CHAPTER-2: BASIC-PROGRAMMING CONCEPTS

2.1 INSTRUCTION SET OF MICROPROCESSOR

2.1.1 FLOW CHART SYMBOLS

To develop the programming logic, programmer must write down various actions which are to be performed in proper sequence. The flow chart is a graphical tool that allows programmer to represent various actions which are to be performed. Figure 2.1 shows the graphical symbols used in flow chart.

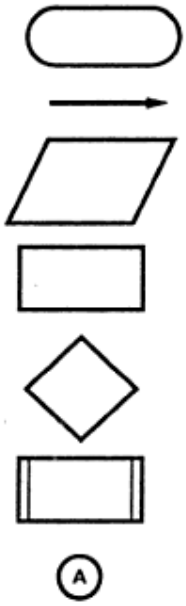


Fig 2.1 Graphical Symbols used in Flow Chart

The description of the different symbols is given below:

1. Oval: indicates start or stop operation.
2. Arrow: indicates flow with direction
3. Parallelogram: indicates input/output operation.
4. Rectangle: indicates process operation
5. Diamond: indicates decision making operation
6. Double sided rectangle indicates execution of subroutine
7. Circle with alphabet indicates continuation.

2.1.2 DATA TRANSFER OPERATIONS

OPCODE	OPERAND	DESCRIPTION
MOV (Copy from source to destination)	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 altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
MVI (Move immediate 8-bit)	Rd,data M,data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57 or MVI M, 57
LDA (Load accumulator)	16-bit address	The contents of a memory location, specified by a16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034 or LDA XYZ
LDAX (Load accumulator indirect)	B/D Reg.pair	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
STA (Store accumulator direct)	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
STAX (Store accumulator indirect)	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. Example: STAX B
LXI (Load register pair immediate)	Reg. pair 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034

LHLD (Load H and L registers direct)	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040
SHLD (Store H and L registers direct)	16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD 2470

2.1.3 ARTHMETIC OPERATIONS

OPCODE	OPERAND	DESCRIPTION
ADD (Add register or memory content into accumlator)	R M	The contents of the operand (register or memory) are added to 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 HLregisters. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M 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 HLregisters. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
ADC (Add register or memory content into accumulator with carry)	R M	The contents of the operand (register or memory) and the Carry flag are added to 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 addition. Example: ADC B or ADC M
ADI (Add Immediate to accumaltor)	8-bit data	The 8-bit data (operand) is 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: ADI 45

ACI (Add Immediate to accumaltor with carry)	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
DAD (Add register pair to H and L registers)	Register 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 (Subtract register or memory content from accumlator)	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
SBB (Subtact register or memory content from accumulator with borrow)	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 specified by the contents of the HL registers. All flags are modified to reflect the result in accumulator. Example: SBB B or SBB M
SUI (Subtract Immediate to accumaltor)	8-bit data	The 8-bit data (operand) is 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 subtraction. Example: SUI 45
SBI (Subtract Immediate to accumaltor with borrow)	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
INR (Increment register or memory by 1)	R M	The contents of the designated register or memory) are incremented 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
INX (Increment register pair by 1)	Rp	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H
DCR (Decrement register or memory by 1)	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
DCX (Decrement register pair by 1)	Rp	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H
DAA (Decimal adjust accumulator)	None	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

2.1.4 LOGICAL OPERATIONS

OPCODE	OPERAND	DESCRIPTION
CMP (Compare register or memory content with accumlator)	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=1if (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
CPI (Compare Immediate to accumlator)	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
ANA (Logical AND register or memory content with accumlator)	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
ANI (Logical AND immediate with accumlator)	8-bit data	The contents of the accumulator are logically ANDed 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 is reset. AC is set. Example: ANI 86
ORA (Logical OR register or memory content with accumlator)	R M	The contents of the accumulator are logically ORed with the contents of the operand (register/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 and AC are reset. Example: ORA B or ORA M
ORI (Logical OR immediate with accumlator)	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
XRA (Logical EX-OR	R M	The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a

register or memory content with accumlator)		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 and AC are reset. Example: XRA B or XRA M
XRI (Logical EX-OR immediate with accumlator)	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
CMA (Complement Accumlator)	None	The contents of the accumulator are complemented. No flags are affected.Example: CMA
STC (Set Carry Flag)	None	The Carry flag is complemented. No other flags are affected. Example: CMC
CMC (Complement Carry Flag)	None	The Carry flag is set to 1. No other flags are affected. Example: STC

2.1.5 BRANCH OPERATIONS

OPCODE	OPERAND	DESCRIPTION
JMP (Jump unconditionally)	16 bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034 or JMP XYZ
Jump Conditionally JC (Jump on Carry) JNC(Jump on No Carry) JZ (Jump on Zero) JNZ (Jump on No Zero) JP (Jump on Positive) JM (Jump on Minus) JPE (Jump on Parity Even) JPO (Jump on Parity Odd)	16 bit address	The 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. Example: JZ 2034 or JZ XYZ
CALL (Unconditional Subroutine 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 CC (Call on Carry) CNC(Call on No Carry) CZ (Call on Zero) CNZ (Call on No Zero) CP (Call on Positive) CM (Call on Minus) CPE (Call on Parity Even) CPO (Call on Parity Odd)	16 bit address	The 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
RET (Return from subroutine unconditionally)	None	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET
Return from subroutine conditionally RC (Call on Carry) RNC(Call on No Carry) RZ (Call on Zero) RNZ (Call on No Zero) RP (Call on Positive) RM (Call on Minus) RPE (Call on Parity Even) RPO (Call on Parity Odd)	None	The program 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 copied into the program counter, and program execution begins at the new address. Example: RZ
PCHL (Load program counter with HL contents)	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
RST 0-7 (Restart)	None	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: **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 interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are: Interrupt Restart Address TRAP 0024H RST 5.5 002CH RST6.5 0034H RST7.5 003CH
CMA (Complement Accumlator)	None	The contents of the accumulator are complemented. No flags are affected.Example: CMA
STC (Set Carry Flag)	None	The Carry flag is complemented. No other flags are affected. Example: CMC
CMC (Complement Carry Flag)	None	The Carry flag is set to 1. No other flags are affected. Example: STC

2.1.6 WRITING ASSEMBLY LANGUAGE PROGRAMS

---

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

Program 1:

```
MVI A, 32H      Store 32H in the accumulator
STA 4000H       Copy accumulator contents at address 4000H
HLT             Terminate program execution
```

Program 2:

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

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

Program 1:

```
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 4000H into accumulator
STA 2000H     Store the contents of accumulator at address 2000H
MOV A, B      Get the saved contents back into A register
STA 4000H     Store the contents of accumulator at address 4000H Program

HLT           Terminate Program Execution
```

