SYLLABUS

ADDITION AND SUBTRACTION :-

1. 8- bit addition
2. 16 - bit addition
3. 8 - bit Subtraction
4. BCD - Subtraction

MULTIPLICATION AND DIVISION :-

1. 8 - bit Multiplication
2. BCD - Multiplication → need to include o/p in o/p pdf
3. 8 - bit Division.

SORTING AND SEARCHING :-

1. Searching for an element in an array.
2. Sorting is ascending and descending order.
3. Finding the largest and Smallest element in the array
4. Reversing array elements.
5. Block Move.

CODE CONVERSION.

BCD to Hexa and Hexa to BCD. ✓ → Algorithm O/P

Binary to ASCII and ASCII to Binary.

ASCII to BCD and BCD to ASCII.

APPLICATION :-

Square of a single byte hexa number.

Square of two-digit BCD number.

square root of a single byte hexa number.

Square root of a two-digit BCD number.

8-bit addition.

LDA 2050_H → Take the first number to A

MOV B A → Transfer the number from A into B

LDA 2051_H → Take the second number to A

ADD B → Add the two number in A and B

STA 2052_H → Store the result in Memory

HLT → End of Program

16 - bit addition.

LHLD 2050_H → load the HL register pair with the first 16 bit number from memory.

XCHG → Exchange the content of the HL register pair with DE register pair the first 16-bit number now in DE register pair.

LHLD 2052_H → load the HL register pair with the second 16 bit number from memory.

DAD D → Add the content of HL & DE register pair.

SHLD 2054_H → Store the 16 bit result in HL register pair in the memory.

HLT → End of program.

8 - bit Subtraction

```
LDA   2050H
MOV   B A
LDA   2051H
SUB   B
STA   2052H
HLT
```

BCD Subtraction

```
LDA   2050H
MOV   B A
LDA   2051H
MOV   C A
MVI   A 99
SUB   C
INR   A
ADD   B
DAA
STA   2052H
HLT
```

8- bit Multiplication

```
LDA   2050H
MOV   B A
LDA   2051H
MOV   C A
XRA   A
MOV   D A
ADD   C
JNC   Goto
```

8 - bit Addition.

ALGORITHM :-

Step - 1 : Get the first Number from the memory location 2050_H to the accumulator.

Step - 2 : Save the first number in B register.

Step - 3 : Get the second number from the memory location 2051_H to the accumlator

Step - 4 : Add the numbers in A and B.

Step - 5 : Store the result which is in accumlator in the memory location 2052_H

Step - 6 : End.

CODING :-

```
LDA   2050H
MOV   B A
LDA   2051H
ADD   B
STA   2052H
HLT
```

Output :-

Address	Data
2050	05
2051	03
2052	08
2053	00

Verified
2/8/24

16-bit Addition.

ALGORITHM :-

Step - 1 : Take the first 16-bit number from memory to HL register pair.

Step - 2 : The contents of HL is transfered into DE register.

Step - 3 : Take the second 16-bit number from memory to HL register pair.

Step - 4 : Add the two 16-bit numbers using DAD D instruction.

Step - 5 : The 16-bit result in HL register pair is moved to memory.

Step - 6 : End.

CODING :-

```
LHLD   2050H
XCHG
LHLD   2052H
DAD    D
SHLD   2054H
HLT
```

Output :-

Address	Data
2050	03
2051	00
2052	02
2053	00
2054	05

8 - bit Subtraction

ALGORITHM :-

Step - 1 :- Get the first number from memory to the accumulator.

Step - 2 :- Store the first number in B register.

Step - 3 :- Get the second number from memory to the accumulator.

Step - 4 :- Subtract the first number from the second number.

Step - 5 :- Store the difference in the memory

Step - 6 :- End.

CODING :-

```
LDA  2050H
MOV  B A
LDA  2051H
SUB  B
STA  2052H
HLT.
```

Address	Data
2050	
2051	
2052	
2053	

14/3/24.

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 Memory.

