INTRODUCTION TO MICRO COMPUTER&**MICROPROCESSORS**

Introduction: A Microprocessor is a multipurpose programmable logic device which reads the binary data according to the instructions and gives the results as output. So, you can understand the Microprocessor as a programmable digital device, which can be used for both data processingand control applications. In view of a computer student, it is the CPU of a Computer or heart ofthe computer. A computer which is built around a microprocessor is called a microcomputer. Amicrocomputer system consists of a CPU (microprocessor), memories (primary and secondary)and I/O devices as shown in the block diagram in Fig 1. The memory and I/O devices are linkedby data and address (control) buses. The CPU communicates with only one peripheral at a timeby enabling the peripheral by the control signal. For example to send data to the output device,the CPU places the device address on the address bus, data on the data bus and enables theoutput device. The other peripherals that are not enabled remain in high impedance state calledtri-state.

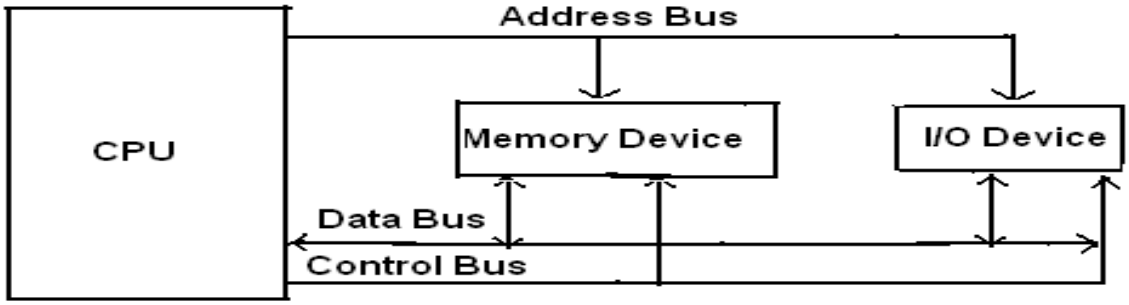


Fig.1 Block diagram of a Microcomputer

The Intel 8085 Microprocessor:

Intel 8085A is a single chip 8-bit N-channel microprocessor which works at +5V DC powersupply. It is a 40 pin IC available as a DIP (Dual Inline Package) chip. 8085A can operate with a3MHZ single phase clock and 8085A-2 version can operate at a maximum frequency of 5MHZ.This 8085 is an enhanced version of its predecessor the 8080A. Its instruction set is upwardcompatible with that of the 8080A. 8085A has an on-chip clock generator with external crystal,LC or RC network. This 8085 microprocessor is built with nearly 6200 transistors. The enhancedversion of 8080 is the Intel 8085AH. It is an N channel depletion load, silicon gate (HMOS) 8-bitprocessor. Here 3MHZ, 5MHZ and 6MHZ selections are available. It has 20% lower powerconsumption than 8085A for 3MHZ and 5MHZ. Its instruction set is 100% software compatiblewith the 8085A. It is also 100% compatible with 8085A.

Central Processing Unit (CPU)

The Central Processing Unit of any microcomputer is the microprocessor. Hence microprocessor is also known as the heart of the computer. The CPU performs the variousactivities in response to a set of instructions called a program. Programs are stored in the memory. The CPU reads in data control signals (instructions) through the input ports and executes one instruction at a time. So, generally speaking, a microprocessor is nothing but the CPU .The Intel 8085 CPU is an 8-bit device with a clock speed of 3 - 5 MHZ. It has 80 basic instructions and 246 op-codes. Its clock cycle is 320 ns. The time for the clock cycle of Intel 8085 is 200 ns. The block diagram of 8085 microprocessor is shown in **Fig 2.** The 8085 CPU consists of three major sections, They are:

(i). Arithmetic and logic unit (ALU) (ii).Registers (iii).Timing and Control unit.

Arithmetic and logic unit (ALU)

The ALU performs all the arithmetic and logical operations like addition, subtraction, complementing, logical AND, logical OR, logical Exclusive OR, incrementing and decrementing, rotate, shift and clear. An ALU is made of many logic gates and adders etc. The arithmetic and logic unit consists of the following units

(a).Accumulator (A).
(b).Temporary register.
(c).Flag register.

(a) Accumulator (A):

It is an 8-bit register which is treated as a special function register. All the I/O data transfers between 8085 and I/O devices are performed via accumulator. One of the operands for arithmetic operations in ALU is from the accumulator. After performing the arithmetic operations the result is stored back in accumulator. It is from the accumulator only, the data is sent out to an output device. Similarly, the data from an input device is read only through the accumulator. The data in the accumulator alone can be rotated or shifted. No other register can be used for these operations. Certain instructions like DAA are performed using only accumulator. So, many times the Accumulator register is treated as a default register.

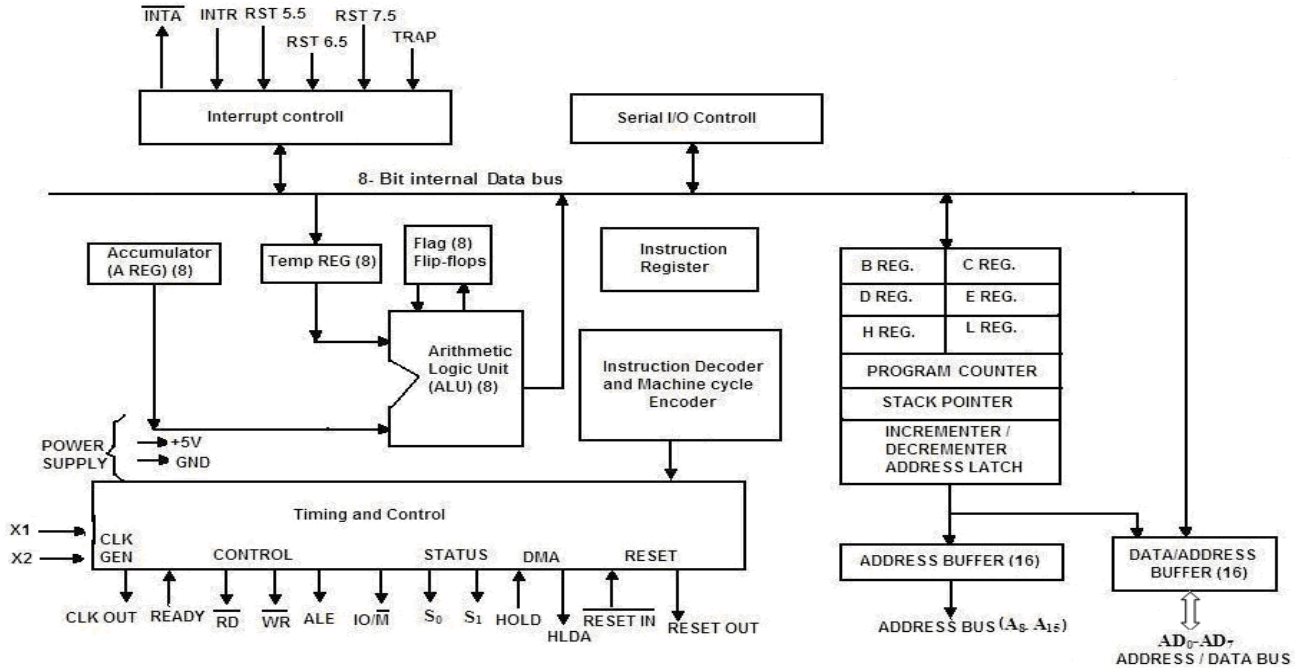


Figure 2 .The Block Diagram of 8085 Microprocessor

b) Temporary register:

This is an 8-bit register which is not accessible to the user. This register is used by the microprocessor to load the second operand during arithmetic/logical operations in ALU. The final result is stored in the Accumulator and the flags are set or reset according to the result of the
Operation.