Program 2:

```
LXI H 2000H   Initialize HL register pair as a pointer to memory location 2000H
LXI D 4000H   Initialize DE register pair as a pointer to memory location 4000H.
MOV B, M      Get the contents of memory location 2000H into B register.
LDAX D        Get the contents of memory location 4000H into A register.
MOV M, A      Store the contents of A register into memory location 2000H.
MOV A, B      Copy the contents of B register into accumulator.
STAX D        Store the contents of A register into memory location 4000H.
HLT           Terminate program execution.
```

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

```
LDA 4200H     Get the number
CMA           Complement the number
ADI, 01 H     Add one in the number
STA 4300H     Store the result
```

HLT Terminate program

Flow chart:

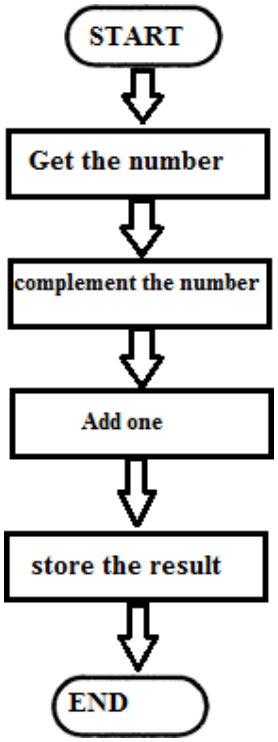


Fig 2.2 Flow Chart

4. Add the contents of memory locations 4000H and 4001H and place the result in memory location 4002H.

Code	Description
LXI H, 4000H	HL points 4000H
MOV A, M	Get first operand
INX H	HL points 4001H
ADD M	ADD second operand
INX H	HL points 4002H
MOV M, A	Store result at 4002H
HLT	Terminate execution program

Sample Problem
(4000H) = 14H
(4001H) = 89H

Result = 14H + 89H = 9DH

Flowchart

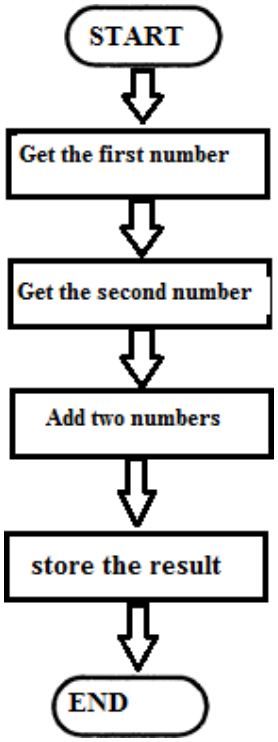


Fig 2.3 Flow Chart

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

Code	Description
LXI H, 4000H	HL points 4000H
MOV A, M	Get first operand
INX H	HL points 4001H
SUB M	Subtract second operand
INX H	HL points 4002H
MOV M, A	Store result at 4002H
HLT	Terminate execution program

Sample problem:
(4000H) = 51H
(4001H) = 19H
Result = 51H – 19H = 38H

Flowchart

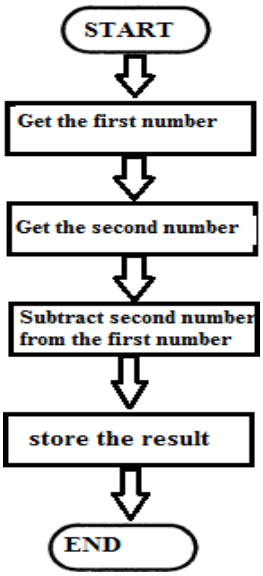


Fig 2.4 Flow Chart

6. 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.

Code	Description
LDA 4201H	Get the Most significant BCD digit
RLC	
RLC	
RLC	
RLC	Adjust the position of the second digit (09 is changed to 90)
ANI FOH	Store result at 4002H
MOV C, A	Make least significant BCD digit zero
LDA 4200H	store the partial result
ADD C	Add lower BCD digit
STA 4300H	Store the result
HLT	Terminate program execution

Sample Problem:
(4200H) = 04
(4201H) = 09
Result = (4300H) = 94

NOTE:
BCD NO.: The numbers "0 to 9" are called BCD (Binary Coded Decimal) numbers. A decimal number 29 can be converted into BCD number by splitting it into two. ie. 02 and 09.

Flow chart:

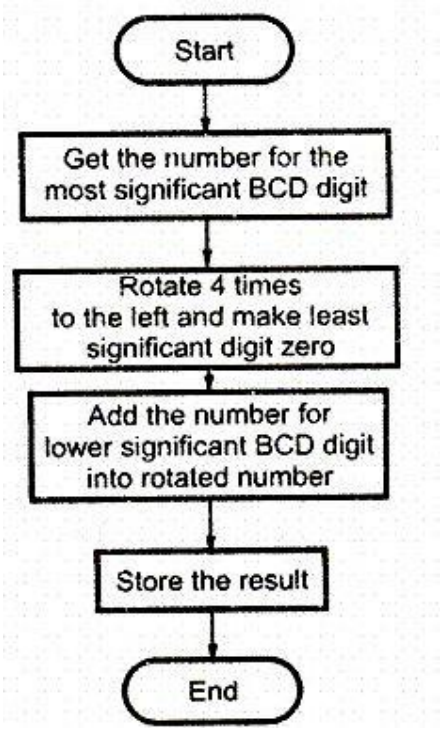


Fig 2.5 Flow Chart

7. 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.

Code	Description
LDA 4200H	Get the packed BCD number
RRC	
RRC	
RRC	
RRC	Adjust higher BCD digit as a lower digit
STA 4301H	store the partial result
LDA 4200H	Get the original BCD number
ANI 0FH	Mask higher nibble
STA 4201H	Store the result
HLT	Terminate program execution

Sample problem:
(4200H) = 58
Result = (4300H) = 08
and (4301H) = 05

Flowchart

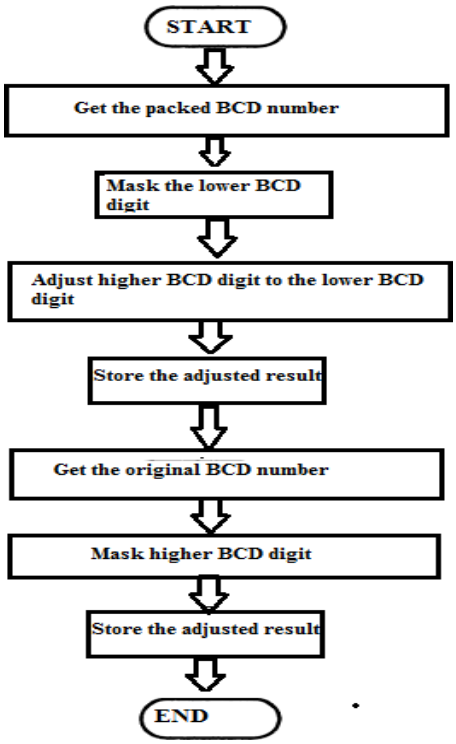


Fig 2.6 Flow Chart

2.1.7 ADDRESSING MODES

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called *Addressing.*

The 8085 has the following 5 different types of addressing.

