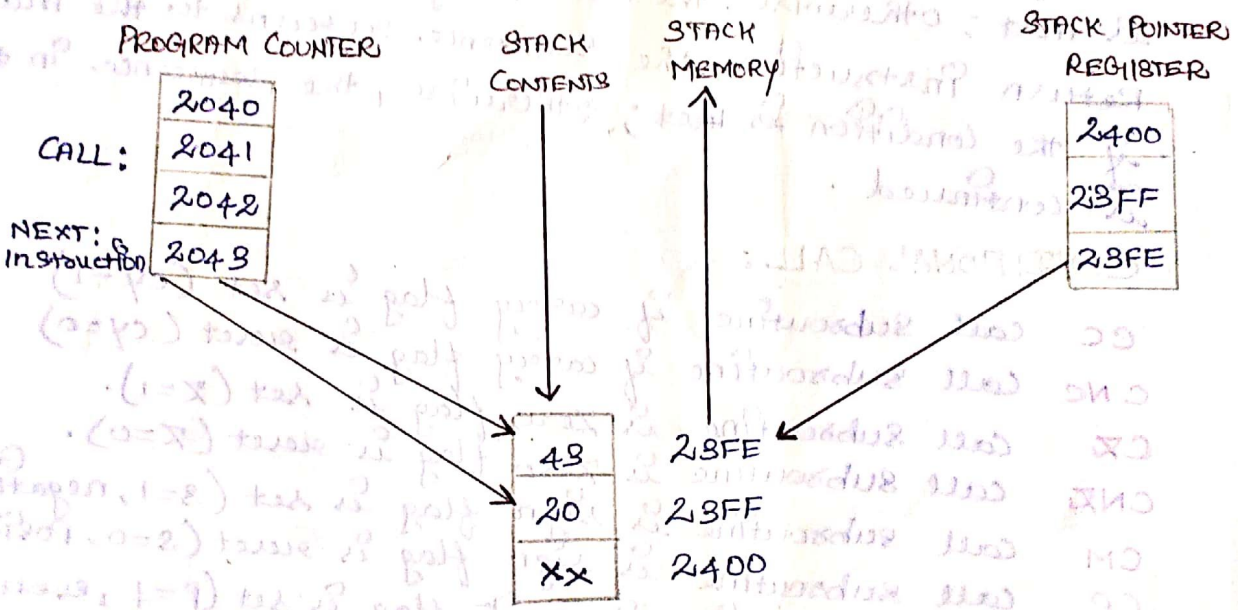
The instruction requires five machine cycles and eighteen T-states. The sequence of events in each machine cycle is as follows.

1. M_1 — opcode Fetch
2. M_2 and M_3 — Memory Read
3. M_4 and M_5 — Storing of program counter.
4. Next Instruction cycle.

RET EXECUTION:

At the end of the subroutine, when the instruction RET is executed, the program execution sequence is transferred to the memory location 2043H. The address 2043H was stored in the top locations of the stack (23FEH and 23FFH) during the CALL instruction.

Contents of the program counter, the stack pointer and the stack during the Execution of the CALL instruction



RESTART, CONDITIONAL CALL, AND RETURN INSTRUCTION:

In addition to the unconditional CALL and RET instructions, the 8085 instruction set includes eight Restart instructions and eight conditional Call and Return instructions.

RESTART (RST) INSTRUCTIONS:

RST instructions are 1-byte call instructions that transfer the program execution to a specific location on page 00H. The list of eight RST instructions is as follows:

RST 0	Call 0000 H	RST 4	Call 0020 H
RST 1	Call 0008 H	RST 5	Call 0028 H
RST 2	Call 0010 H	RST 6	Call 0030 H
RST 3	Call 0018 H	RST 7	Call 0038 H.