(**c).Flag register**: The flag register is an 8- bit register which generally reflect data conditions in the accumulator with certain exceptions. Hence this flag register is also known as Status register. Though this flag register is an eight bit register, it contains only 5 flag bits and the remaining three bits are undefined as shown in Fig.3.3 in the Flag register each flag bit is a Flip-Flop. i.e., the bit may be either in the flip state or flop state.

S - Sign Flag

After execution of an arithmetic and logic operation, if bit D7 of the result (Normally in the accumulator) is 1, the sign flag is set. This Flag is used with signed numbers.For example in agiven byte, if D7 is 1, the number is treated as a negative number. Else (if it is zero), it is viewedas a positive. In arithmetic operations with signed numbers bit D7is reserved for indicating thesign and theremaining seven bits are used to denote the magnitude of the number.

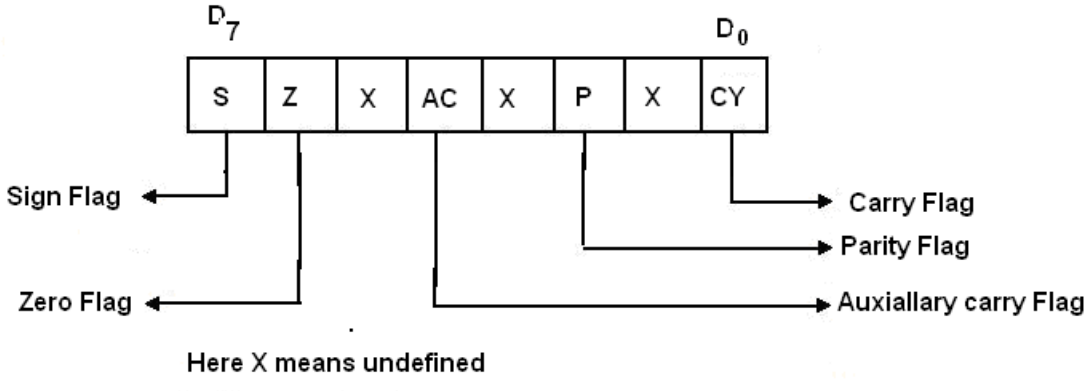


Figure 3. Flag Register

Z - Zero Flag

This Flag is set (made 1) if the result after any arithmetic operation is zero, and the flag is reset (made 0) if the result is not zero. So, this flag is set or reset based on the results in the accumulator as well as in the other registers.

AC – Auxiliary carry Flag

In this arithmetic operation, when a carry is generated by and passed on to bit 4 , the AC flag is set. This flag is used internally for BCD arithmetic and is not available for the programmer to change the sequence of a program with a jump instruction. But the Z and CY flags can be used for this purpose.

P-Parity Flag

If the result after an arithmetic and logical operation has an even number of 1s, this parity flag is set to 1 otherwise (if number of 1s is odd) the flag is reset (made0). For example the data byte 10111101 has even parity and the data byte 10011011 has odd parity. So P bit=0. **Carryflag**

After an arithmetic operation, like addition, subtraction if there exists a carry or barrow, this flag CY is set to 1 else it is reset (made0)

Example : Let us consider the addition of two binary numbers 11011001 and 11101101 and check the Flag register.

Example: Let us consider the addition of two binary numbers 11011001 and 11101101 and Check the Flag register

D7 D0
1 1 0 1 1 0 0 1
1 1 1 0 1 1 0 1
1 1 0 0 0 1 1 0

In the result, the sum is not zero, So Z-Flag is reset (Z=0). There is a carry from the third bit to fourth bit. So AC Flag is set (AC=1). In the result, the no. of 1s is even. So, parity is even (P=1). After addition, there is a carry. So carry Flag is set (CY=1). The Flag register contents after addition are shown below:

D7 D0

1	0	X	1	X	1	X	1

Register Organization

The 8085 microprocessor has different types of registers. It includes six , 8 – bit registers (B, C, D. E, H and L), one 8-bit Accumulator and two 16-bit registers (SP and PC). Also there are two 8-bit temporary registers W and Z.

The various registers of 8085 are classified into three types. They are
(i).Temporary registers.
(ii).General purpose registers
(iii).Special purpose registers

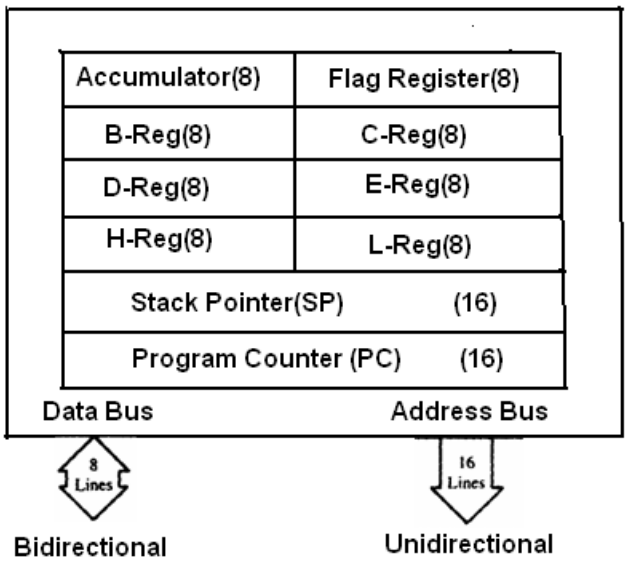
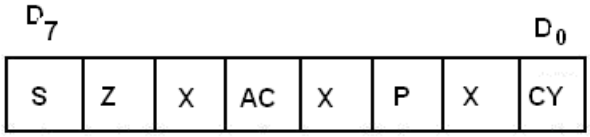


Figure 4. Register organization

Special purpose registers:

The Special purpose registers, as their name indicates, are used for some specificpurpose. The Special purpose registers are Accumulator (A), Flag Register, Instruction Register (IR), Program Counter (PC) and Stack Pointer (SP).Accumulator (Register A): It is an 8-bit tri-state register. It is mainly used for arithmetic, logic, loadand store operations. It is also used in I/O operations. In most of operations, the result is storedin Accumulator after execution.Flag Register: It is an 8-bit register, which consists of only

five flags. Each flag bit is a flip flop thatindicates either a set or reset state. The five flags are Sign, Zero, Auxiliary carry, Parity andCarry as shown below



Here X means undefined

Sign Flag: The sign flag is set to1 if the most significant bit of the result of an arithmetic or logic operations is 1. Else it is reset (0). **Zero Flag:** The Zero status flag is set to 1 if the result of an arithmetic or logic operation is Zero For non-Zero result it is reset to 0.G

Program Counter (PC):

It is a 16-bit special purpose register, which stores the address of the next instruction to befetched or executed. The execution of a program is initiated by loading the PC by the address ofthe first instruction of the program. Once the first instruction is executed, the PC is automaticallyincremented to point to the next instruction unless a jump to some specific address occurs. Thisprocess is repeated till the last instruction of the program.In case of JUMP or CALL instructions, current address is stored in the Program Counter.The processor then fetches the next instruction from the new address specified by the JUMP orCALL instruction. In conditional JUMP and conditional CALL instructions, if the condition is notsatisfied, the processor increments the Program Counter by three so that it points the instructionfollowed by the conditional JUMP or CALL instruction. Otherwise the processor fetches the nextinstruction from the new address specified by JUMP or CALL instruction.