1. Immediate Addressing
2. Direct Addressing
3. Register Addressing
4. Register Indirect Addressing
5. Implied Addressing

1. Immediate Addressing Mode

In immediate addressing mode, the data is specified in the instruction itself. The data will be apart of the program instruction. All instructions that have 'I' in their mnemonics are of Immediate addressing type. eg. MVI B, 3EH- Move the data 3EH given in the instruction to B register.

2. Direct Addressing

In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory blocks. This type of addressing can be identified by 16- bit address present in the instruction. eg. LDA 1050H- Load the data available in memory location 1050H in accumulator.

3. Register Addressing

In register addressing mode, the instruction specifies the name of the register in which the data is available. This type of addressing can be identified by register names (such as 'A', 'B' in the instruction.) *Eg.* MOV A, B -Move the content of B register to A register.

4. Register Indirect Addressing

In register indirect addressing mode, the instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair. This type of addressing can be identified by letter 'M' present in the instruction. *Eg.* MOV A, M - The memory data addressed by HL pair is moved to A register.

5. Implied Addressing

In implied addressing mode, the instruction itself specifies the type of operation and location of data to be operated. This type of instruction does not have any address, register name, immediate data specified along with it. *Eg.* CMA - Complement the content of accumulator.

2.2 PROGRAMMING TECHNIQUES

2.2.1 LOOPING, COUNTING & INDEXING

Looping-In this technique, the program is instructed to execute certain set of instructions repeatedly to execute a particular task number of times.

Counting-This technique allows programmer to count how many times the instruction/set of instructions are executed.

Indexing-This technique allows programmer to point or refer the data stored in sequential memory location one by one.

2.2.2 ADDITIONAL DATA TRANSFER AND 16 BIT ARITHMETIC INSTRUCTION

The data transfer operations are:

1.**MVI rd , byte** :- This instruction moves the immediate data given after the instruction into register rd.
2.**MOV rd , rs** :- This instruction moves data from register rs to register rd.
3. LXI rp, 16-bit:- This instruction moves 16-bit data or address to register pair.
4. XCHG: The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

2.2.3 ARITHMETIC OPERATION RELATED TO MEMORY

Examples:

1. Calculate the sum of series of numbers. The length of series is in memory location 2200H and series itself begins from 2201H.

(a) Assume the sum to be 8 bit number, so ignore carry. Store the sum at 2300H.

(b) **Assume the sum to be 16 bit number. Store the sum at memory locations 2300H & 2301H.**

(a)

```
        LDA 2200H
        MOV C,A
        SUB A
        LXI H, 2201H
BACK:   ADD M
        INX H
        DCR C
        JNZ
        BACK
        STA
        2300H
        HLT
```

(b)

```
        LDA 2200H
        MOV C,A
        LXI H,2201H
        SUB A
        MOV B, A
Y:      ADD M
        JNC X
        INR B
        INX H
        DCR C
        JNZ Y
        STA 2300H
        MOV A, B
        STA 2301H
        HLT
```

2. Multiply two 8-bit numbers stored in memory location 2200H & 2201H. Store the result in memory location 2300H & 2301H.

```
LDA 2200H
MOV E,A
MVI D,00
LDA 2201H
MOV C,A
```

```
      LXI H, 0000H
   X: DAD D
     DCR C
     JNZ X
     SHLD 2300H
     HLT
```

3. WAP to divide 17 by 4.

```
    MVI D, 00
    MVI E, 00
    MVI A, 17
    MVI B, 04
Y: SUB B
    JC X
   INR D
   JMP  Y
X: ADD B
    MOV E,A
    HLT
```

4. WAP to move a block of data from location A000H to B000H. Assume block size is 10.

```
     LXI H, A000H
     LXI D, B000H
     MVI C, 0AH
X: MOV A, M
    STAX D
    INX H
    INX    D
    DCR   C
    JNZ    X
    HLT
```

5. WAP to move a block of data from location A000H to A005H. Assume block size is 10.

```
     LXI    H,   A009H
     LXI    D,   A00EH
     MVI C, 0AH
X: MOV A, M
      STAX D
      DCX H
    DCX D
    DCR C
    JNZ X
    HLT
```

6. 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.

Sample problem:
(4000H)= 15H
(4001H)=1CH
(4002H)=B7H
(4003H)= 5AH
Result = 1C15 + 5AB7H =76CCH
(4004H) = CCH
(4005H) = 76H

Source Program 1:
LHLD 4000H : Get first I6-bit number in HL
XCHG : Save first I6-bit number in DE
LHLD 4002H : Get second I6-bit number in HL
MOV A, E : Get lower byte of the first number
ADD L : Add lower byte of the second number
MOV L, A : Store result in L register
MOV A, D : Get higher byte of the first number
ADC H : Add higher byte of the second number with CARRY
MOV H, A : Store result in H register
SHLD 4004H : Store I6-bit result in memory locations 4004H and 4005H.
HLT : Terminate program execution

Source program 2:

LHLD 4000H : Get first I6-bit number
XCHG : Save first I6-bit number in DE
LHLD 4002H : Get second I6-bit number in HL
DAD D : Add DE and HL
SHLD 4004H : Store I6-bit result in memory locations 4004H and 4005H.
HLT : Terminate program execution

7. 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.

