MICROPROCESSOR AND MICROCONTROLLER BASED SYSTEMS (FOR CSE & IT)

UNIT 2

8085 INSTRUCTION SET AND ASSEMBLY LANGUAGE PROGRAMMING 9 Hrs. Instruction classifications, Writing and executing simple programs - Arithmetic and logic operations - Data transfer - Branching - Looping – Indexing - Counter and time delays - Writing subroutine - Conditional call and return instruction, simple programs.

Instruction Set Classification

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: **data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.**

1. **Data Transfer (Copy) Operations**

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. In technical manuals, the term *data transfer* is used for this copying function. However, the term *transfer* is misleading; it creates the impression that the contents of the source are destroyed when, in fact, the contents are retained without any modification. The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
1. Between Registers.	1. Copy the contents of the register B into register D.
2. Specific data byte to a register or a memory location.	2. Load register B with the data byte 32H.
3. Between a memory location and a register.	3. From a memory location 2000H to register B.
4. Between an I/O device and the accumulator.	4.From an input keyboard to the accumulator.

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altere . If one of the operands is a memory ocation, its ocation is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is sto ed in the destination register or memory. If the ope and is a memo y ocation, its location is specified by the contents of the HL registers. Example: MVI B, 57 MVI M, 57
Load accumulator		
LDA	16-bit address	The c ntents f a mem ry location, specified by a16-bit address in the perand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034 or LDA XYZ
Load accumulator indirect		
LDAX	B/D Reg. pai	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Store accumulator direct		
S A	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350 or STA XYZ
Store accumulator indirect		
S AX	Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

LXI Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034
Load H and L registers direct LHLD 16-bit address	WorldThecontentsofregisterHareexchangedwiththecontents fregister D, and the contents of register L are exchanged The instruction copies the contents of the memory location pointed out by the 16-bit a ress into register L and copies the contents of the next memory location into register H. The contents of source memory ocations are not altered. Example: LHLD 2040
Store H and L registers direct SHLD 16-bit address	The contents fregiste L a e sto ed into the memory location specified by the 16-bit add ess in the operand and the contents f H register are stored into the next memory location by incrementing the perand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the l w- rder address and the third byte specifies the high-order address. Example: SHLD 2470
Exchange H and L with D and E XCHG none	with the contents of register E. Example: XCHG

2. Arithmetic Operations

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition - Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

Subtraction - Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's compliment, and the results if negative, are expressed in 2's complement. No two other registers can be subtracted directly.

Increment/Decrement - The 8-bit contents of a register or a memory location can be incremented or

decrement by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decrement by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is store in the accumulator. If the operand is a memory location, its location is specified by the contents of the HLregisters. A f ags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the ope and (registe o memo y) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulat . If the pe and is a memory location, its l cati n is specified by the contents of the HL registers. All flags are m dified t reflect the result of the addition. Example: ADC B r ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accum ulator and the result is stored in the accum ulator. All flags are modified to reflect the result of the addition. Example: ADI 45
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

SUB R
M
The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.
Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB R
M
The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specifie by the contents of the HL registers. All flags are mo ifie to reflect the result in accumulator.
Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data
The 8-bit data (perand) is subt acted f m the contents of the accumulator and the result is sto ed in the accumulator. All flags are modified t reflect the esult f the subt action.
Example: SUI 45

Subtract immediate from accumulator with borrow

SBI 8-bit data
The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion.
Example: SBI 45

Increment register or memory by 1

INR R
M
The contents of the designated register or memory) are incWorldemented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: INR B or INR M

Increment register pair by 1

I X R
The contents of the designated register pair are incremented by 1 and the result is stored in the same place.
Example: INX H

Decrement register or memory by 1 DCR R M	The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: DCR B or DCR M
Decrement register pair by 1 DCX R	The contents of the designate register pair are decremented by 1 and the result is store in the same place. Example: DCX H
Decimal adjust accumulator DAA none	The contents of the accumulato a e change f om a binary value to two 4-bit binary coded decima (BCD) digits. This is the only instruction that uses the auxi ia y f ag to perform the binary to BCD conversion, and the conve sion p ocedure is described below. S, Z, AC, P, CY flags a e alte ed to reflect the results of the perati n. If the value f the low- rder 4-bits in the accumulator is greater than 9 r if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greate than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example: DAA

3. Logical Operations

These instructions perform various logical operations with the contents of the accumulator.