Stack Pointer (SP):

It is a 16-bit special purpose register which always stores the address of top of the Stack.i.e. it lways points to top of the Stack. Stack is a part of the memory location used to store thedata temporarily. A stack works on Last in First out (LIFO) basis. As the Stack pointer alwayspoints to the top of the Stack, only top of the Stack of the memory can be accessed. When aWrite operation (PUSH) takes place, the contents of the stack pointer is decremented by two sothat the SP points to the new location

The remaining blocks of 8085 microprocessor block diagram

Instruction Register and Decoder: The instruction register and the decoder are also part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The Decoder decodes the instruction and develops the sequence of events to follow. The instruction register is a 8 – bit special register, but it is not a programmable and is not accessible to the user. The instruction decoder decodes the instruction at a binary level and sends the appropriate signals to the control unit.

Increment/ Decrement Address Latch: This is a 16 bit special register not accessible to the user. This register is used by the CPU to increment/ decrement the contents of the Stack Pointer (SP) and increment program counter (PC) during instruction execution. During first T - state of op code fetch machine cycle (T1) the microprocessor increments the PC register contents to point to the next location. This increment operation takes place on increment/ decrement register address latch. The 16 bit address that is sent out through AD0-AD7 and A8-A15 are latched into this register. The address bus AD0-AD7 continues to be available on the bus after T1 state from this latch.

Address Buffer: This is an 8-bit unidirectional buffer. It is used to drive external higher order address bus. It is also used to tri-state the higher order address (A8-A15) bus under certain conditions like reset, hold, and halt and also when address lines are not in use.

Address/Data Buffer: This is an 8-bit bi-directional buffer. It is used to drive multiplexed address/data bus. It means low order address bus (A7-A0) and data bus (D7-D0). It is also used to tri-state the multiplexed address/data bus under certain conditions like reset, hold, and halt and also when A/D bus lines are not in use.

Serial I/O control:

This control provides two lines SOD (Serial Out Data) and SID (serial In Data) for serial communication. These lines are used during serial data transmission over long distance where data is transmitted and received bit by bit. The Serial Output Data (SOD) pin is used to send data out serially and serial Input Data (SID) pin is used to receive data serially by the 8085 microprocessors.

8085 Interrupts:

The 8085 microprocessor has five interrupts. They are TRAP, RST 7.5, RST 6.5, RST5.5 and INTR. Among all these interrupts TRAP has the highest priority and INTR (Interrupt Request) has the lowest priority. The TRAP is also a non maskable interrupt. The numbers succeeding the RST (7.5, 6.5, and 5.5) are related to the call locations. The various interrupts, their locations in the order of highest to lowest priority are given in Table 3.1. Here RST means RESTART. Among these interrupts INTR is the only non-vector interrupt whereas the other interrupts are vectored interrupts

S.N o	Interrupts	Call locations
1	TRAP (Highest priority)	0024H
2	RST 7.5	(7.5 x 8)H = 003CH
3	RST 6.5	(6.5 x 8)H = 0034H
4	RST 5.5	(5.5 x 8)H = 002CH
5	INTR (least priority)	No location

Table 1 . Various Interrupts ,Call locations in order of highest to lowest priority

Timing and control Unit

This unit of the microprocessor issues necessary timing and control signals for the execution of instructions. It generates three types of signals namely status, control and timing signals required for the operation of memory and I/O devices. This unit with the help of these signals controls the entire operation of the microprocessor and the peripherals. The signals associated with this unit are two control signals. and three status signals IO/ , S1 and S0 to identify the nature of the operation and one special signal ALE which indicates the starting of the operation. These signals are explained below in detail. -Read (active low): This is a Read control signal. This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus. - Write (active low): This is a Write control signal. This signal indicates that the data on the data bus are to be written into a selected memory or I/O device.

S. No	IO/$\overline{M}$	S_1	S_0	Status
1	0	0	0	Memory Write
2	0	1	0	Memory Read
3	1	0	1	I/O Write
4	1	1	0	I/O Read
5	1/0	1	1	Opcode fetch
6	1	1	1	Interrupt Acknowledge
7	*	0	0	Halt
8	*	X	X	Hold
9	*	X	X	RESET

Table 2. Status signals of 8085

Timing Diagram :

The graphical representation of the time taken for the execution of each instruction by a microprocessor is known as timing diagram. The execution time is denoted by T-states. One Tstate is equal to the time period of the internal clock signal of the microprocessor For Ex: If the internal clock frequency of 8085 microprocessor is 3 MHZ, One T-state is equal to the time period of the internal clock signal of the microprocessor For Ex: If the internal clock frequency of 8085 microprocessor is 3 MHZ, One T-state is equal to

$$\frac{1}{f} = \frac{1}{3x10^6} = 0.333x10^{-6}\text{sec} = 333x10^{-9}\text{sec. (333 nano seconds nearly)}$$

Instruction cycle, Machine cycle, fetch and execute cycles

An instruction is a command given to the microprocessor to perform a specific operation on the given data. Sequence of instructions written for a processor to perform a particular task is called a program. Program & data are stored in the memory. The microprocessor fetches one instruction from the memory at a time & executes it. It executes all the instructions of the program one by one to produce the final result. The necessary steps that a microprocessor carries out to fetch an instruction &

necessary data from the memory & to execute it constitute an instruction cycle. In other words, an instruction cycle is defined as the time required completing the execution of an instruction. An instruction cycle consists of a fetch cycle and an execute cycle. The time required to fetch an opcode (fetch cycle) is a fixed slot of time while the time required to execute an instruction (execute cycle) is variable which depends on the type of instruction to be executed.

Instruction cycle(IC) = Fetch cycle(FC) + Execute cycle(EC)

This is shown diagrammatically in the **Fig.11**

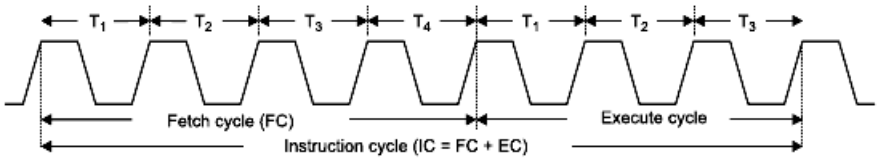


Figure 11 Instruction cycle

Machine cycle:

Machine cycle is defined as the time required for completing the operation of accessing either memory or I/O device. In the 8085, the machine cycle may consist of three to six T states. The T-state is defined as one sub-division of the operation performed in one clock period. These sub-divisions are internal states synchronized with the system clock.In every machine cycle the first operation is op-code fetch and the remaining will be read or write from memory or IO devices.**Fetch operation:**

The first byte of an instruction is its op-code. An instruction may be more than one bytelong. Theother bytes are data or operand address. The program counter (PC) keeps thememory address ofthe next instruction to be executed. In the beginning of a fetch cycle thecontent of the program counter, which is the address of the memory location where op-code isavailable, is sent to the memory. The memory places the op-code on the data bus so as totransfer it to the microprocessor. The entire operation of fetching an op-code takes three clockcycles.

Execute operation:

The op-code fetched from the memory goes to the instruction register (IR). From theinstruction register it goes to the decoder circuitry which decodes the instruction. After theinstruction is decoded, execution begins. If the operand is in general purpose registers executionis immediately performed

The time taken for decoding and execution is one clock cycle. If an instruction contains data oroperand and address which are still in the memory, the microprocessor has to perform someread operations to get the desired data. After receiving the data it performs execute operation. Aread cycle is similar to a fetch cycle. In case of a read cycle the quantity received from thememory are data or operand address instead of an op-code. In some instructions write operationis performed. In write cycle data are sent from the microprocessor to the memory or an outputdevice. Thus we see that in some cases an execute cycle may involve.

Instruction set of 8085

An Instruction is a command given to the microprocessor to perform a given task on specified data. Each instruction has two parts one is the task to be performed called theoperation code (op-code) and the second is the data to be operated on, known as operand. The operand or data can be specified in various ways.