Sample problem:
(4000H) = 19H
(400IH) = 6AH
(4004H) = I5H
(4003H) = 5CH
Result = 6A19H – 5C15H = OE04H
(4004H) = 04H
(4005H) = OEH

Flowchart

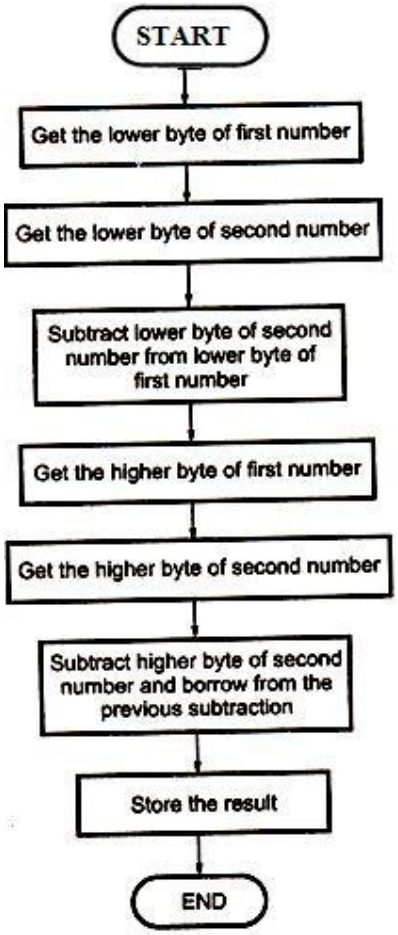


Fig2.7 Flow Chart

Source program:

```
LHLD 4000H     Get first 16-bit number in HL
XCHG           Save first 16-bit number in DE
LHLD 4002H     Get second 16-bit number in HL
MOV A, E       Get lower byte of the first number
SUB L          Subtract lower byte of the second number
MOV L, A       Store the result in L register
MOV A, D       Get higher byte of the first number
SBB H          Subtract higher byte of second number with borrow
MOV H, A       Store l6-bit result in memory locations 4004H and 4005H.
SHLD 4004H     Store l6-bit result in memory locations 4004H and 4005H.
HLT            Terminate program execution.
```

8. WAP to divide 16 bit number stored in memory location 2200H & 2201H by the 8 bit number stored at memory location 2202H. Store the quotient in memory location 2300H & 2301H and remainder in 2302H & 2303H.

```
LHLD 2200H
LDA 2202H
MOV C, A
```

```
    LXI D, 0000H
Y: MOV A, L
    SUB C
    MOV L, A
    JNC X
    DCR H
X: INX D
  MOV A, H
  CPI 00H
  JNZ Y
  MOV A, L
  CMP C
  JNC Y
  SHLD 2302H
  XCHG
  SHLD 2300H
  HLT
```

9. WAP to find the largest number in a block of data. The length of block is in memory location 2200H and the block itself begins from location 2201H. Store the maximum number in 2300H.

```
LDA 2200H
MOV C, A
XRA A
LXI H, 2201H
X: CMP M
JNC Y
MOV A, M
Y: INX H
DCR C
JNZ X
STA 2300H
HLT
```

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

Source program:

```
        MVI B, 09          Initialize counter
START   LXI H, 2200H       Initialize memory pointer
        MVI C, 09H         Initialize counter 2

BACK:   MOV A, M           Get the number
        INX H              Increment memory pointer
        CMP M              Compare number with next number
        JC SKIP            If less, don't interchange
        JZ SKIP            If equal, don't interchange
        MOV D, M
```

```
        MOV M, A
        DCX H
        MOV M, D
        INX H              : Interchange two numbers
SKIP:DCR C                 : Decrement counter 2
        JNZ BACK           : If not zero, repeat
        DCR B              : Decrement counter 1
        JNZ START
        HLT                : Terminate program execution
```

Flow chart

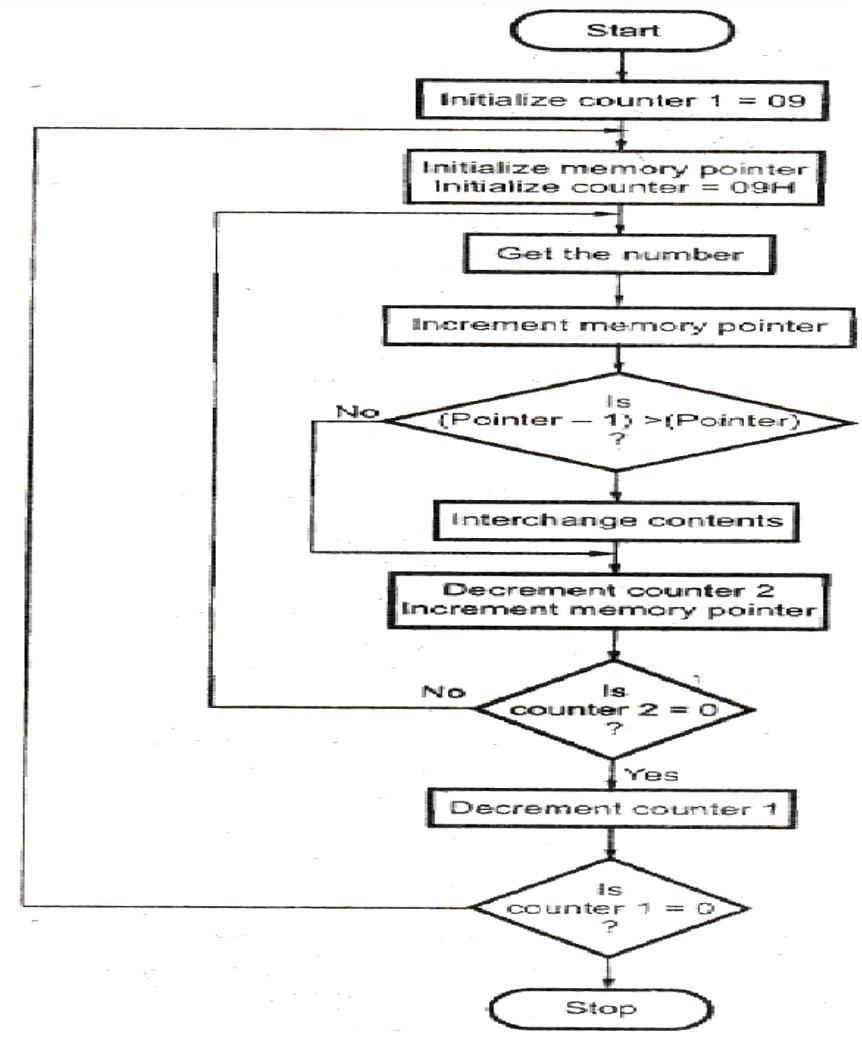


Fig 2.8 Flow Chart

2.2.6 UNIVERSITY QUESTIONS RELATED TO THE TOPIC

Two-mark questions

Q-1. Write a program to find 2's compliment of a 8-bit number in 8085.
Q-2. What is the difference between MOV and MVI instruction.
Q-3. Explain Implicit addressing mode with example.

Q-4. Explain the function of SUB A and ORA A instruction.
Q-5. State the difference between RAL and RRC instructions

Five-mark questions

Q-6. (i) Specify the address of the output port, and explain the type of numbers that can be displayed at output port.

```
        MVI A, BYTE1
        ORA A
        JP OUTPRT
        XRA A
OUTPRT: OUT F2 H
        HLT
```

(ii) If BYTE1 = 92H, what is the output at PORT F2H?