AND, OR Exclusive-OR - Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. The results are stored in the accumulator.

Rotate- Each bit in the accumulator can be shifted either left or right to the next position.

Compare- Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.

Complement - The contents of the accumulator can be complemented. All 0s are replaced by 1s and all 1s are replaced by 0s.

Opcode	Operand	Description
Compare register or memory with accumulator		
CMP	R M	The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:

if (A) < (reg/mem): carry flag is set, s=1
if (A) = (reg/mem): zero flag is set, s=0
if (A) > (reg/mem): carry and zero flags are reset, s=0
Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data — The second byte (8- bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

if(A) < data: carry flag is set, s=1

if (A) = data: zero flag is set, s=0
if (A) > data: carry and zero flags are reset, s=0
Example: CPI 89

Logical AND register or memory with accumulator

ANA R / M — The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.
Example: ANA B or ANA M

Logical AND immediate with accumulator

ANI 8-bit data — The contents of the accumulator are ogica y ANDe with the 8-bit data (operand) and the esu t is p ace in the accumulator. S, Z, P are modified to ef ect the esult of the operation. CY is reset. AC is set.
Example: ANI 86

Exclusive OR register or memory with accumulator

XRA R / M — The contents f the accumulator are Exclusive ORed with the contents f the perand (register r memory), and the result is placed in the accumulator. If the operand is a n, its address is specified by the contents memory locati of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.
Example: XRA B or XRA M

Logical OR register or memory with accumulator

ORA R / M — The contents of the accumulator are logically ORed with the contents f the operand (register/memory), and the

Examp e: JZ 2034 or JZ XYZ
Result is placed in the accumulator. If the operand is a
n, its address is specified by the contents
mem y cati of
HL registe s. S, Z, P are modified to reflect the result of the
operation. CY and AC are reset.
Example: ORA B or ORA M

Exclusive OR immediate with accumulator

XRI 8-bit data — The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.
Example: XRI 86

Logical OR immediate with accumulator

ORI 8-bit data — the contents of the accumulator are logically ORed with the
8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.
Example: ORI 86

Complement accumulator

CMA none — The contents of the accumulator are complemented. No flags
are affected.Example: CMA

Complement carry

CMC none — The Carry flag is complemented. No other flags are affected.
Example: CMC

Set Carry

STC none — The Carry flag is set to 1. No other flags are affected.
Example: STC

Branching Operations

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

Jump - Conditional jumps are an important aspect of the decision-making process in the programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called *unconditional jump.*

Call, Return, and Restart - These instructions change the sequence of a program either by calling

a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transfe ed to the memo y ocation specified by the 16-bit add ess given in the ope and. Example: JMP 2034 or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred t the memory location specified by the 16- bit address given in the operand based on the specified flag of the PS as described below.
Examp e: JZ 2034 or JZ XYZ

World

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call

CALL	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034 or CALL XYZ

Call conditionally

Operand: 16-bit address

he program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.
Example: CZ 2034 or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

RET none

The program sequence is t ansfe ed f om the subroutine to the calling program. The two bytes f om the top of the stack are copied int the pr gram c unte , and p og am execution begins at the new address.
Example: RET

Return from subroutine conditionally

Operand: none

The p ogram sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copie into the program counter, and program execution begins at the new address.