CLASSIFICATION OF INSTRUCTIONS

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. The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. This instruction set includes all the 8080A instructions plus two additional instructions namely SIM and RIM.

S. No	Instruction	Description
1	JC (16 bit Addr)	Jump on carry (if CY=1)
2	JNC (16 bit Addr)	Jump on no carry (if CY=0)
3	JZ (16 bit Addr)	Jump on Zero (if Z=1)
4	JNZ (16 bit Addr)	Jump on no Zero (if Z=0)
5	JP (16 bit Addr)	Jump on plus (if D_7=0; S=0)
6	JM (16 bit Addr)	Jump on minus (if D_7=1; S=1)
7	JPE (16 bit Addr)	Jump on Even Parity (if P=1)
8	JPO (16 bit Addr)	Jump on Odd Parity (if P=0)

Table 3 various conditional jump instructions

Table 3 various conditional jump instructions

RESET (RST) Instruction

The 8085 processor provides eight RST instructions to transfer the program control to a specific location on page 00H. These instructions are 1-byte instructions. The various RST instructions and their call locations are given in the following Table 3.4

S. No	Mnemonics	Hex code	Call location In Hex
1	RST 0	C7	0000
2	RST1	CF	0008
3	RST2	D7	0010
4	RST3	DF	0018
5	RST4	E7	0020
6	RST5	EF	0028
7	RST6	F7	0030
8	RST7	FF	0038

Table 4 Various RST instructions and their call locations

DETAILED INSTRUCTION SET

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Move 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 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
Move immediate 8-bit		
MVI	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, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	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
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct		
LHLD	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 2040H

Store accumulator direct

STA	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 4350H

Store accumulator indirect

STAX	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

Store H and L registers direct

SHLD	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 2470H

Exchange H and L with D and E

XCHG	none	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. Example: XCHG

Copy H and L registers to the stack pointer

SPHL	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

Exchange H and L with top of stack

XTHL	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 register pair onto stack

PUSH Reg. pair — 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 PSW

Pop off stack to register pair

POP Reg. pair — The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

Example: POP H or POP PSW

Output data from accumulator to a port with 8-bit address

OUT 8-bit port address — The contents of the accumulator are copied into the I/O port specified by the operand.

Example: OUT F8H

Input data to accumulator from a port with 8-bit address

IN 8-bit port address — The contents of the input port designated in the operand are read and loaded into the accumulator.

Example: IN 8CH

ARITHMATIC INSTRUCTIONS

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 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: ADD B or ADD M

Add register to accumulator with carry

ADC 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

Subtract register or memory from accumulator

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 specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SBB B or SBB M

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 designated register pair are decremented by 1 and the result is stored in the same place.

Example: DCX H

Decimal adjust accumulator

DAA 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.

If the value of the low-order 4-bits in the accumulator is greater than 9 or 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 greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

BRANCHING INSTRUCTIONS

Opcode	Operand	Description

Jump unconditionally

JMP 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

Example: JMP 2034H or JMP XYZ

Jump conditionally

Operand: 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 2034H or JZ XYZ

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 2034H or CALL XYZ

Call conditionally

Operand: 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 2034H 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

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

RST 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:

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
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

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
if (A) = (reg/mem): zero flag is set
if (A) > (reg/mem): carry and zero flags are reset

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
if (A) = data: zero flag is set
if (A) > data: carry and zero flags are reset

Example: CPI 89H

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 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 86H

Exclusive OR register or memory with accumulator

XRA	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 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

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 86H

Logical OR register or memory with accumulator

ORA	R M	The contents of the accumulator are logically ORed 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 and AC are reset. Example: ORA B or ORA M

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 86H

Rotate accumulator left

RLC none

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.S, Z, P, AC are not affected.

Example: RLC

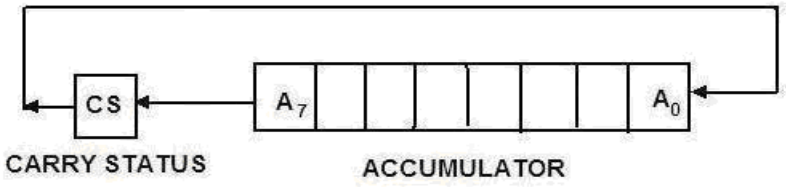


Figure 3.12 Schematic Diagram for RLC

Rotate accumulator right

RRC none

Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.S, Z, P, AC are not affected.
Example: RRC

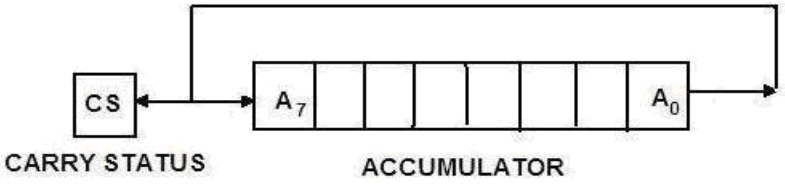


Figure 3.13 Schematic Diagram for RRC

Rotate accumulator left through carry

RAL none

Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.CY is modified according to bit D7. S, Z, P, AC are not affected.

Example: RAL

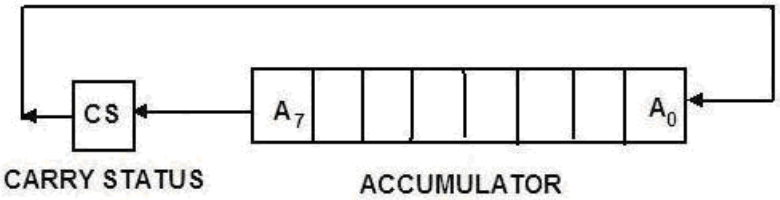


Figure 3.14 Schematic Diagram for RAL

Rotate accumulator right through carry

RAR none

Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected.

Example: RAR

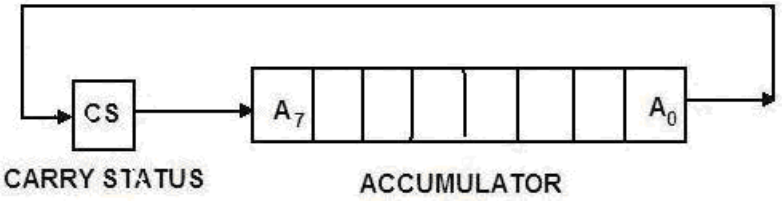


Figure 3.15 Schematic Diagram for RAR

CONTROL INSTRUCTIONS

Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
HLT	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
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
EI	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