Step - 4 :- Transfer the Subtrahend to C register

Step - 5 :- Load 99 in accumulator to find 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 De

Step - 9 :- Store the result available in accumulator in Memory.

Step - 10 :- End.

CODING :-

```
LDA   2050H
MOV   B A
LDA   2051H
MOV   C A
MVI   A 99
SUB   C
INR   A
ADD   B
DAA
STA   2052H
HLT.
```

output :-

Address	Data
2050	20
2051	10
2052	10
2053	00
2054	00

Verified
14/3/24

Multiplication And Division

8-bit Multiplication

ALGORITHM :-

Step - 1 :- Get the Multiplier from the memory to the accumulator.

Step - 2 :- Transfer the Multiplier to B register.

Step - 3 :- Get the Multiplicand from the memory to the accumulator.

Step - 4 :- Transfer the Multiplicand to C register

Step - 5 :- Initialize accumulator to zero.

Step - 6 :- Clear the D register for storing the carry.

Step - 7 :- Add Multiplicand in C register to accumulator.

Step - 8 :- If the value in accumulator exceeds FF_H. increment D register by one

Step - 9 :- Decrement the Multiplier in B by one.

Step - 10 :- If the value in B register is not equal to zero, then go to step 7.

Step -11 :- If the value in B register becomes zero. Store the lower byte of the result in the accumulator to Memory.

Step -12 :- Store the higher byte of the result in D register in Memory

Step -13 :- End

CODING :-

```
        LDA   2050H
        MOV   B A
        LDA   2051H
        MOV   C A
        XRA   A
        MOV   D A
HERE:   ADD   C
        JNC   GOTO
        INR   D
GOTO:   DCR   B
        JNZ   HERE
        STA   2052H
        MOV   A D
        STA   2053H
        HLT
```

Output :-

Address	Data
2050	15
2051	10
2052	50
2053	01
2054	00

8/8/24

BCD - Multiplication

ALGORITHM :-

Step - 1 :- Take the Multiplicand in accumulator from memory.

Step - 2 :- Transfer the Multiplicand to B register.

Step - 3 :- Take the Multiplier in accumulator from Memory.

Step - 4 :- Transfer it to C register.

Step - 5 :- Clear the accumulator.

Step - 6 :- Add the content of B register with that of the accumulator.

Step - 7 :- Use DAA to convert the number in accumulator to decimal and save it in D register.

Step - 8 :- Decrement the decimal count in C register by one.
Adding 99 to C register is equivalent to decrementing by 1.
Eg :- (15 + 99 = 114 Which is taken as 14 with carry 1).

Step - 9 :- If (C) is not zero, go to Step 6.

Step -10 :- Store the BCD product in Memory.

Step -11 :- End.

CODING :-

```
    LDA   2050H
    MOV   B A
    LDA   2051H
    MOV   C A
    XRA   A
AGAIN: ADD   B
    DAA
    MOV   D A
    MOV   A C
    ADI   99
    DAA
    MOV   C A
    MOV   A D
    JNZ   AGAIN
    STA   2052H
    HLT.
```

8 - bit Division.

ALGORITHM :-

Step - 1 :- Get the divisor from the memory to the accumulator.

Step - 2 :- Transfer the divisor to B register.

Step - 3 :- Get the dividend from the memory to the accumulator.

Step - 4 :- Initialize C register with -1 (i.e) FF_H to store the quotient.

Step - 5 :- Increment the C register content by one.

Step - 6 :- Subtract the divisor in B from the dividend in A.

Step - 7 :- If the dividend is greater than the divisor, then go to Step 5.

Step - 8 :- If the dividend is less than the divisor, add the contents of A and B to get the remainder.

Step - 9 :- Store the remainder in Memory.

Step - 10 :- Store the quotient in Memory.

Step - 11 :- End.