Q-7. Identify the register contents and flag status as the following instructions are execute:

	A	S	Z	CY
SUB A				
MOV B,A				
DCR B				
INR B				
SUI 01 H				
HLT				

Q-8. Write an assembly language program for the division of 8 bit number.

Ten-mark questions

Q-9. Define addressing modes and how many addressing modes are available in the 8085 microprocessors. Explain each with an example.
Q-10. Explain the function of instructions in the form of addressing modes, machine cycles and no. of T state requirement XCHG, DAD, SHLD, LDAX, SBB, ADD, INX, DAA, DCR, SUI.
Q-11. Write an 8085-assembly language program for the addition and subtraction of two 8-bit numbers.
Q-12. Write an assembly language program to find the largest number in a series of number stored from location 2000 H to 200A H. Store the result at location 3000 H. Explain the program with a relevant Flow Chart.

GATE questions

Q-13. How many Machine cycles are required to execute the following 8085 instructions?
(I.) LDA 3000H (II.) LXI D, F0F1H
Q-14. The following five instructions were executed on an 8085 microprocessor.

```
MVI A, 33H
MVI B, 78H
ADD B
ANI 32H
HLT
```

What is the value of the Accumulator after execution of the fourth instruction?

Q-15. Write an 8085-microprocessor program, which calculates the product of two 8-bit numbers stored in registers B and C?

Q-16. An 8085-assembly language program is given below. Assume that the carry flag is initially unset. The content of the accumulator and B register after the execution of the program is

```
MVI A,07H
ADI F2H
MOV B,A
SUI C2H
ADD B
HLT
```

Q-17. What will be the output of the A and Carry flag after the execution of the SUB A and ADD A instruction? If the contents of the A and Carry flag are A7 (in hex) and 0 respectively.

Q-18. The Following program is executed

```
    MVI A, 05H
    MVI B, 05H
PTR:ADD B
    DCR B
    JNZ PTR
    ADI 03H
    HLT
```

At the end of program what will be the content of Accumulator.

2.3 COUNTER AND TIME DELAYS

2.3.1 DELAY USING SINGLE REGISTER

A loop counter is set up by loading a register with certain value. Then using DCR and INR the contents of register are modified. A loop is set up with a conditional jump that looks back or not depending on whether the count has reached the termination count. The operation of a counter can be described using flow chart as shown in Figure 3.4

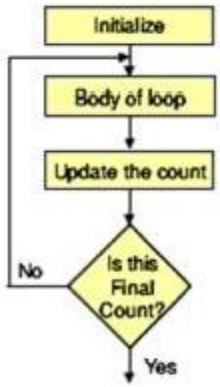


Fig 2.9 Flow Chart

Implementing loop using DCR instruction:

```
MVI C, 15H
LOOP:DCR C
     JNZ LOOP
```

2.3.2 DELAY USING REGISTER PAIR

Using a register pair as a loop counter:

```
      LXI B,FFFFH
LOOP: DCX B
      MOV A,B
      ORA C
      JNZ LOOP
      RET
```

2.3.3 DELAY USING NESTED LOOP

Using a single register, one can repeat a loop for maximum 255 times.

```
      MVI D, FFH
LOOP1: LXI B, 1000H
```

```
LOOP:DCX B
     MOV A, C
     ORA B
     JNZ LOOP
     DCR D
    JNZ LOOP1
```

2.3.4 DELAY ROUTINE PROCESS

Delay Routine

Delay routines are subroutines used for maintaining the timings of various operations inmicroprocessor.In control applications, certain equipment needs to be ON/OFF after a specified time delay. In some applications, a certain operation has to be repeated after a specified time interval. In such cases, simple time delay routines can be used to maintain the timings of the operations.

DELAY ROUTINE PROCESS

A delay routine is generally written as a subroutine (It need not be a subroutine always. It can beeven a part of main program). In delay routine a count (number) is loaded in a register of microprocessor. Then it is decremented by one and the zero flag is checked to verify whether thecontent of register is zero or not. This process is continued until the content of register is zero.When it is zero, the time delay is over and the control is transferred to main program to carry out the desired operation.

The delay time is given by the total time taken to execute the delay routine. It can be computed by multiplying the total number of T-states required to execute subroutine and the time for one T-state of the processor. The total number of T-states can be computed from the knowledge of T-states required for each instruction. The time for one T-state of the processor is given by the inverse of the internal clock frequency of the processor.

For example, if the 8085 microprocessor has 5 MHz quartz crystal then, the internal clock frequency = 5 / 2 = 2.5 MHz
Time for one T-state= $1 / 2.5 \times 10^6 = 0.4\mu sec$

- For small time delays (< 0.5 msec) an 8- bit register can be used.
- For large time delays (< 0.5 Sec) l6-bit register should be used.
- For very large time delays (> 0.5 sec), a delay routine can be repeatedly called in the main program.

The disadvantage in delay routines is that the processor time is wasted. An alternate solution is to use dedicated timer like 8253/8254 to produce time delays or to maintain timings of various operations.

2.3.1 EXAMPLES OF DELAY PROGRAMS

---

1. Write a delay routine to produce a time delay of 0.5 msec in 8085 processor-based system whose clock source is 6 MHz quartz crystal.

Solution

The delay required is 0.5 msec, hence an 8-bit register of8085 can be used to store a Count value and then decrement to zero. The delay routine is written as a subroutine as shown below.

Delay routine

```
        MVI D, N        ; Load the count value, N in D-register.
Loop:   DCR D           ; Decrement the count.
        JNZ Loop        ; If count is zero go to
        RET             ; Return to main program.
```

The following table shows the T-state required for execution of the instructions in the subroutine.

Table 3.1 Calculation to find the count value

Instruction	*T-State required for execution of aninstruction*	*Number of times theinstruction is executed*	*Total T-States*
CALL addr16	18	1	18 x 1 = 18
MVI D, N	7	1	7 x 1 = 7
DCR D	4	N times	4 x N = 4N
JNZ LOOP	10	(N-1) times	10 x (N-1) = 10N – 10
RET	7	1	7 x 1 = 7
	10	1	10 x 1 = 10
TOTAL T-STATES FOR DELAY SUBROUTINE			14N + 32

Calculation to find the count value, N:
External clock frequency = 6 Mhz
Internal clock frequency = External Frequency / 2= 6 / 2= 3 Mhz
Time period for 1 T-State= 1 / Internal clock frequency= 1 / $3x10^6$= 0.333µS
No. of T-states requiredfor delay of 0.5mS = Required time delay / Time for one T-state
= 0.5mS / 0.333µS= 1500.10≈ 1500 = 1500_{10}

From above table, we know that; 14N + 32 = 1500
N = (1500 – 32) / 14 = 104.857_{10}≈ 105_{10}= 69H
Therefore by replacing the count value, N by 69H in the above program , a delay of 0.5mSec can be produced

2. Write an ALP for 8085 to count from AAH to 00H, with a time delay of 2ms for each count. Assume the external frequency given to the processor is 2MHz.