Read interrupt mask

RIM 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

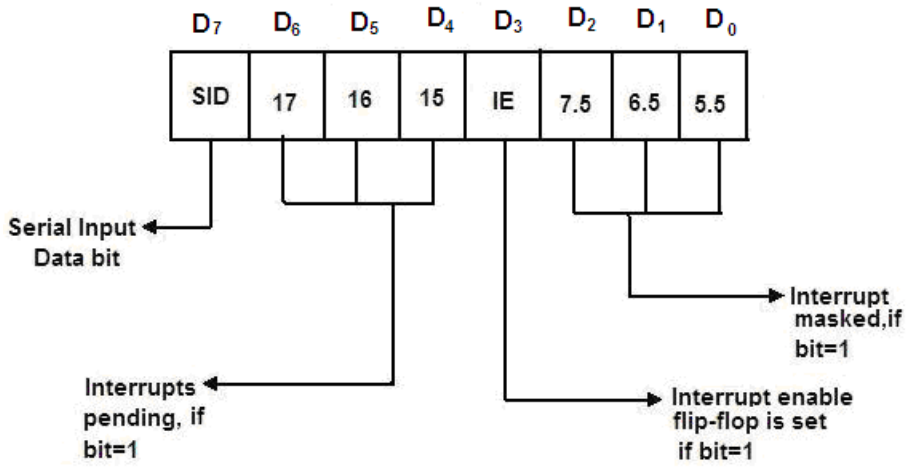


Figure 3.16 Read Interrupt Mask Register

Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM

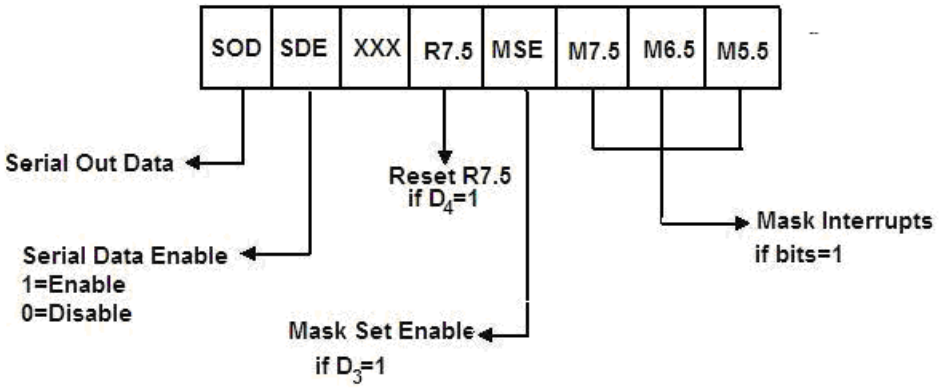


Figure 3.17 Set Interrupt Mask Register

ASSEMBLY LANGUAGE PROGRAMMING EXAMPLES:

Addition Programs

Example 1: Addition of two 8-bit numbers whose sum is 8-bits.

Explanation: This assembly language program adds two 8-bit numbers stored in two memory locations .The sum of the two numbers is 8-bits only. The necessary algorithm and flow charts are given below.

ALGORITHM:

Step1. : Initialize H-L pair with memory address XX00 (say: 9000).
Step2. : Clear accumulator.
Step3. : Add contents of memory location M to accumulator.
Step4. : Increment memory pointer (i.e. XX01).
Step5. : Add the contents of memory indicated by memory pointer to accumulator.
Step6. : Store the contents of accumulator in 9002.
Step7. : Halt

PROGRAM:

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	
8000	21,00,90		LXI	H, 9000	Initialise memory pointer to point the first data location 9000.
8003	3E		MVI	A, 00	Clear accumulator
8004	00				
8005	86		ADD	A, M	The first number is added to accumulator [A] ← [A] + M
8006	23		INX	H	Increment the memory pointer to next location of the Data.
8007	86		ADD	A, M	The 2nd number is added to contents of accumulator
8008	32		STA	9002	The contents of accumulator are stored in memory location 9002.
8009	02				
800A	90				
800B	76		HLT		Stop the execution

Ex: Input: Ex: (i) 9000 – 29 H **Ex :(ii)** 9000 –49 H
9001 – 16 H 9001 –32 H
Result: Ex: (i) 9002 – 3F H **Ex :(ii)** 9002 – 7B

Flow Chart

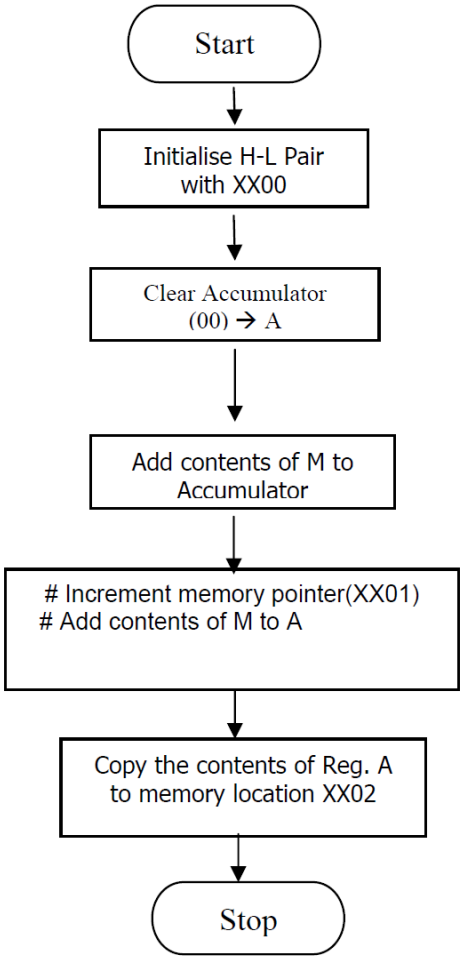


Fig 3.18

Example 2: Addition of two 8-bit numbers whose sum is 16 bits.
Explanation: The first 8-bit number is stored in one memory location (say 8500) and the second 8-bit number is stored in the next location (8501).Add these two numbers and check for carry. Store the LSB of the sum in one memory location (8502) and the MSB (carry) in the other location(8503).

ALGORITHM:

Step1. : Initialize H-L pair with memory address X (say: 8500).
Step2: Clear accumulator.
Step3. : Add contents of memory location M to accumulator.
Step4. : Increment memory pointer (i.e. 8501).
Step5. : Add the contents of memory indicated by memory pointer to accumulator.
Step6. : Check for Carry
Step7: Store the sum in 8502.
Step8 : Store the Carry in 8503 location
Step9 : Halt

PROGRAM

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	
8000	21,00 ,85		LXI	H, 8500 H	Initialise memory pointer to point the first data location 9000.
8003	3E		MVI	A,00	Clear accumulator
8004	00				
8005	86		ADD	A, M	The first number is added to accumulator [A]← [A]+M
8006	0E		MVI	C,00	Initial value of Carry is 0
8007	00				
8008	23		INX	H	Increment the memory pointer to next location of the Data.
8009	86		ADD	A, M	The 2nd number is added to contents of accumulator
800A	32		JNC	FWD	Is Carry exists ? No,go to the label FWD
800B	0E				
800C	80				
800D	0C		INR	C	Make carry =1
800E	32	FWD	STA	8502 H	The sum is stored in memory location 8502.
800F	02				
8010	85				
8011	79		MOV	A,C	
8012	32		STA	8503 H	Store the carry at 8503 location
8013	03				
8014	85				
8015	76		HLT		Stop the execution

Flow Chart

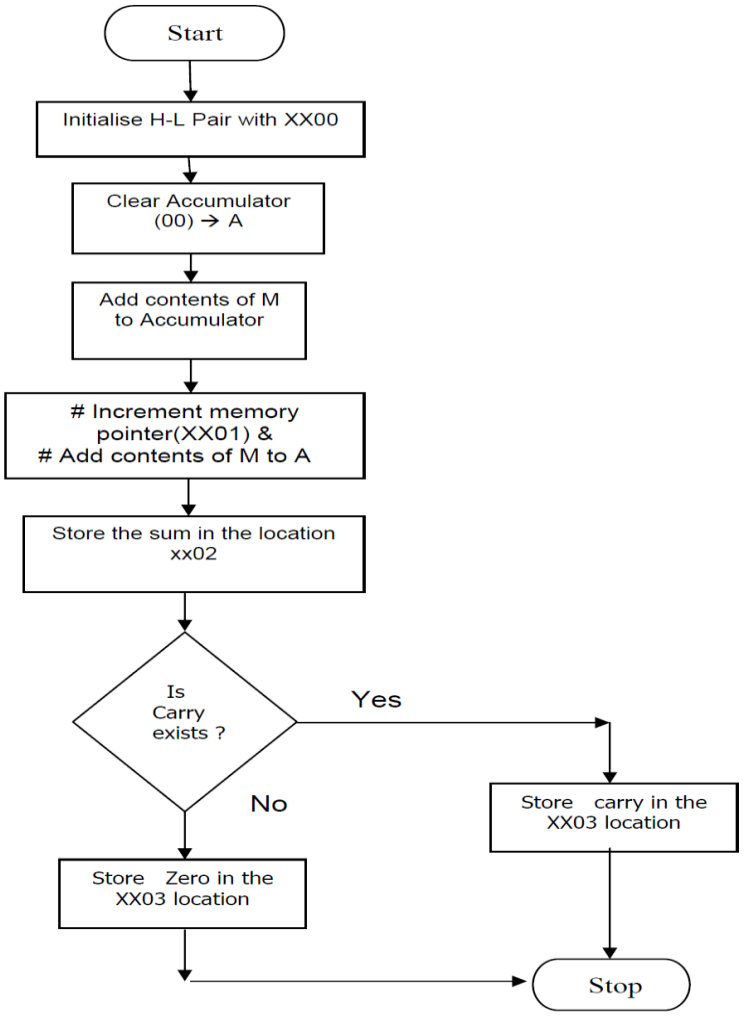


Fig 3.19

Example 3: Decimal addition of two 8-bit numbers whose sum is 16 bits.