CODING:-

```
LDA   2050H
MOV   B A
LDA   2051H
MVI   C FFH
HERE: INR   C
SUB   B
JNC   HERE
ADD   B
STA   2053H
MOV   A C
STA   2052H
HLT.
```

Output :

Address	Data
2050	10
2051	10
2052	00
2053	01
2054	00

Sorting And Searching

Searching for an element in an array.

ALGORITHM:-

Step -1:- Initialize the HL register pair with the starting address of the memory that stores an array of elements.

Step -2:- Load the number of element in the array as a count in C register.

Step -3:- Take an element from the memory to the accumulator.

Step -4:- Compare it with the element to be searched.

Step -5:- If a Match is found, go to Step 9.

Step -6:- Else, increment the memory address to take the next element for comparison.

Step -7:- Decrement the count in C register.

Step -8:- If the count is not zero, go to Step 3.

Step -9:- Store the Searched element in another memory location.

Step -10 :- Store the address of that element in the next two consecutive memory location.

Step -11 :- If no match is found after checking all the elements in the array, then store FF_H in a Memory location indicating the search is a failure.

Step -12 :- End :

CODING :-

```
      LXI   H  2050H
      MVI   C  05H
LOOP2 MOV   A  M
      CPI   EEH
      JZ    LOOP 1
      INX   H
      DCR   C
      JNZ   LOOP 2
      MVI   A  FFH
      STA   2100H
      JMP   EXIT
LOOP1 STA   2100H
      SHLD  2101H
EXIT  HLT.
```

Input :

Address	Data
2050	25
2051	1A
2052	EE
2053	5D
2054	98

Result :

2100	EE
2101	52
2102	20

Search is success

Input :

Address	Data
2050	24
2051	1C
2052	1A
2053	3E
2054	27

Result :

2100	FF

Search is not a success

Sorting in Ascending And Descending Order.

ALGORITHM:-

Step-1:- Load 'N' bytes in memory whose starting address is in HL register pair.

Step-2:- Load (N-1) in C register, to be used as a counter.

Step-3:- Copy the count in D register.

Step-4:- Initialize HL register pair with the memory address.

Step-5:- Move the first data from memory to accumulator.

Step-6:- Increment memory address in HL.

Step-7:- Move the second data from memory to B register.

Step-8:- Compare the data in A register with the data in B register.

Step-9:- If data in A register is smaller than the data in B register go to Step 11.

Step-10:- Else, 'swap' the first data with the second data stored in Memory.

Step-11:- Decrement count in register D by one.

Step - 12 :- If count in D register is n
go to step 5.

Step - 13 :- Decrement the count in C
by one

Step - 14 :- If count in C register is no
go to Step 8

Step - 15 :- End.

CODING :-

```
      MVI   C  09H
LOOP3 MOV   D  C
      LXI   H  2050H
LOOP2 MOV   A  M
      INX   H
      MOV   B  M
      CMP   B
      JC    LOOP 1
      MOV   M  A
      DCX   H
      MOV   M  B
      INX   H
LOOP1 DCR   D
      JNZ   LOOP 2
      DCR   C
      JNZ   LOOP 3
      HLT.
```

Address	Data
2050	25
2051	32
2052	16
2053	34
2054	45
2055	60H
2056	69

Output:

Address	Data
2050	00
2051	00
2052	00
2053	16
2054	25
2055	32
2056	13
2057	45
2058	56
2059	60

Finding the largest and Smallest element in an array.

ALGORITHM:-

Step - 1:- Load 'N' bytes in Memory whose starting address is in HL register pair.

Step - 2:- Load (N-1) in C register, to be used as a counter.

Step - 3:- Move the first data from memory to accumulator.

Step - 4:- Increment memory address in HL.

Step - 5:- Move the second data from memory to B register

Step - 6:- Compare the data in A register with the data in B register.

Step - 7:- If data in A register is smaller then go to step 9.

Step - 8:- Else, 'swap' the first data with the second data stored in Memory.

Step - 9:- Decrement count register C by one.

Step - 10:- If C register content is not equal to zero, go to step 3.

Step - 11:- End.

