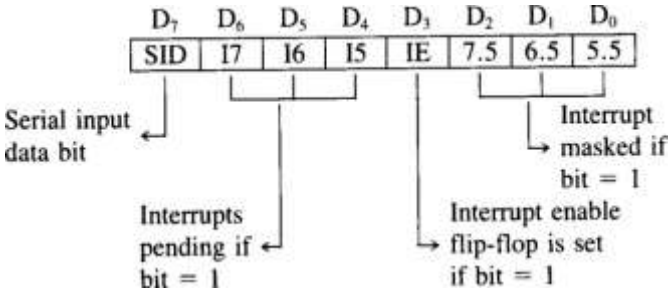
2.4 CONTROL INSTRUCTIONS

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



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

2.3 ARITHMETIC INSTRUCTIONS

Includes the instructions which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.

Ex: i) ADD B ii) SUB C iii) INR D iv) INX H

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
Add immediate to accumulator		
ADI	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 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H
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

Subtract immediate from accumulator

SUI	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 45H

Subtract immediate from accumulator with borrow

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

Increment register or memory by 1

INR	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

Increment register pair by 1

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

Decrement register or memory by 1

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

Decrement register pair by 1

DCX	R	The contents of the 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

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 accumulaotr

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

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

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 D_0. CY is modified according to bit D_7. S, Z, P, AC are not affected.
Example: 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 D_0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D_7. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RAR

Complement accumulator

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

Complement carry

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

Set Carry

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

2.2 DATA TRANSFER INSTRUCTIONS:

Includes the instructions that moves (copies) data between registers or between memory locations and registers. In all data transfer operations the content of source register is not altered. Hence the data transfer is copying operation.

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not 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 A

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 A

2.1 INSTRUCTION FORMAT OF 8085

The 8085 have 74 basic instructions and 246 total instructions. The instruction set of 8085 is defined by the manufacturer Intel Corporation. Each instruction of 8085 has 1 byte opcode. With 8 bit binary code, we can generate 256 different binary codes. In this, 246 codes have been used for opcodes.

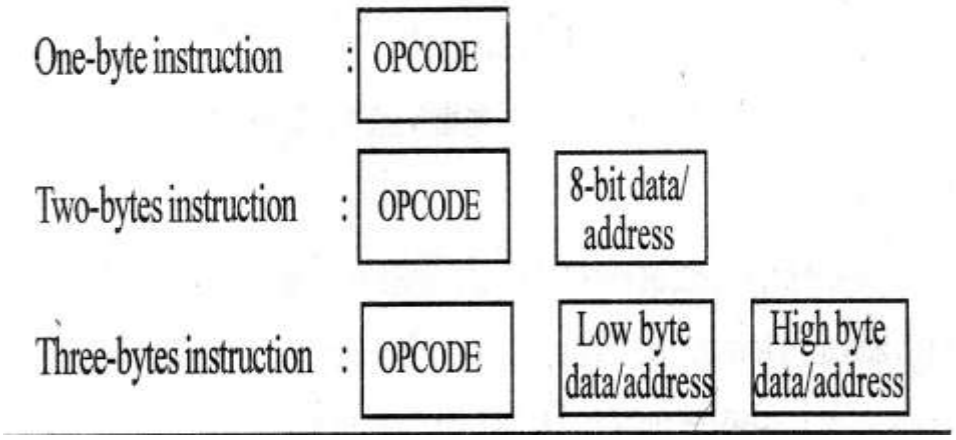


Figure 2.1.1 Instruction Format of 8085

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-131]

The size of 8085 instructions can be 1 byte, 2 bytes or 3 bytes.

- The 1-byte instruction has an opcode alone.
- The 2 bytes instruction has an opcode followed by an eight-bit address or data.
- The 3 bytes instruction has an opcode followed by 16 bit address or data. While storing the 3 bytes instruction in memory, the sequence of storage is, opcode first followed by low byte of address or data and then high byte of address or data.

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

Immediate Addressing

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.

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.

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.

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.

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.

2.5 LOOPING,COUNTING AND INDEXING

LOOPING:

The programming technique used to instruct the microprocessor to repeat tasks is called looping.This task is accomplished by using jump instructions.

Classification Of Loops:

1.continuous loop

2.Unconditional loop

Continuous Loop:

Repeats a task continuously.A continuous loop is set up by using the unconditional jump instruction.A program with a continuous loop does not stop repeating the tasks until the system is reset.