Explanation: Decimal addition of two 8-bit numbers is same as that of two 8-bitnumbersprogram except that the use of DAA instruction. The first 8-bit number is stored in one memorylocation (say 8500) and the second 8-bit number is stored in the next location(8501).Add thesetwo numbers and use the DAA instruction to get the result in decimal. Also check for carry. Storethe LSB of the sum in one memory location(8502) and the MSB (carry) in the otherlocation(8503).

ALGORITHM:

Step1. : Initialize H-L pair with memory address XXXX (say: 8500).
Step2. : Clear Carry register C.
Step3. : Move contents of memory location M to accumulator.
Step4. : Increment memory pointer (i.e. 8501).
Step5. : Add the contents of memory indicated by memory pointer to accumulator.
Step6. : Apply the instruction DAA(Decimal adjust after addition)
Step7: Check for Carry
Step8: Store the sum in XX02.
Step9: Store the Carry in XX03 location
Step10: Halt

Flow Chart

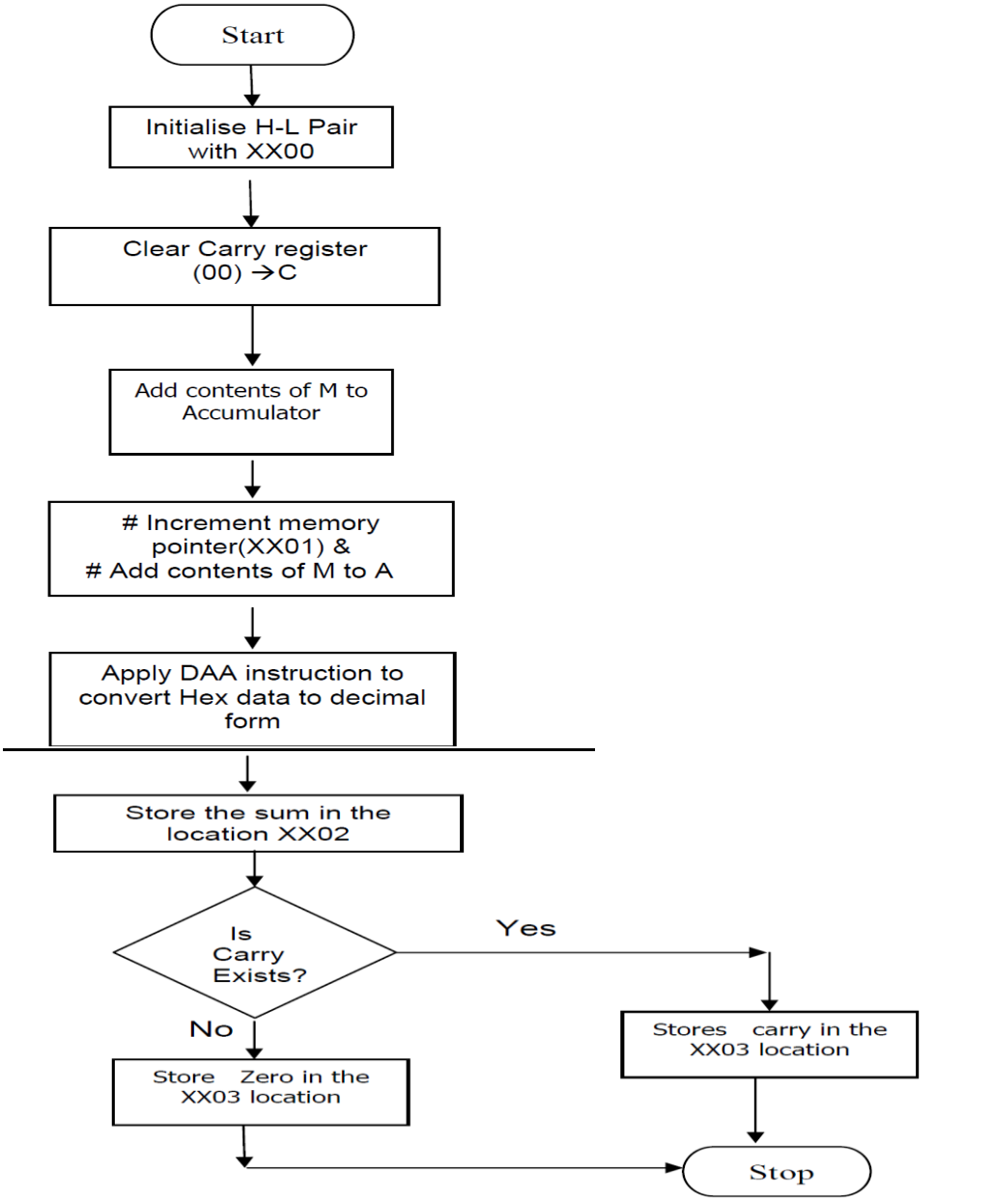


Fig .20

PROGRAM

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	
8000	21, 00,85		LXI	H, 8500 H	Initialise memory pointer to point the first data location 9000.
8003	0E		MVI	C, 00	Clear accumulator
8004	00				
8005	7E		MOV	A, M	The first number is added to accumulator [A]← [A]+M
8006	23		INX	H	Increment the memory pointer to next location of the Data.
8007	86		ADD	A, M	The 2^{nd} number is added to contents of accumulator
8008	27		DAA		
8009	D2		JNC	FWD	Is Carry exists? No, go to the label FWD
	0D				
	80				
800C	0C		INR	C	Make carry =1
800D	32	FWD	STA	8502 H	The contents of accumulator are stored in memory location 8502.
800E	02				
800F	85				
8010	79		MOV	A, C	Carry is moved to accumulator
8011	32		STA	8503 H	A Carry is stored in the location 8503
8012	03				
8013	85				
8014	76		HLT		Stop the execution

Multiplication Programs

Example 7: Multiplication of two 8-bit numbers. Product is 16-bits.

Explanation: The multiplication of two binary numbers is done by successive addition. Whenmultiplicand is multiplied by 1 the product is equal to the multiplicand, but when it is multiplied byzero, the product is zero. So, each bit of the multiplier is taken one by one and checked whetherit is 1 or 0 .If the bit of the multiplier is 1 the multiplicand is added to the product and the productis shifted to left by one bit. If the bit of the multiplier is 0 , the product is simply shifted left by onebit. This process is done for all the 8-bits of the multiplier.