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

PCHL none — The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
Example: PCHL

Restart

RS 0-7 — The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Program 2:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional inte upts an these interrupts generate RST instructions inte na y and thus do not require any external hardware. These inst uctions and their Restart addresses are:

Interrupt	Resta t Add ess
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

SAMPLE PROGRAMS

1. Store the data byte 32H into memory location 4000H. MVI A, 52H : Store 32H in the accumulator

```
STA 4000H :    Copy accumulator contents at address 4000H
HLT :          Te minate program execution

LXI H :        Load HL with 4000H
MVI M :        Store 32H in memory location pointed by HL register pair
HL :           Terminate program execution
```

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

```
LDA 2000H    : Get the contents of memory location 2000H into accumulator
MOV B, A     : Save the contents into B register
LDA 4000H    : Get the contents of memory location 4000Hinto accumulator
STA 2000H    : Store the contents of accumulator at address 2000H
MOV A, B     : Get the saved contents back into A register
S TA 4000H   : Store the contents of accumulator at address 4000H
```

4. Machine Control Operations

These instructions control machine functions such as Halt, Interrupt, or do nothing.
The microprocessor operations related to data manipulation can be summarized in four functions:

1. copying data
2. performing arithmetic operations
3. performing logical operations
4. testing for a given condition and alerting the program sequence

Some important aspects of the instruction set are noted below:

1. In data transfer, the contents of the source are not destroyed; only the contents of the destination are changed. The data copy instructions do not affect the flags.
2. Arithmetic and Logical operations are performed with the contents of the accumulator, and the results are stored in the accumulator (with some expectations). The flags are affected according to the results.
3. Any register including the memory can be used for increment and decrement.
4. A program sequence can be changed either conditionally or by testing for a given data condition.

8. Instruction Format

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each

instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.
For example:

Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

MOV rd, rs
rd <-- rs copies contents of rs into rd.
Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B
Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r
A <-- A + r

Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

Task	**Opcode**	**Operand**	**Binary Code**	**Hex Code**	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110 DATA	3E Data	First Byte Second Byte

Assume that the data byte is 32H. The assembly language instruction is written as

Mnemonics	**Hex code**
MVI A, 32H	3E 32H

The instruction would require two memory locations to store in memory.

MVI r,data r <-- data
Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

ADI data
A <-- A + data

OUT port
where port is an 8-bit device address. (Port) <-- A. Since the byte is not the data but points directly to where it is located this is called direct addressing.

Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.
opcode + data byte + data byte

For example:

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011 1000 0101 0010 0000	C3 85 20	First byte Second Byte Third Byte

This instruction would require three memory locations to store in memory.

Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.
rp <-- data16

Example:
LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A <-- (addr) Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

Looping:
The program is written in such a way that it executes certain set of instructions repeatedly to execute a certain task for a number of times. Example: to perform multiplication of two numbers. This is nothing but repeated addition.

Counting: The program has a track of how many times the instruction or a set of instructions are executed. Eg: When division operation is performed a register is used to count the number of times the subtraction is done. The content of that register will give the quotient.

Indexing: it allows the user to point or refer the data stored in a sequential memeory location one by one.

Sample Programs

Write an assembly program to multiply a number by 8

Program

```
MVI   A, 30H
RRC
RRC
RRC
OUT   PORT1
HLT
```

ADDITION OF TWO 8 BIT NUMBERS

PROGRAM:

```
        MVI   C, 00    Initialize C register to 00
        LDA   4150     Load the value to Accumulator.
        MOV   B, A     Move the content of Accumulator to B register.
        LDA   4151     Load the value to Accumulator.
        ADD   B        Add the value of register B to A
        JNC   LOOP     Jump on no carry.
        INR   C        Increment value of register C
LOOP:   STA   4152     Store the value of Accumulator (SUM).
        MOV   A, C     Move content of register C to Acc.
        STA   4153     Store the value of Accumulator (CARRY)
        HLT            Halt the program.
```

OBSERVATION:

```
Input:    80 (4150)
          80 (4251)
Output:   00 (4152)
          01 (4153)
```

SUBTRACTION OF TWO 8 BIT NUMBERS

PROGRAM:

```
        MVI     C, 00     Initialize C to 00
        LDA     4150      Load the value to Acc.
        MOV     B, A      Move the content of Acc to B register.
        LDA     4151      Load the value to Acc.
        SUB     B
        JNC     LOOP      Jump on no carry.
        CMA               Complement Accumulator contents.
        INR     A         Increment value in Accumulator.
        INR     C         Increment value in register C
LOOP:   STA     4152      Store the value of A-reg to memory address.
        MOV     A, C      Move contents of register C to Accumulator.
        STA     4153      Store the value of Accumulator memory
        HLT               Terminate the program.
```