CODING:-

```
        LXI   H  2050H
        MVI   C  09H
LOOP:   MOV   A  M
        INX   H
        MOV   B  M
        CMP   B
        JC    LOOP1
        MOV   M  A
        DCX   H
        MOV   M  B
        INX   H
LOOP1:  DCR   C
        JNZ   LOOP2
        HLT.
```

Input:-

Address	Data
2050	12
2051	34
2052	54
2053	10
2054	15

Output Data:-

Address	Data
2050	12
2051	34
2052	10
2053	15
2054	54

Verified.
28/03/24

Reversing array elements.

ALGORITHM :-

Step - 1 :- Initialize HL register pair with the starting address of the Memory which contain an array of 'N' numbers.

Step - 2 :- Initialize DE register pair with the last address of the memory.

Step - 3 :- Load C register with the Number of exchanges required which is equal to N/2 or (N-1)/2 depending on the number of elements in the array being even or odd.

Step - 4 :- Get the first element from the memory pointed by HL, to B register.

Step - 5 :- Get the last element from the memory by DE, to A register.

Step - 6 :- Store the content of A to memory pointed by HL.

Step - 7 :- Transfer the content B to A and then to memory pointed by DE. Now the contents of the first Memory location and last Memory location are exchanged.

Step-8 :- Increment the memory address in HL and decrement the memory address in DE.

Step-9 :- Decrement the count in C register by one.

Step-10 :- End.

CODING :-

```
      LXI   H   2050H
      LXI   D   2054H
      MVI   C   02H
LOOP: MOV   B   M
      LDAX  D
      MOV   M   A
      MOV   A   B
      STAX  D
      INX   H
      DCX   D
      DCR   C
      JNZ   LOOP
      HLT.
```

Input Data :-

Address	Data
2050	10
2051	20
2052	30
2053	40
2054	50

Output Data :-

Address	Data
2050	50
2051	40
2052	30
2053	20
2054	10

Verified,
28/03/24

Block Move

ALGORITHM :-

Step - 1 :- Initialize HL register pair with the starting address of the Memory which contain an array of N Numbers ; source block.

Step - 2 :- Initialize DE register pair with the memory to which the elements are going to be transferred ; destination block.

Step - 3 :- Load C register with the number of elements in the given array.

Step - 4 :- Get the element from the memory pointed by HL, source block.

Step - 5 :- Store the element in the memory pointed by DE, destination block.

Step - 6 :- Increment the memory address in HL to get the next element from the source block.

Step - 7 :- Increment the memory address in DE, to store the next element in the destination block.

Step-8:- Decrement the count in c register by one.

Step-9:- If (c) is not zero, go to Step 4.

Step-10:- End.

CODING:-

```
      LXI   H   2050H
      LXI   D   2160H
      MVI   C   05H
LOOP: MOV   A   M
      STAX  D
      INX   H
      INX   D
      DCR   C
      JNZ   LOOP
      HLT
```

Input Data:-

Address	Data
2050	56
2051	54
2052	32
2053	40
2054	21

Output Data:-

Address	Data
2160	56
2161	54
2162	32
2163	40
2164	21

Verified,
28/03/24.

CODE CONVERSION.

BCD to Hexa Decimal Conversion

Step 1:

Input the BCD Number.

Take the BCD Number as input, which is a sequence of 4-bit group where each group represent a decimal digit

Step 2: Group the BCD Number into 4-bit Block.

Divide the BCD Number into separate group of 4bits. Each group represent the decimal digit.

Step 3: Convert Each 4-bit Group to its Hexadecimal Equivalent.

Convert each 4-bit BCD group to its corresponding hexa decimal digit

Step 4: Combine the Hexa decimal Digits

After converting Each 4-bit BCD group to hexa decimal, Combine all the hexadecimal digits to form the final hexa decimal result

Step 5: Output the Hexadecimal Number

Output the final hexadecimal number formed by the combined Hexa decimal digit.