ALGORITHM:

Step 1 : Initialise H-L pair with the address of multiplicand.(say 8500)
Step 2 : Exchange the H-L pair by D-E pair. so that multiplicand is in D-E pair.
Step 3 : Load the multiplier in Accumulator.
Step 4 : Shift the multiplier left by one bit.
Step 5 : If there is carry add multiplicand to product.
Step 6 : Decrement the count.
Step 7 : If count ‧ 0; Go to step 4
Step 8 : Store the product i.e. result in memory location.
Step 9 : Stop the execution

Flow Chart

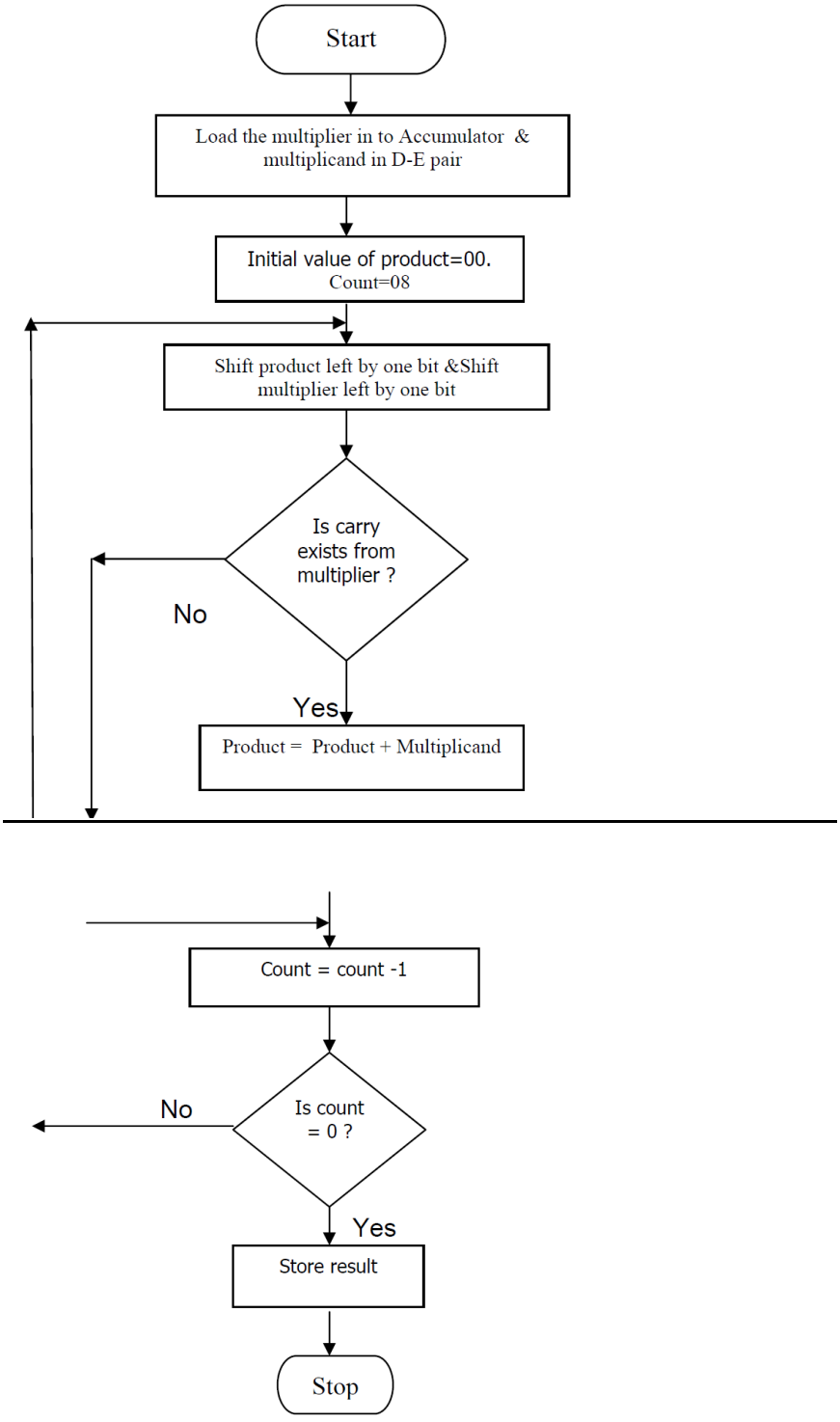

Start
Load the multiplier in to Accumulator & multiplicand in D-E pair
Initial value of product=00.
Count=08
Shift product left by one bit &Shift multiplier left by one bit
Is carry exists from multiplier ?
No
Yes
Product = Product + Multiplicand
Count = count -1
Is count = 0 ?
No
Yes
Store result
Stop

Figure 25

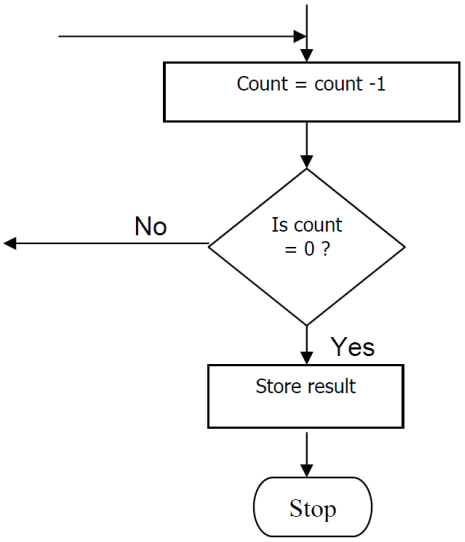


Figure 25

PROGRAM:

ADDRESS	HEX - CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	2A,00,85		LHLD	H, 8500	Load the multiplicand in to H-L pair
8003	EB		XCHG		Exchange the multiplicand in to D-E pair
8004	3A		LDA	8502	Multiplier in Accumulator
8005	02				
8006	85				
8007	21		LXI	H.0000	Initial value in H-L pair is 00
8008	00				
8009	00				
800A	0E		MVI	C,08	Count =08
800B	08				
800C	29	LOOP	DAD	H	Shift the partial product left by one bit.
800D	17		RAL		Rotate multiplier left by one bit
800E	D2		JNC	FWD	Is Multiplier bit =1? No go to label FWD
800F	12				
8010	80				
8011	19		DAD	D	Product =Product +Multiplicand

8012	0D	FWD	DCR	C	COUNT=COUNT-1
8013	C2		JNZ	LOOP	
8014	0C				
8015	80				
8016	22		SHLD	8503	Store the result in the locations 8503 & 8504
8017	03				
8018	85				
8019	76		HLT		Stop the execution

Division Programs

Example 7: Division of a 16- bit number by a 8-bit number.

Explanation: The division of a 16/8-bit number by a 8-bit number follows the successive subtraction method. The divisor is subtracted from the MSBs of the dividend .If a borrow occurs, the bit of the quotient is set to 1 else 0.For correct subtraction process the dividend is shifted left by one bit before each subtraction. The dividend and quotient are in a pair of register H-L.The vacancy arised due to shifting is occupied by the quotient .In the present example the dividend is a 16-bit number and the divisor is a 8-bit number. The dividend is in locations 8500 &8501.Similarly the divisor is in the location 8502.The quotient is stored at 8503 and the remainder is stored at 8504 locations.

ALGORTHM:

STEP1. : Initialise H-L pair with address of dividend.
STEP2. : Get the divisor from 8502 to register A & then to Reg.B
STEP3. : Make count C=08
STEP4. : Shift dividend and divisor left by one bit
STEP 5: Subtract divisor from dividend.
STEP6. : If carry = 1 :goto step 8 else step7.
STEP7. : Increment quotient register.
STEP8. : Decrement count in C
STEP9. : If count not equal to zero go to step 4
STEP10: Store the quotient in 8503
STEP11: Store the remainder in 8504
STEP12: Stop execution.

Stack and Subroutines

Stack is a set of memory locations in the Read/Write memory which is used for temporary storage of binary information during the execution of a program. It is implemented in the Last-infirst- out (LIFO) manner. i.e., the data written first can be accessed last, One can put the data on the top of the stack by a specialoperationknown as Push. Data can be read or taken out from the top of the stack by another special instruction known as POP.