Internal Frequency in 8085 = External frequency / 2 ie.,= 2Mhz / 2= 1Mhz1
T-State= 1 / f (internal frequency)= 1µS

Main program for counting from AA to 00

```
        MVI C, AAH
Loop:CALL Delay
        DCR C
        JNZ Loop
        HLT
```

Delay program for delay of 2ms

```
Delay:MVI D, 4AH
Next:  NOP
        NOP
        NOP
        NOP
        DCR D
        JNZ Next
        RET
```

ILLUSTRATIVE PROGRAM:

Hexadecimal counter

Hex-up counter counts from FFH to 00H

```
        MVI B, 00H
NEXT:   DCR B
        MVI C, 05
DELAY:  DCR C
        JNZ DELAY
        MOV A, B
        OUT PORT1
        JMP NEXT
```

zero-to-nine/(module ten) counter

```
START: MVI B, 00H
          MOV A, B
DPLAY: OUT PORT1
          LXI H, 16 BIT
LOOP: DCX H
        MOV A, L
        ORA H
        JNZ LOOP
        INR B
        MOV A, B
        CPI 0AH
        JNZ DPLAY
        JZ START
```

Generating Pulse Waveforms

WAP to generate square wave from SOD pin.

```
        LXI SP, 27FFH
        LXI B, 1388H
BACK: MVI A, C0H
        SIM
        CALL DELAY
        MVI A, 40H
        CALL DELAY
        DCX B
        MOV A, C
        ORA B
        JNZ BACK
        HLT
      MVI D, FFH
DELAY: DCR D
       JNZ DELAY
       RET
```

OR

Generating Pulse waveform

```
        MVI D, AAH
X:      MOV A, D
        RLC
        MOV D, A
        ANI 01H
        OUT PORT1
        MVI B, COUNT
Y:      DCR B
        JNZ Y
        JMP X
```

Debugging counter and time delay

It is designed to count from 100_{10} to 0 in hex continuously with a 1 second delay between each count. The delay is set up using two loops- a loop within a loop. The inner loop is expected to provide approximately 100ms delay and is repeated 10 times, using outer loop to provide a total delay of 1 sec. the clock period of system is 330ns.

Program:

```
MVI A, 64H                    7
X: OUT PORT1                  10
Y:MVI B, 10H                  7
Z:LXI D, X                    10
DCX D                         6
```

NOP	4
NOP	4
MOV A, D	4
ORA E	4
JNZ Z	10/7
DCR B	4
JZ Y	10/7
DCR A	4
CPI 00H	7
JNZ X	10/7

Delay in loop1=32T X count x $330x10^{-9}$ 100ms =32T X count x $330x10^{-9}$

Count=9470

2.4.4 UNIVERSITY QUESTIONS RELATED TO THE TOPIC

Ten-mark questions

Q-1. Write an assembly language program to count from 0 to 9 with one second delay between each count. At the count of 9, the counter should reset itself to 0 and repeat the sequence continuously. Display each count at one of the Output ports with system clock frequency 1MHz.

Q-2. Write a program continuously in hexadecimal from FFH to 00H in a system with 0.5us clock period. Use reg. C to set up a 1ms delay between each count and display the no. at one of the output ports.

2.4 STACK AND SUBROUTINE

2.4.1 INTRODUCTION

STACK

It is a part of memory, reserved in RAM, used to temporarily store information during execution of program. Starting address of stack is loaded in stack pointer (a 16 bit register). The address pointed by SP is known as top of stack, which is always an empty memory location. Stack can be defined anywhere in RAM. But generally it is initialized from highest address of RAM to avoid any data loss. The figure shows the stack initialization. Stack is a LIFO type of memory

When information is stored onto stack, the SP decrements the pointer to lower empty address.

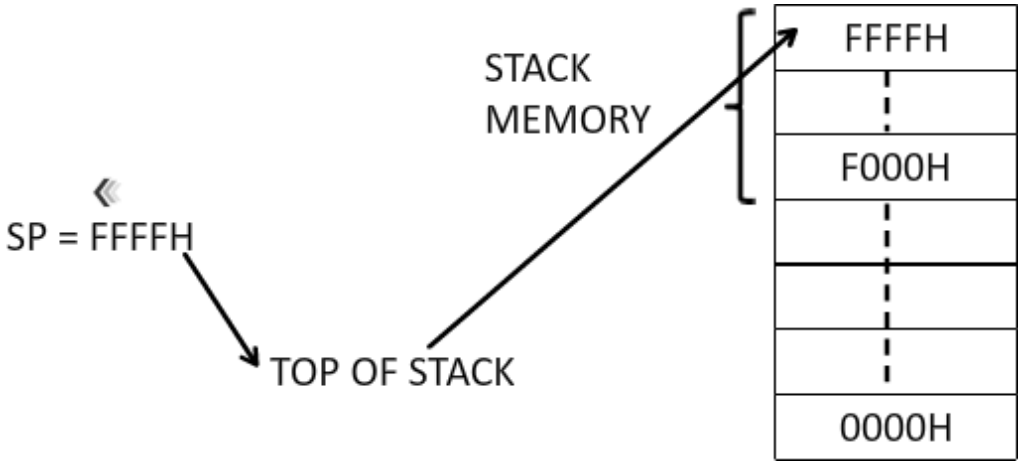


Fig 2.10 Stack Process

When information is read from stack, the SP register increments to higher empty address.

SUBROUTINE

In computers, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task have to be performed. A subroutine is often coded so that it can be started (called) several times and from several places during one execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the call, once the subroutine's task is done. It is implemented by using Call and Return instructions.