CONDITIONAL CALL AND RETURN INSTRUCTIONS:

The conditional call and Return instructions are based on four data conditions (flags): carry, zero, sign and parity. The conditions are tested by checking the respective flags. In case of a conditional call instruction, the program is transfered to the subroutine if the condition is met; otherwise, the main program is continued. Conditional Return instruction, the sequence returns to the main program if the condition is met; otherwise, the sequence in the subroutine is continued.

CONDITIONAL CALL:

CC	Call subroutine if carry flag is set (CY=1)
CNC	Call subroutine if carry flag is reset (CY=0)
CZ	Call subroutine if zero flag is set (Z=1).
CNZ	Call subroutine if zero flag is reset (Z=0).
CM	Call subroutine if sign flag is set (S=1, negative number)
CP	Call subroutine if sign flag is reset (S=0, positive number)
CPE	Call subroutine if parity flag is set (P=1, even parity)
CPO	Call subroutine if parity flag is reset (P=0, odd parity)

CONDITIONAL RETURN:

RC	Return if carry flag is set (CY=1)
RNC	Return if carry flag is reset (CY=0)
RZ	Return if zero flag is set (Z=1)
RNZ	Return if zero flag is reset (Z=0)
RM	Return if sign flag is set (S=1, negative number)
RP	Return if sign flag is reset (S=0, positive number)
RPE	Return if parity flag is set (P=1, even parity)
RPO	Return if parity flag is reset (P=0, odd parity).

UNIT - IV

P. Sowanya

* MULTI BYTE ADDITION : (16-bit, 32-bit, and 64-bit addition)

ALGORITHM :

Step 1 : Clear the accumulator and carry flag.

Step 2 : Store the number of bytes to be added in C register. C register is the counter.

Step 3 : Initialize HL register with memory address where the Latest Significant Byte of first data is stored.

Step 4 : Initialize DE register with memory address where the Least Significant Byte of second data is stored.

Step 5 : Load a byte from memory whose address is in DE to accumulator.

Step 6 : Add with carry, the accumulator content with a byte from memory whose address is in HL pair.

Step 7 : Store the partial sum in memory pointed to by HL pair

Step 8 : Increment address in HL to point to next byte of first data.

Step 9 : Increment address in DE to point to next byte of Second data.

Step 10 : Decrement the count in C register by one.

Step 11 : If the count in C register is not zero, go to step 5.

Step 12 : End.

PROGRAM :-

```
    XRA A          ; Clear the accumulator and carry flag.
    MVI C, 02H     ; Load the C register with the number
                   ; of bytes. C register is the counter.
    LXI H, 2050H   ; Load the HL register pair with memory
                   ; address where the first data is
                   ; Stored.
    LXI D, 2060H   ; Load the DE register pair with memory
                   ; address where the second data is
                   ; Stored.
HERE: LDAX D       ; Take the lower byte from the memory
                   ; address 2060H into the accumulator.
    ADC M          ; Add with carry, the content of the
                   ; accumulator with the byte in the
                   ; memory address 2050H.
    MOV M, A       ; Store the result in the memory whose
                   ; address is 2050H.
    INX H          ; Increment the address in HL register by
    INX D          ; Increment the address in DE register.
    DCR C          ; Decrement the C register content by one.
    JNZ HERE       ; if the content of C register is not zero
                   ; then jump to the location named HERE.
    HLT            ; End of program.
```

EXAMPLE:

Input data 1: 12 34 → Addend
Input data 2: AB CD → Augend
Sum : BE 01

```
  0001 0010 0011 0100 = 1234
+ 1010 1011 1100 1101 = ABCD
-------------------------------
  1011 1110 0000 0001
   B    E    0    1
```

Input 1 : 2050 24
2051 12

Input 2 2060 CD
2061 AB

Sum : 2050 01
2051 BE

*** MULTI BYTE SUBTRACTION:** (16-bit, 32-bit and 64-bit subtraction)

ALGORITHM:

Step 1 : Clear the accumulator and carry flag.

Step 2 : Store the number of bytes to be added in C register.
C register is the counter.

Step 3 : Initialize HL register with memory address where first data is stored.