3.14.1 Application of Stack: Stack provides a powerful data structure which has applications in many situations. The main advantage of the stack is that,

We can store data (PUSH) in it with outdestroying previously stored data. This is not true in the case of other registers and memory locations.

stack operations are also very fast

The stack may also be used for storing local variables of subroutine and for the transfer of parameter addresses to a subroutine. This facilitates the implementation of re-entrant subroutines which is a very important software property.

The disadvantage is, as the stack has no fixed address, it is difficult to debug and document a program that uses stack.

3.14.2 Stack operation: Operations on stack are performed using the two instructions namely PUSH and POP. The contents of the stack are moved to certain memory locations after PUSH instruction. Similarly, the contents of the memory are transferred back to registers by POP instruction.

For example let us consider a Stack whose stack top is 4506 H. This is stored in the 16-bit Stack pointer register as shown in **Fig.3.29**

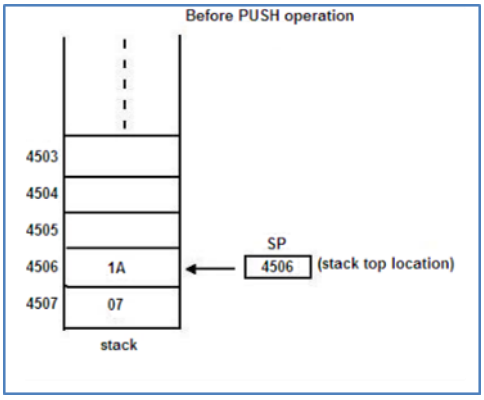


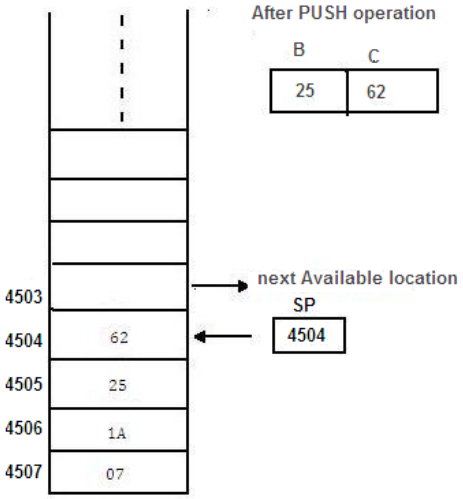
Figure 3.29: The PUSH operation of the Stack

Let us consider two registers (register pair) B & C whose contents are 25 & 62.

Reg. B Reg. C

25	62

After PUSH operation the status of the Stack is as shown in **Fig 3.30**



Let us now consider POP operation: The **Figs 3.31 & 3.32** explains before and after the POP operation in detail

Figure 3.30 After PUSH operation the status of the stack

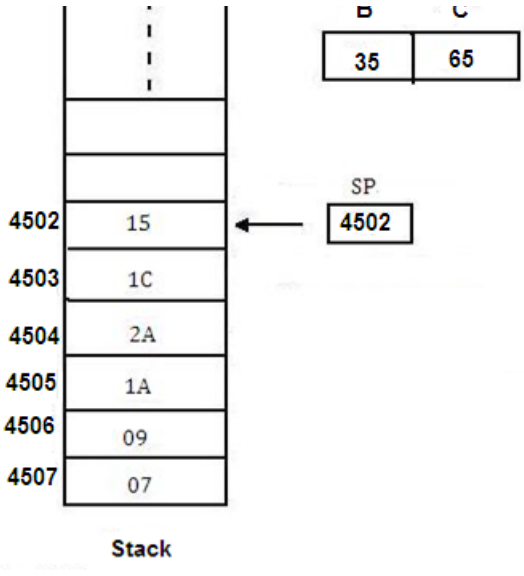


Figure 3.32: After POP operation the status of the stack

Before the operation the data 15 and 1C are in the locations 4502 & 4503 and after the popoperation the data is copied to B-C pair and now the SP register points to 4504 location.This isshown in Fig.3.32.

Programming Example FOR PUSH & POP

Write a program to initialize the stack pointer (SP) and store the contents of the register pair H-L on stack by using PUSH instruction. Use the contents of the register pair for delay counter and at the end of the delay retrieve the contents of H-L using POP.

Memory Location	Label	Mnemonics	Operand	Comments
8000 8003 8006 8007 . . . . 8.00A		LXI LXI PUSH DELAY . . . POP	SP, 4506 H H,2565 H H CALL . . . H	Initialize Stack pointer Push the contents.

Subroutine:

It is a set of instructions written separately from the main program to execute a function that occurs repeatedly in the main program. For example, let us assume that a delay is needed three times in a program. Writing delay programs for three times in a main program is nothing but repetition. So, we can write a subroutine program called 'delay' and can be called any number of times we need Similarly, in 8085 microprocessor we do not find the instructions for multiplication and division. For this purpose we write separate programs.

So, in any main program if these operations are needed more than once, the entire program will become lengthy and complex. So, we write subroutine programs MUL & DIV separately from main program and use the instructionCALL MUL (or) CALL DIV in the main program. This can be done any number of times. At the end of every subroutine program there must be an instruction called 'RET'. This will take the control back to main program.

The 8085 microprocessor has two instructions to implement the subroutines. They are CALL And RET. The CALL instruction is used in the main program to call a subroutine and RET Instruction is used at the end of the subroutine to return to the main program. When a subroutineIs called, the contents of the program counter, which is the address of the instruction followingthe CALL instruction is stored on the stack and the program execution is transferred to thesubroutine address. When the RET instruction is executed at the end of the subroutine, thememory address stored on the stack is retrieved and the sequence of execution is resumed inthe main program.

Diagrammatic representation

Let us assume that the execution of the main program started at 8000 H. It continues until aCALL subroutine instruction at 8020 H is encountered. Then the program execution transfers to8070 H. At the end of the subroutine 807B H. The RET instruction is present. After executing thisRET, it comes back to main program at 8021 H as shown in the following Fig. 3.34.

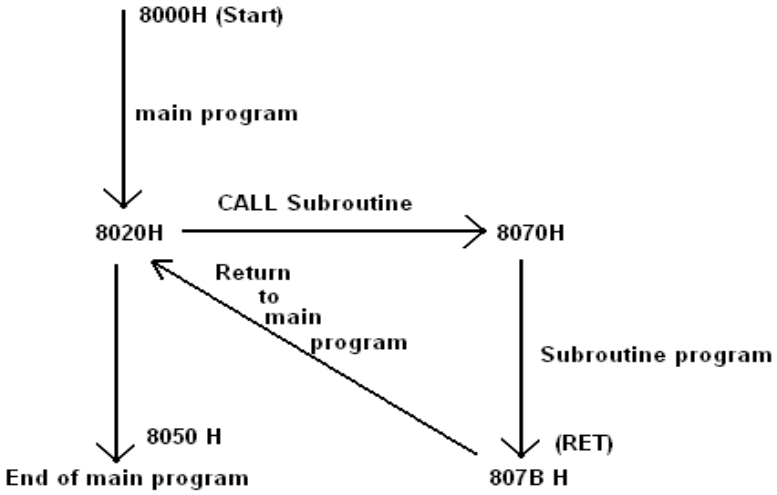


Fig 3.33: Diagrammatic representation of subroutine programexecution

The same is explained using the assembly language program example.

Memory Address	Mnemonics	Operand	Comments
8000 | | |	LXI	SP, 8400 H	Initialize the Stack pointer at 8400 H
8020 8021 8022 8023 | | | 802F	CALL Next instruction | | | HLT	8070 H	Call a subroutine program stored at the location 8070 H. (It is a three by Instruction) The address of the next instruction following CALL instruction. End of the main program .