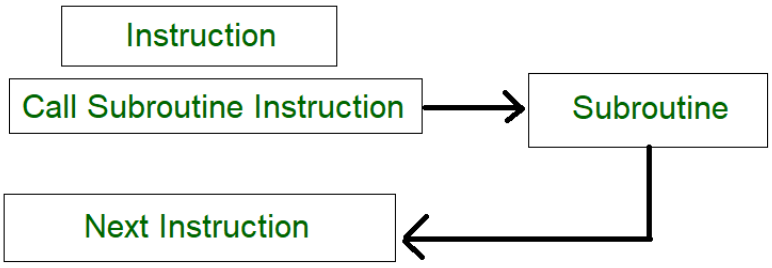


Fig 2.11 Subroutine Process

2.4.2 INSTRUCTION RELATED TO STACK AND SUBROUTINE

OPCODE	OPERAND	DESCRIPTION
SPHL (Copy H and L registers to the stack pointer)	None	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.Example: SPHL
XTHL (Exchange H and L with top of stack)	None	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL
PUSH (Push register pair onto stack)	Rp	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high- order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low- order register (C, E, L, flags) are copied to that location. Example: PUSH B or PUSH A
POP (Pop register pair onto stack)	Rp	The contents of the stack designated in the operand are copied onto the register pair in the following sequence. The stack pointer register is decremented and the contents of the high- order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low- order register (C, E, L, flags) are copied to that location. Example: POP B or POP D
CALL (Unconditional Subroutine 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 CC (Call on Carry) CNC(Call on No Carry) CZ (Call on Zero)	16 bit address	The 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

CNZ (Call on No Zero) CP (Call on Positive) CM (Call on Minus) CPE (Call on Parity Even) CPO (Call on Parity Odd)		the next instruction after the call (the contents of the program counter) is pushed onto the stack. Example: CZ 2034 or CZ XYZ
RET (Return from subroutine unconditionally)	None	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET
Return from subroutine conditionally RC (Call on Carry) RNC(Call on No Carry) RZ (Call on Zero) RNZ (Call on No Zero) RP (Call on Positive) RM (Call on Minus) RPE (Call on Parity Even) RPO (Call on Parity Odd)	None	The program 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 copied into the program counter, and program execution begins at the new address. Example: RZ

2.4.3 ADVANCED SUBROTUINE CONCEPTS

Subroutine nesting is a common Programming practice In which one Subroutine calls another Subroutine.

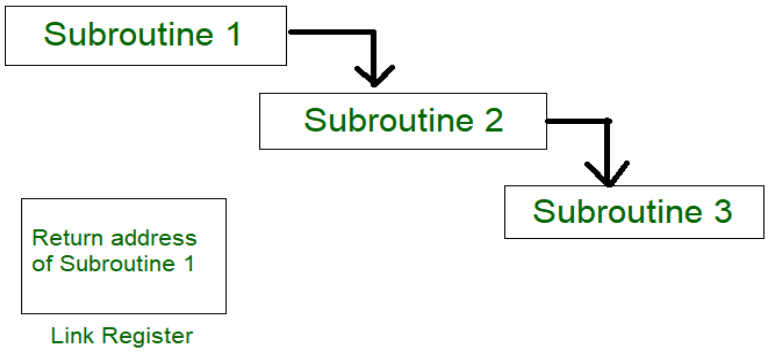


Fig 2.12 Subroutine Nesting Process

From the above figure, assume that when Subroutine 1 calls Subroutine 2 the return address of Subroutine 2 should be saved somewhere. So if the link register stores the return address of Subroutine 1 this will be

(*destroyed/overwritten*) by the return address of Subroutine 2. As the last Subroutine called is the first one to be returned (Last in first out format). So stack data structure is the most efficient way to store the return addresses of the Subroutines.

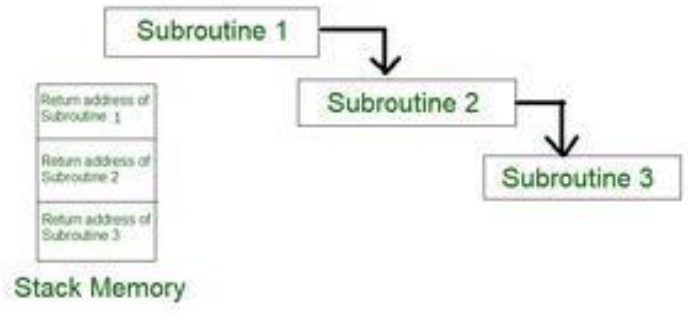


Fig 2.13 Return Address of Subroutine stored in Stack

2.4.4 CALL AND JUMP INSTRUCTION COMAPRISON

CALL Instruction: Execution of a CALL instruction will transfer the program control from existing program to another program. i.e., Sub program specified by the 16-bit address in CALL instruction will be executed. The called program should have RET – return instruction as its last instruction. Time taken for its execution is 9 / 18T

Main_____ addr16:_____

_____ _____

_____ _____

_____

CALL addr16 _____

_____

_____

_____

RET

JMP Instruction: Execution of a JMP instruction will transfer the program control from one location toanother location within the same program. Time taken for its execution is 7 / 10T

Main_____

_____

_____

JMP addr16

_____

_____

addr16:_____

_____

2.4.5 UNIVERSITY QUESTIONS RELATED TO THE TOPIC

Two-mark questions

Q-1. What is the position of Stack Pointer after PUSH instruction is executed?
Q-2. Explain the purpose of Stack Pointer.
Q-3. Stack of 8085 works on ……..Principle.

Five-mark questions

Q-4. Write a program of swapping the content of register pair with the help of stack.
Q-5. Explain the operation of PUSH and POP instruction with example.

Ten-mark questions

Q-6. Explain Conditional CALL instruction and nested Subroutine concept in 8085.
Q-7. Write a short note on STACK and SUBROUTINE.

2.5 THE 8085 INTERRUPTS

2.5.1 INTRODUCTION

Interrupt is a signal send by an external device to the processor, to the processor to perform a particular task or work. Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.

When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt then the processor suspends its current activity and executes an interrupt service subroutine to complete the data transfer between the peripheral and processor. After executing the interrupt service routine the processor resumes its current activity. This type of data transfer scheme is called interrupt driven data transfer scheme.

2.5.2 TYPES OF INTERRUPTS

2.5.2.1 HARDWARE & SOFTWARE INTERRUPTS OF 8085

The interrupts are classified into software interrupts and hardware interrupts.

•The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, lf a software interrupt instruction is encountered, then the processor executes an interrupt service routine (ISR).
•An external device initiates the hardware interrupts by placing an appropriate signal at the interrupt pin of the processor. If the interrupt is accepted, then the processor executes an interrupt service routine (ISR).

Software Interrupts Of 8085

The software interrupts are program instructions. When the instruction is executed, the processor executes an interrupt service routine stored in the vector address of the software interruptinstru ction. The software interrupts of 8085 are RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST6 and RST7.