Step 4 : Initialize DE register with memory address where second data is stored.

Step 5 : Load a byte from memory whose address is in DE to accumulator.

Step 6 : Subtract with borrow, the accumulator content with a byte from memory whose address is in HL pair.

Step 7 : Store the result in memory.

Step 8 : Increment address in HL to point to next byte of fir data.

Step 9 : Increment address in DE to point to next byte of second data.

Step 10 : Decrement the count in C register by one.

Step 11 : If the count in C register is not zero, go to step 6

Step 12 : End.

PROGRAM:

```
XRA A       ; clear the accumulator and carry flag.
MVI C,02H   ; Load the C register with the number of bytes.
            ; C register is the counter.
LXI H,2050H ; Load the HL register pair with memory
            ; address where the first data is stored.
LXI D,2060H ; Load the DE register pair with memory
            ; address where the second data is stored.
HERE: LDAX D ; Take the lower byte from the memory address
            ; 2060H into the accumulator.
   SBB M    ; Sub with borrow, the byte in the memory
            ; address 2050 from the content of accumulator.
   MOV M,A  ; Store the result in the memory whose
            ; address is 2050 H
   INX H    ; Increment the address in HL register by one
   INX D    ; Increment the address in DE register by one
   DCR C    ; Decrement the C register content by one
JNZ HERE    ; If the content of C register is not zero, then
            ; jump to the location named HERE.
   HLT      ; End of program.
```

EXAMPLE: (57DD from ABCD)

Input data 1 : 57 DD → Subtrahend

Input data 2 : AB CD → Minuend

Difference : 53 F0

[0-0 = 0
0-1 = 11 (difference with a borrow 1.
1-0 = 0
1-1 = 0].

Input 1 2050 DD Input 2 2060 CD
2052 57 2061 AB

Difference 2050 F0
2052 53

(57DD = 0101 0111 1101 1101
ABCD = 1010 1011 1100 1101).

(ie) 57DD 1010 1011 1100 1101
(-)ABCD 0101 0111 1101 1101
0101 0011 1111 0000
⇒ 53F0).

* BCD ADDITION : (Result < 99)

ALGORITHM:

Step 1 : Take the addend in accumulator from memory.

Step 2 : Transfer the addend to B register.

Step 3 : Take the augend in accumulator from memory.

Step 4 : Add the two numbers.

Step 5 : Using DAA, convert the result in the accumulator to Decimal (BCD).

Step 6 : Store the sum in memory.

Step 7 : End.

PROGRAM:

```
LDA 2050H ; Get the addend from memory into accumulator.
MOV B,A   ; Transfer the number to B register.
LDA 2051H ; Get the augend from memory into accumulator.
ADD B     ; Add the two numbers.
DAA       ; Convert the Hex result in accumulator to decimal
STA 2052H ; Store the result in memory.
HLT       ; End of program.
```

EXAMPLE:

Input data : 2050 : 25
2051 : 56

Sum : 2052 : 81

25 = 0010 0101
56 = 0101 0110
0111 1011 = 7B.
↳ Low order greater than 9 : add 0110.

0111 1011
(+) 0000 0110
1000 0001 = 81)

*** BCD SUBTRACTION:**

ALGORITHM:

Step 1 : Take the minuend in accumulator from memory.

Step 2 : Transfer the minuend to B register.

Step 3 : Take the subtrahend in accumulator from memory.

Step 4 : Transfer the subtrahend to C register.

Step 5 : Load 99 in accumulator to find the 9's complement of the subtrahend.

Step 6 : Add one to get 10's complement of the subtrahend.

Step 7 : Add the minuend to the 10's complement of the subtrahend.

Step 8 : Using DAA, convert the number in accumulator to decimal (BCD).

Step 9 : Store the result available in accumulator in memory.

Step 10 : End.