OBSERVATION:

Input: 06 (4150)

02 (4251)

Output: 04 (4152)

01 (4153)

MULTIPLICATION OF TWO 8 BIT NUMBERS

PROGRAM:

```
        MVI     D, 00     Initialize register D to 00
        MVI     A, 00     Initialize Accumulator content to 00
        LXI     H, 4150
        MOV     B, M      Get the first number in B - reg
        INX     H
        MOV     C, M      Get the second number in C- reg.
LOOP:   ADD     B         Add content of  A - reg to register
        JNC     NEXT      Jump on no carry to NEXT.
        INR     D         Increment content of register D
NEXT:   DCR     C         Decrement content of register C.
        JNZ     LOOP      Jump on no zero to address
        STA     4152      Store the result in Memory
```

```
MOV   A, D
STA   4153    Store the MSB of result in Memory
HLT           Terminate the program.
```

OBSERVATION:

Input: FF (4150)
FF (4151)
Output: 01 (4152)
FE (4153)

DIVISION OF TWO 8 BIT NUMBERS

PROGRAM:

```
        LXI   H, 4150
        MOV   B, M     Get the dividend in B – reg.
        MVI   C, 00    Clear C – reg for qoutient
        INX   H
        MOV   A, M     Get the divisor in A – reg.
NEXT:   CMP   B        Compare A - reg with register
        JC    LOOP     Jump on carry to LOOP
        SUB   B        Subtract A – reg from B- reg.
        INR   C        Increment content of register
        JMP   NEXT     Jump to NEXT
LOOP:   STA   4152     Store the remainder in
        MOV   A, C
        STA   4153     Store the quotient in memory
        HLT            Terminate the program.
```

OBSERVATION:

Input: FF (4150)
FF (4251)

Output: 01 (4152) ---- Remainder
FE (4153) ---- Quotient

LARGEST NUMBER IN AN ARRAY OF DATA

PROGRAM:

```
        LXI     H,4200      Set pointer for
        MOV     B,M         array Load the
        INX     H           Count
        MOV     A,M         Set 1st element as largest data
        DCR     B           Decrement the count
LOOP:   INX     H
        CMP     M           If A- reg > M go to AHEAD
        JNC     AHEAD
        MOV     A,M         Set the new value as largest
AHEAD:  DCR     B
        JNZ     LOOP        Repeat comparisons till count =
        STA     4300        Store the largest value at 4300
        HLT
```

OBSERVATION:

Input: 05 (4200) ----- Array Size
0A (4201)
F1 (4202)
1F (4203)
26 (4204)
FE (4205)

Output: FE (4300)

SMALLEST NUMBER IN AN ARRAY OF DATA

PROGRAM:

```
        LXI     H,4200      Set pointer for
        MOV     B,M         array Load the
        INX     H           Count
        MOV     A,M         Set 1st element as largest data
        DCR     B           Decrement the count
LOOP:   INX     H
        CMP     M           If A- reg < M go to AHEAD
        JC      AHEAD
        MOV     A,M         Set the new value as smallest
AHEAD:  DCR     B
        JNZ     LOOP        Repeat comparisons till count =
        STA     4300        Store the largest value at 4300
        HLT
```

OBSERVATION:

Input: 05 (4200) ----- Array Size
0A (4201)
F1 (4202)
1F (4203)
26 (4204)
FE (4205)

Output: 0A (4300)

ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

```
        LXI     H,4200
        MOV     C,M
        DCR     C
RFPFAT: MOV     D.C
        LXI     H,4201
LOOP:   MOV     A,M
        INX     H
        CMP     M
        JC      SKIP
        MOV     B,M
```

```
            MOV     M,A
            DCX     H
            MOV     M,B
            INX     H
SKIP:       DCR     D
            JNZ     LOOP
            DCR     C
            JNZ     REPEAT
            HLT
```

OBSERVATION:

Input	4200	05 (Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

Output:	4200	05(Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

ARRANGE AN ARRAY OF DATA IN DESCENDING ORDER

PROGRAM:

```
            LXI     H,4200
            MOV     C,M
            DCR     C
REPEAT:     MOV     D,C
            LXI     H,4201
LOOP:       MOV     A,M
            INX     H
```

```
        CMP     M
        JNC     SKIP
        MOV     B,M
        MOV     M,A
        DCX     H
        MOV     M,B
        INX     H
SKIP:   DCR     D
        JNZ     LOOP
        DCR     C
        JNZ     REPEAT
        HLT
```

OBSERVATION:

Input	4200	05 (Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

Output:	4200	05(Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

BCD TO HEX CONVERSION

PROGRAM:

```
        LXI     H,4150
        MOV     A,M     Initialize memory
        ADD     A       MSD X 2
        MOV     B,A     Store MSD X 2
```

```
        ADD     A       MSD X 4
        ADD     A       MSD X 8
        ADD     B       MSD X 10
        INX     H       Point to LSD
        ADD     M       Add to form HEX
        INX     H
        MOV     M,A     Store the result
        HLT
```

OBSERVATION:

Input: 4150 : 02 (MSD)
4151 : 09 (LSD)

Output: 4152 : 1D H

HEX TO BCD CONVERSION

PROGRAM:

```
        LXI     H,4150  Initialize memory pointer
        MVI     D,00    Clear D- reg for Most significant
        XRA     A       Clear Accumulator
        MOV     C,M     Get HEX data
LOOP2:  ADI     01      Count the number one by one
        DAA             Adjust for BCD count
        JNC     LOOP1
        INR     D
LOOP1:  DCR     C
        JNZ     LOOP2
        STA     4151    Store the Least Significant Byte
        MOV     A,D
        STA     4152    Store the Most Significant Byte
        HLT
```

OBSERVATION:

Input: 4150 : FF

Output: 4151 : 55 (LSB)
4152 : 02 (MSB)

HEX TO ASCII CONVERSION

PROGRAM:

```
        LDA     4200    Get Hexa Data
        MOV     B,A
        ANI     0F      Mask Upper Nibble
        CALL    SUB1    Get ASCII code for upper nibble
        STA     4201
        MOV     A,B
        ANI     F0      Mask Lower Nibble
        RLC
        RLC
        RLC
        RLC
        CALL    SUB1    Get ASCII code for lower nibble
        STA     4202
        HLT
SUB1:   CPI     0A
        JC      SKIP
        ADI     07
SKIP:   ADI     30
        RET
```

OBSERVATION:

Input:	4200	E4(Hexa data)
Output:	4201	34(ASCII Code for 4)
	4202	45(ASCII Code for E)

ASCII TO HEX CONVERSION

PROGRAM:

```
        LDA   4500
        SUI   30
        CPI   0A
        JC    SKIP
        SUI   07
SKIP:   STA   4501 HLT
```

OBSERVATION:

Input: 4500 31

Output: 4501 0B

SQUARE OF A NUMBER USING LOOK UP TABLE

ALGORITHM:

1. Initialize HL pair to point Look up table
2. Get the data .
3. Check whether the given input is less than 9.
4. If yes go to next step else halt the program
5. Add the desired address with the accumulator content
6. Store the result

PROGRAM:

```
        LXI   H,4125      Initialsie Look up table
        LDA   4150        Get the data
        CPI   0A          Check input > 9
        JC    AFTER       if yes error
        MVI   A,FF        Error Indication
        STA   4151
        HLT
AFTER:  MOV   C,A         Add the desired Address
        MVI   B,00
        DAD   B
        MOV   A,M
        STA   4151        Store the result
        HLT               Terminate the program
```

LOOKUP TABLE:

4125	01
4126	04
4127	09
4128	16
4129	25
4130	36
4131	49
4132	64
4133	81

OBSERVATION:

Input: 4150: 05

Output: 4151 25 (Square)

Input : 4150: 11

Output: 4151: FF (Error Indication)

RESULT:

Thus the program to find the square of the number from 0 to 9 using a Look up table was executed