Interrupt	Vector address
RST0	0000H
RST1	0008H
RST2	0010H
RST3	0018H
RST4	0020H
RST5	0028H
RST6	0030H
RST7	0038H

Table 3.2 Vector address for software interrupts

The software interrupt instructions are included at the appropriate (or required) place in the main program. When the processor encounters the software instruction, it pushes the content of PC(Program Counter) to stack. Then loads the Vector address in PC and starts executing the Interrupt Service Routine (ISR) stored in this vector address. At the end of ISR, a return instruction – RET will be placed. When the RET instruction is executed, the processor POP the content of stack to PC. Hence the processor control returns to the main program after servicing the interrupt.
Execution of ISR is referred to as servicing of interrupt. All software interrupts of 8085 are vectored interrupts. The software interrupts cannot be masked and they cannot be disabled. The software interrupts are RST0, RST1,…., RST7 (8 Nos).

Hardware Interrupts Of 8085

An external device, initiates the hardware interrupts of 8085 by placing an appropriate signal at the interrupt pin of the processor. The processor keeps on checking the interrupt pins at the second T-state of last machine cycle of every instruction. If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupt. The acceptance of the interrupt is acknowledged by sending an INTA signal to the interrupted device. The processor saves the content of PC

(program Counter) in stack and then loads the vector address of the interrupt in PC. (If the interrupt is non-vectored, then the interrupting device has to supply the address of ISR when it receives INTA signal). It starts executing ISR in this address. At the end of ISR, a return instruction, RET will be placed. When the processor executes the RET instruction, it POP the content of top of stack to PC. Thus the processor control returns to main program after servicing interrupt. The hardware interrupts of 8085 are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR

2.5.2.2 VECTORED & NON-VECTORED INTERRUPTS OF 8085

Further, the interrupts may be classified into Vectored and Non-Vectored Interrupts.

1. **Vectored interrupt-** In vectored interrupts, the processor automatically branches to the specific address in response to an interrupt.
2. **Non-vectored interrupt**- But in non-vectored interrupts the interrupted device should give the address of the interrupt service routine (ISR).

In vectored interrupts, the manufacturer fixes the address of the ISR to which the program control is to be transferred. The vector addresses of hardware interrupts are given in table as shown below:

Interrupt	Vector address
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH
TRAP	0024H

Table 3.3 Vector address for interrupts

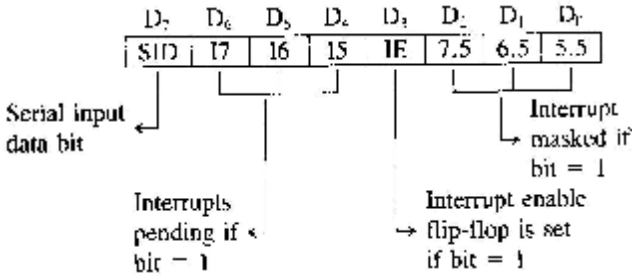
- The TRAP, RST 7.5, RST 6.5 and RST 5.5 are vectored interrupts. The INTR is a non-vectored interrupt. Hence when a device interrupts through INTR, it has to supply the address of ISR after receiving interrupt acknowledge signal.

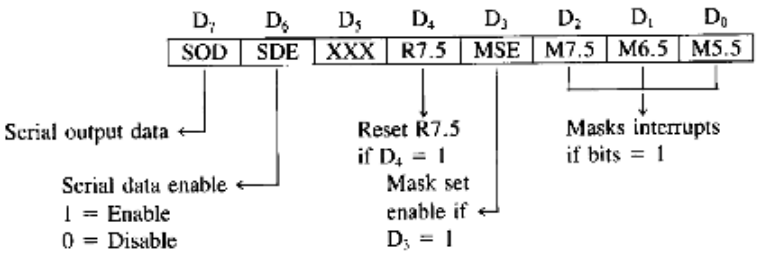
- The TRAP interrupt is edge and level sensitive. Hence, to initiate TRAP, the interrupt signal has to make a low to high transition and then it has to remain high until the interrupt is recognized.

- The RST 7.5 interrupt is edge sensitive (positive edge). To initiate the RST 7.5, the interrupt signal has to make a low to high transition an it need not remain high until it is recognized.

- The RST 6.5, RST 5.5 and INTR are level sensitive interrupts. Hence for these interrupts the interrupting signal should remain high, until it is recognized.

2.5.2.3 MASKABLE & NON-MASKABLE INTERRUPTS OF 8085

OPCODE	OPERAND	DESCRIPTION
NOP (No Operation)	None	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
HLT (Halt and enter wait state)	None	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
DI (Disable interrupts)	None	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
EI (Enable interrupts)	None	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI
RIM (Read interrupt mask	None	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations. Example: RIM 
SIM(Set Interrupt Mask)	None	This is a multipurpose instruction and used to implement the8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.



- SOD — Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX — Don't Care
- R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE — Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- M7.5 — D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- M6.5 — D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- M5.5 — D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

- The hardware vectored interrupts are classified into maskable and non-maskable interrupts.
- TRAP is non-maskable interrupt
- RST 7.5, RST 6.5 and RST 5.5 are maskable interrupt. Masking is preventing the interrupt from disturbing the main program. When an interrupt is masked the processor will not accept the interrupt signal.
- The interrupts can be masked by moving an appropriate data (or code) to accumulator and then executing SIM instruction. (SIM - Set Interrupt Mask). The status of maskable interrupts can be read into accumulator by executing RIM instruction (RIM - Read Interrupt Mask).All the hardware interrupts, except TRAP are disabled, when the processor is reset. They can also be disabled by executing Dl instruction. (Dl-Disable Interrupt).
- When an interrupt is disabled, it will not be accepted by the processor. (i.e., INTR, RST 5.5, RST 6.5 and RST 7.5 are disabled by DI instruction and upon hardware reset).
- To enable (to allow) the disabled interrupt, the processor has to execute El instruction (El-Enable Interrupt).

2.5.3 INSTRUCTIONS RELATED TO THE INTEERUPTS

2.5.4 UNIVERSITY QUESTIONS RELATED TO THE TOPIC

Two-mark questions

Q-1. Enlist the number and name of hardware interrupt (which require an external signal to interrupt) present in an 8085 microprocessor.

Q-2. What are the differences between vectored and non-vectored interrupt? What are the addresses of hardware- vectored interrupts in 8085?

Q-3. Differentiate Maskable and Non-Maskable Interrupts.

Five-mark questions

Q-4. What is process followed for Interrupt.

Q-5. Give the classification of Interrupt in all possible types.

Ten-mark questions

Q-6. What are interrupts? Give the classification of interrupts. Explain the hardware and software interrupts used in 8085.

GATE questions

Q-7. In a microcomputer, wait states are used to……

Q-8. In a microprocessor, when a CPU is interrupted, it…….

Q-9. In an 8085μP system, the RST instruction will cause an interrupt……