CODING:-

```
ORG 0000H ;
MOV A, BCD_DATA ;
CALL BCD-TO-HEX;
HLT ;Halt
BCD_TO_HEX:
```

```
ANI 0FH ;
CPI 0AH ;
JC NOT_TEN ;
ADD 06H;
NOT_TEN: ADD 30H;
MOV C,A;
MOV A, BCD_DATD;
RRC ;
RRC ;
RRC ;
RRC ;
ANI 0F H;
CPI 0A H;
JC NOT_TEN_2;
ADD 06H;
NOT_TEN_2;
ADD 30H
MOV B,A;
RET ;
BCD_DATD DB 32H;
END;
```

OUTPUT:

Register C: 32H (ASCII '2')

Register B: 33H (ASCII '3').

Hexa decimal to BCD conversion:

Step 1: Input the Hexadecimal Number.
Take the hexadecimal number as input, which consist of hexadecimal digits (0-F).

Step 2: Convert Each Hexadecimal Digit to a 4 bit Binary.
For each hexadecimal digit, convert it to its corresponding 4-bit binary (BCD) representation

Step 3: Combine the BCD Groups.
Combine all the 4-bit BCD groups together to form the final BCD representation

Step 4: Output the BCD Number:
Output the final BCD number formed by combining all the BCD groups.

CODING:

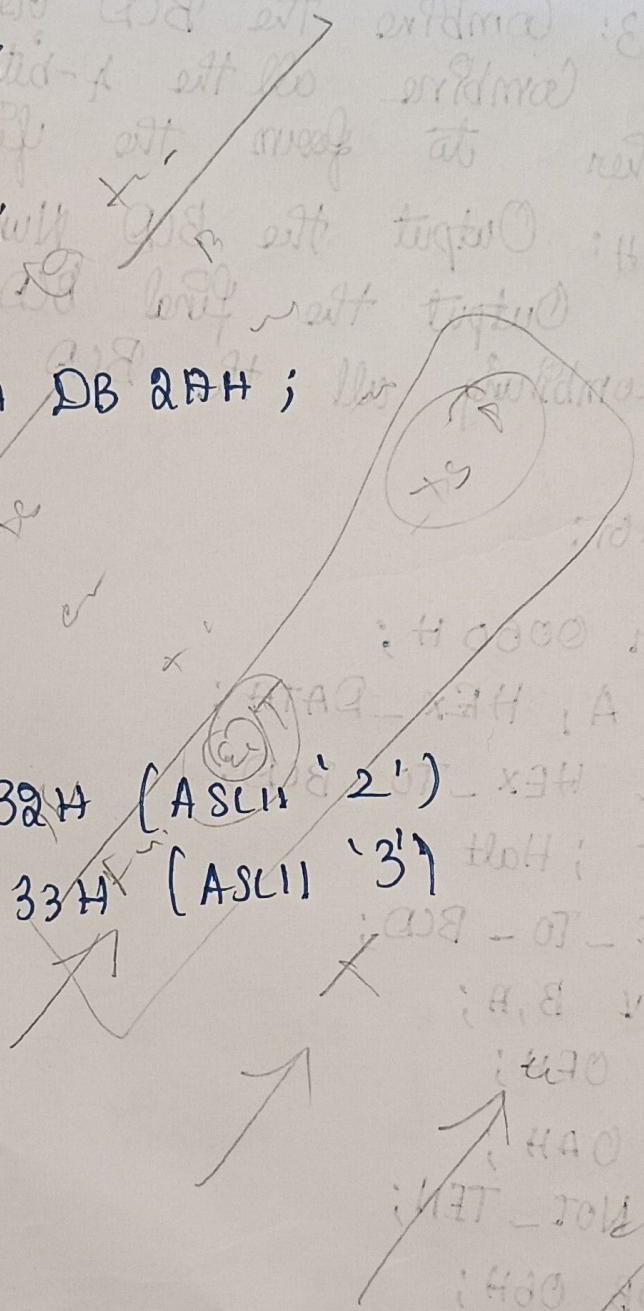
```
ORG 0060H;
MOV A, HEX_DATA;
CALL HEX_TO_BCD;
HLT ; Halt
HEX_TO_BCD;
MOV B,A;
ANI 0FH;
CPI 0AH ;
JC NOT_TEN;
SUB 06H;
NOT_TEN
  DAA ;
  RLC ;
  MOV C, A ;
```

```
MOV A,B ;
ANI OFH ;
API OAH ;
JC NOT_TEN_2;
SUB 06H ;
NOT_TEN_2;
DAA ;
MOV B,A ;
RLC ;
MOV A,C;
RET ;
HEX_DATA DB 2AH ;
END;
```