PROGRAM:

```
LDA 2050H ; Take the minuend in accumulator.
MOV B,A   ; Transfer the minuend to B register.
LDA 2051H ; Take the subtrahend in accumulator.
```

```
MOV C,A ; Transfer the Subtrahend to C register.
MVI A,99 ; Load 99 in accumulator.
SUB C ; Find the 9's complement of the Subtrahend..
INR A ; Increment the accumulator content by one to find
      ; the 10's complement of the Subtrahend
ADD B ; Add the minuend with the 10's complement of
      ; the Subtrahend.
DAA ; Convert the result to decimal
STA 2052H ; Store the result in memory.
HLT ; End of program
```

EXAMPLE:

Input data : 2050 : 85
2051 : 39
2052 : 46

(85 → Minuend, 39 → Subtrahend)

1000 0101 = 85
(-) 0011 1001 = 39
0111 1100 → It is wrong.

So make it as 9's comp

9's → 99 − 39 = 60
(+) 10's → 60 + 1 = 61

85
+ 61
146

* BCD MULTIPLICATION:

ALGORITHM:

Step 1: Load the first value to the accumulator.
Step 2: Move the accumulator contents to B register.
Step 3: Load the second value to the accumulator.
Step 4: Move accumulator contents to C register.
Step 5: perform Exclusive – OR with Accumulator.

Step 6 : Add B with accumulator

Step 7 : perform binary to BCD.

Step 8 : Decrement C register by 1.

Step 9 : if Z = 0 then go to step 6.

Step 10 : Store accumulator content in the memory.

Step 11 : End.

PROGRAM :

```
LDA 8050H   ; Get 1st value in accumulator.
MOV B, A    ; Move A to B register. get 2nd value in
            ; ← Accumulator.
LDA 8051H
MOV C, A    ; Move A to C register.
XRA A       ; Exclusive-OR with A.
ADD B       ; Add B with Accumulator.
DAA         ; Change Binary to BCD.
DCR C       ; Decrement C Register.
JNC 8009    ; Jump on non Zero.
STA 8052H   ; Start accumulator contents
HLT         ; Stop.
```

[0X0 = 0
0X1 = 0
1X0 = 0
1X1 = 1]

Example:

Input 8050 : 02
8051 : 10

Output 8052 : 14

→ Multiplicand
02 X 10 ← Multiplier

0000 0010 = 02
0001 0000 = 10

XRA ← 0001 0010
0000 0010 (+) = 02 (multiplicand)
0001 0100 = 14.

✱ BCD DIVISION :-

ALGORITHM :-

Step 1 : Load the dividend value in the accumulator.
Step 2 : Move Accumulator value to B register.
Step 3 : Load the divisor value in the accumulator.
Step 4 : Move FF to C Register
Step 5 : Increment C Register by 1
Step 6 : Subtract B from A.
Step 7 : If Cy = 0 then goto Step 5.
Step 8 : Add B register value to Accumulator.
Step 9 : Store Accumulator content in the memory spac
Step 10 : Move C Register value to Accumulator.
Step 11 : Store Accumulator value in the memory spac
Step 12 : end.

PROGRAM :-

```
LDA  8051 H ; Get 1st value on Accumulator
MOV  B, A   ; Move A to B register.
LDA  8050 H ; Get 2nd value in Accumulator
MVI  C, FFH ; Move FF to C register.
8009: INR  C   ; Increment C register
SUB  B   ; Subtract B from A.
JNC  8009 ; Jump on non carry.
```

ADD B ; Add B with accumulator.

STA 8052H ; Store accumulator contents

MOV A,C ; Move c to Accumulator.

STA 8053H ; Store Accumulator contents.

HLT ; Stop.

EXAMPLE:

Input 8050 : 06 03
8051 : 03 06

output 8052 : 02
8053 : 00

$[0 \div 1 = 0$
$1 \div 1 = 0]$

03 ÷ 06 — Dividend
→ Divisor

0000 0110 → 06
(→) 0000 0011 → 03
A = 0000 0011 → 03
(→) 0000 0011 → 03
0000 0000 → 00.