OUTPUT:

Register C: 32H (ASCII '2')

Register B: 33H (ASCII '3')



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.

CODING :-

```
        LDA    2050H
        CPI    0AH
        JC     SKIP7
        ADI    07H
SKIP7:  ADI    30H
        STA    2051H
        HLT.
```

OUTPUT :-

Address	Data
2050	7
2051	37

ASCII to Binary.

ALGORITHM:-

Step -1:- Take the ASCII number in accumulator.

Step -2:- Subtract 30_H 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.

CODING:-

```
        LDA     2050H
        SUI     30H
        CPI     0AH
        JC      SKIP 7
        SUI     07H
SKIP 7: STA     2051H
        HLT.
```

OUTPUT:-

address	Data
2050	69
2051	32

Verified.
28/03/24

ASCII to BCD

ALGORITHM :-

Step - 1 :- Take the ASCII number (i.e. ASCI number from 30 to 39) in accumulator.

Step - 2 :- Subtract 30H from the accumulator

Step - 3 :- Store the BCD result in Memory.

Step - 4 :- End.

CODING :-

```
LDA   2050H
SUI   30H
STA   2051H
HLT.
```

OUTPUT :-

address	Data
2050	45
2051	15

Verified

28/03/24

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.

CODING :-

```
LDA   2050H
ADI   30H
STA   2051H
HLT.
```

OUTPUT :-

Address	Data
2050	10
2051	40

verified
28/03/24

Application.

The Square of a Single Byte Hexa Decimal Number.

Step 1: Input the Hexa decimal Number.
Take the single byte Hexa decimal number into as input. A single byte Hexa decimal number can range from 0x00 to 0xFF.

Step 2: Convert the Hexadecimal Number to Decimal.
Convert the given hexadecimal Number into its decimal Equivalent, This will allow for easier mathematical Operations

Step 3: Square the Decimal Number.
Square the decimal value obtained from the previous step.

Step 4: Convert the Squared Decimal Result Back to Hexadecimal.
Convert the squared decimal result back into its hexa decimal form.

Step 5: Output the Result.
Output the final result in hexadecimal format. This is the Square of the original input hexadecimal number.

CODING:

```
ORG 0000H ;
MOV A, HEX_DATA ;
CALL SQUARE ;
HLT ;
SQUARE :
    MOV B, A ;
    MVI C, 00H ;
    MOV D, C ;
LOOP :
    ADD B ;
    JC CARRY :
    INR C ;
CARRY :
    DCR B ;
    JNZ LOOP ;
    MOV A, C ;
    RET ;
    HEX_DATA DB 0AH ;
    END ;
```

RESULT:

A = 64H (Hexa decimal) = 100 (Decimal).

21. Calculate the Square of a Two Digit BCD Number;

Step 1: Input the Two-Digit BCD Number.

The two-digit BCD number consists of two 4-bit groups representing each decimal digit; for Example, 0001 0010 for the decimal number 12 in BCD.

Step 2: Convert the BCD Number to Decimal.

Convert the two digit BCD number into its decimal Equivalent. Each 4-bit BCD group represents one decimal digit.

Step 3: Square the Decimal Number.

Once you have the decimal Equivalent, Square the decimal value.

Step 4: Convert the Squared Decimal Result Back to BCD.