02
03 | 06
06
0

A = 03
B = 03

Ex2: Divide 10 by 4.

10
0100 | 1010
100
10

Quotient = 2
Reminder = 2.

BCD to BINARY CONVERSION: (Decimal to Hex) (8-bit)

ALGORITHM:

Step 1: Load the BCD number in B register.

Step 2: Copy it in accumulator.

Step 3: Mask the upper nibble and store the lower nibble in C register (units in C).

Step 4: Again, get the number in B register into the accumulator.

Step 5: Mask the lower nibble and rotate the byte four times to the right to get higher digit.

Step 6: Move it to D register (Tens in D).

Step 7: Multiply the Tens digit in D by 10_{10} or 0AH.

Step 8: Add the units digit to the result.

Step 9: Store the Hex result in memory

Step 10: End.

(Masking = changing
(ie) 98 = 1001 1000
(AND immediate data) ← ANI OF = 0000 1111
0000 1000
upper bit

PROGRAM:

```
LDA 2050H ; Load the BCD number in B register.
MOV B,A   ; Copy it in accumulator (Transfer it to B).
ANI 0FH   ; Masks the upper nibble and get the lower nibble
MOV C,A   ; Store the lower nibble in C register C=08
MOV A,BH  ; Take the number in B into accumulator B=98
ANI F0H   ; masks the lower nibble A=90
RRC       ; Rotate the byte four times to get the
RRC       ; higher digit.
```

A: 1001 1000
1111 0000
1001 0000

RRC

RRC

MOV D,A ; Put the higher digit in D register D = 09

XRA A ; clear the accumulator A = 0

MVI E,0AH ; Load E register with 0AH or 10_{10} E = 10

LOOP: ADD D; Multiply the contents of D and E 09 - A

DCR E ; registers . C.

JNZ LOOP

ADD C ; Add the lower nibble to the content of the accumulator

STA 2051H ; Store the result in memory .

HLT ; End of program .

EXAMPLE:

Input data : 2050 - 98

Result : 2051 - 62H

2 | 98
2 | 49 - 0
2 | 24 - 1
2 | 12 - 0
2 | 6 - 0
2 | 3 - 0
1 - 1

(1100010)

0110 0010

6 2 = 62H)

*** BINARY TO BCD CONVERSION : (Hex to Decimal) (8-bit)**

ALGORITHM :

Step 1 : Initialize memory to store the BCD result.

Step 2 : Load the Hex number in accumulator.

Step 3 : Load $64_H = 100_{10}$ in B register.

Step 4 : Call the division subroutine (DIVN) to divide the Hex number by 100_{10} or 64H.

Step 5 : Load $0A_H = 10_{10}$ in B register.

Step 6 : call division subroutine to divide the remainder by 10 or 0AH

Step 7 : Get the final remainder.

Step 8 : End.

PROGRAM:

```
LDA 2050H  ; Load the Hex number in accumulator.
LXI H,2051H ; Initialize memory to store BCD result.
MVI B,64H  ; Load 64H (or 100₁₀) in B register
CALL DIVN  ; call DIVN subroutine to divide the given number by
           ; 64H
MVI B,0AH  ; Load 0AH (or 10₁₀) in B register
CALL DIVN  ; call DIVN subroutine to divide the given number by
MOV M,A    ; Store the final remainder in memory
HLT        ; End of program.
```

DIVN SUBROUTINE

```
MVI M,FFH  ; Store FFH (or -1) in memory.
LOOP: INR M ; Increment the content of memory by one to
           ; clear the memory.
SUB B      ; (A) <— (A) - (B)
JNC LOOP   ; If the result is positive, continue in Loop.
ADD B      ; Else, get the remainder by adding the contents of A
           ; and B.
INX H      ; Increment the memory address.
RET        ; Return to the main program
```

EXAMPLE:

Input data : 2050 : FE_H Result : 2051 : 02

2052 : 05

2053 : 04

$(FE = 15 \times 16^1 + 14 \times 16^0$

$240 + 14 = 254$ #) (So $\frac{1111}{F=15}\ \frac{1110}{E=14} = \frac{0010}{2}\ \frac{0101}{5}\ \frac{0100}{4}$)

* ASCII to BCD

ALGORITHM:

Step1 : Take the ASCII number (i.e ASCII number from 30 to 39) in Accumulator.