Convert the squared decimal result back into BCD format, breaking the squared result into its individual decimal digits and representing each digit with a 4-bit BCD group.

Step 5:

Output the Result.

CODEING:

```
ORG 0000H;
MOV A, MSB_;
MOV B, A ;
MOV A, LSB_DATA;
MOV C, A;
CALL SQUARE_ROOT;
HLT;
SQUARE_ROOT;
MVI D, 00H;
MVI E, 00H;
LOOP;
MOV A, B ;
MOV B, A ;
MOV A, E ;
ADD B ;
MOV E, A ;
MOV A, C;
MOV C, A;
MOV A, E;
ADD C ;
JNC CONTINUE_LOOP;
DCR B ;
SUB B ;
INX H;
CONTINUE_LOOP:
INX D;
CPI 10H;
JNZ LOOP;
RET ;
MSB_DATA DB 03H;
LSB_DATA DB 02H;
```

Output:

Input BCD_DATA = 32H

Square of 32 (Decimal): 32 X 32 = 1024

Square of 32 (HexaDecimal): 1024 is 0400H.

The Output of this Program is the value 0400H (The square of 32 in Hexa decimal) stored in 20 D and C register.

3. The Square Root of a single-Byte Hexadecimal Number:

Step 1: input the Hexa decimal Number

Take the single-byte hexa decimal number as input. A single-byte hexa decimal number ranges from 0x00 to 0xFF (0 to 255 in decimal)

Step 2: convert the Hexa decimal Number to Decimal.

Convert the hexa decimal input number into its decimal equivalent. This conversion allows you to easily calculate the Square root

Step 3: Calculate the Square Root of the Decimal Number

Calculate the Square root of the decimal number value obtained from the Previous Step.

Step 4: convert the Square Root Result back to Hexadecimal.

After calculating the Square root in decimal, convert the result back into hexa decimal format.

Step 5: Output the result

Output the final result in hexadecimal format

Output:

A = 03H (Hexadecimal)

CODING:

```
ORG 0000H;
MOV A, HEX_DATA;
ALL SQUARE_ROOT;
HLT;
SQUARE_ROOT;
MVI B, 00H;
MVI C, 00H;

LOOP:
MOV D, A;
SUB C;
JC END_LOOP;
INR C;
JMP LOOP:
DCR C;
MOV A, C;
RET;

HEX_DATA DB 09H;
END;
```

output:

A = 03H (Hexa Decimal) = 3 (Decimal)

4. Calculate the square root of a Two-Digit BCD Number

Step 1: Input the Two-Digit BCD Number

A two-digit BCD number consists of two 4-bit groups, each representing one decimal digit. for example, the BCD representation 0001, 0010 represent the decimal number 12

Step 2: convert the BCD Number to Decimal

Convert the BCD number into its Decimal equivalent. Each 4-bit group corresponds to one decimal digit

Step 3: calculate the square root of the Decimal Number

After obtaining the decimal number, calculate the square root that decimal number

Step 4: convert the square root Result BACK to BCD

convert the square root result back into BCD if the square root is a whole number. convert decimal digit back into 4-bit BCD representation

Step 5: Output the Result

Output the square root in BCD format.

CODING:-

```
MOV A, MSB-;
MOV B, A;
MOV A, LSB-DATA;
MOV C, A;
CALL Square-Root;
HLT;
Square-ROOT;
.MVI D, 00H;
MVI E, 00H;
LOOP;
MOV A, B;
MOV B, A;
MOV A, E;
ADD B;
MOV E, A;
MOV A, C;
MOV C, A;
MOV A, E;
ADD C;
JNC CONTINUE-LOOP;
DCR B;
SOB B;
INX H;
CONTINUE-LOOP;
INX D;
CPI 10H;
JNZ LOOP;
MOV A, E;
RET;
MSB-DATA DB 03H;
LSB-DATA DB 02H
```

output:

MSB-DATA = 03H

LSB-DATA = 02H