Step2 : Subtract 30H from the accumulator.

Step3 : Store the BCD result in memory.

Step 4 : End.

1111 1110
0110 0100
1001 1010
0110 0100
1111 1110

PROGRAM:

LDA 2050H ; Take the ASCII number in accumulator.

SUI 30H ; Subtract 30H.

STA 2051H ; Store the BCD result in memory.

HLT ; End of program.

(ie) (38 - 30) = 08.

EXAMPLE:

Input data : 2050 : 38

Result : 2051 : 08

0 - 9	30H - 39H
A - Z	41H - 5AH
a - z	61H - 7AH

The Hexa digit 0-9 represented by 30H - 39H. Hence for ASCII code, we subtract 30H, we will get corresponding binary value.

(A-Z) we subtract 37H.

* BCD to ASCII

ALGORITHM:

Step 1 : Take the BCD number (i.e. decimal number from 0 to 9) in accumulator.

Step 2 : Add 30_H with accumulator.

Step 3 : Store the ASCII result in memory.

Step 4 : End.

PROGRAM:

```
LDA 2050H ; Take the BCD number in accumulator
ADI 30H   ; Add 30H to the BCD number
STA 2051H ; Store the ASCII result in memory.
HLT       ; End of program
```

EXAMPLE:

Input data : 2050 : 08 (08 + 30 = 38.)

Result : 2051 : 38.

* BINARY to ASCII:

ALGORITHM:

Step 1 : Load the Binary number (i.e Hex number) in accumulator.

Step 2 : If the given number is from 0 to 9, add 30_H to accumulator and go to Step 4.

Step 3 : If the given number is from $0A_H$ to $0F_H$ add 07_H in addition to 30_H.

Step 4 : Store the ASCII result in memory.

Step 5 : End.

PROGRAM:

```
LDA 2050H ; Take the binary number in accumulator.
CPI 0AH   ; Compare the given number with 0AH.
JC SKIP7  ; If the number is less than 9, no need to add 7.
ADI 07H   ; Else, add 7 to the accumulator.
SKIP7: ADI 30H ; Add 30H to accumulator.
STA 2051H ; Store the ASCII result in memory.
HLT       ; End of program.
```

EXAMPLE:

Input data : 2050 : 0BH

Result : 2051 : 42H

0B = 0000 1011 (Compare with 10, if it is greater so add 07)

0000 1011 → 0B
(+) 0000 0111 → 07
0001 0010 = 12H (Now add 30)

(12 + 30 = 42H)

* ASCII to BINARY:

ALGORITHM:

Step 1: Take the ASCII number in accumulator.

Step 2: Subtract 30H from the accumulator.

Step 3: If the result is from 0 to 9, go to Step 5.

Step 4: If the result is greater than 9, Subtract 7 from accumulator.

Step 5: Store the binary result in memory.

Step 6: End.

PROGRAM:

```
LDA 2050H ; Take the ascii number in accumulator.
SUI 30H   ; Subtract 30H from the accumulator.
CPI 0AH   ; compare result in accumulator with 0AH.
JC SKIP7  ; If the result is less than 0AH, no need to
          ; Subtract 7
SUI 07H   ; Else, Subtract 7.
SKP7: STA 2051H ; Store the binary result in memory.
      HLT       ; End of program.
```

EXAMPLE:

Input data : 2050 : 42H

(42 − 30 = 12)

(compare with 10, it is greater so sub 07)

```
  0001 0010 = 12
(-)0000 0111 = 07
  0000 1011 = 0B
```

Result : 2051 : 0BH.