Conditional Loop:

A conditional loop is set up by a conditional jump instructions. These instructions check flags(Z,CY,P,S) and repeat the tasks if the conditions are satisfied. These loops include counting and indexing.

Conditional Loop And Counter:

- A counter is a typical application of the conditional loop.
- A microprocessor needs a counter,flag to accomplish the looping task.
- Counter is set up by loading an appropriate count in a register.
- Counting is performed by either increment or decrement the counter.
- Loop is set up by a conditional jump instruction.
- End of counting is indicated by a flag.

Conditional Loop,Counter And Indexing:

Another type of loop which includes counter and indexing .

Indexing:

Pointing of referencing objects with sequential numbers. Data bytes are stored in memory locations and those data bytes are referred to by their memory locations.

Example:

Steps to add ten bytes of data stored in memory locations starting ata given location and display the sum.

The microprocessor needs

- A counter to count 10 data bytes.
- An index or a memory pointer to locate where data bytes are stored.
- To transfer data from a memory location to the microprocessor(ALU)
- To perform addition
- Registers for temporary storage of partial answers
- A flag to indicate the completion of the stack
- To store or output the result.

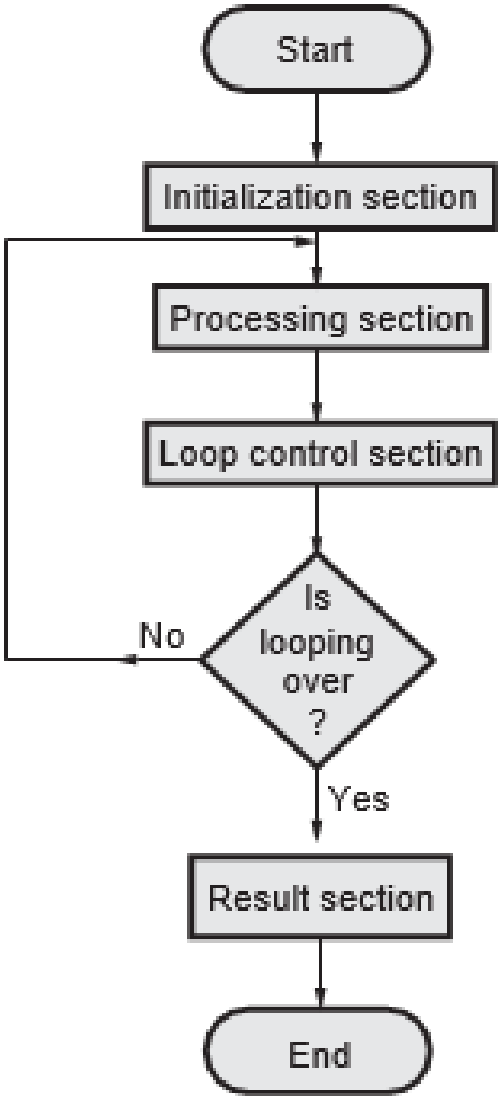


Figure 2.5.1 Looping flow chart

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-190]

1. The initialization section establishes the starting values of loop counters for counting how many times loop is executed, Address registers for indexing which give pointers to memory locations and other variables.

The actual data manipulation occurs in the processing section. This is the section which does the work.

The loop control section updates counters, indices (pointers) for the next iteration.

The result section analyzes and stores the results.

The processor executes initialization section and result section only once, while it may execute processing section and loop control section many times. Thus, the execution time of the loop will be mainly dependent on the execution time of the processing section and loop control section. The flowchart 1 shows typical program loop. The processing section in this flowchart is always executed at least once. If you position of the processing and loop control section then it is possible that the processing section may not be executed at all, if necessary.

2.7 STACK OPERATIONS

- The stack is a group of memory location in the R/W memory (RAM) that is used for temporary storage of data during the execution of a program.
- Address of the stack is stored into the stack pointer register. The 8085 provide two instructions PUSH & POP for storing information on the stack and reading it back.
- Data in the register pairs stored on the stack by using the instruction PUSH.
- Data is read from the stack by using the instruction POP.
- PUSH & POP both instruction works with register pairs only.
- he storage and retrieval of the content of registers on the stack fallows the LIFO(Last- In-First-Out) sequence.

Operation of the stack by PUSH and POP Instruction

```
2000  LXI SP, 2099H     ; this instruction define stack
2003  LXI H, 42F2H      ; this instruction store 42F2 in to the HL pair
2006  PUSH H            ; store HL pair on to the stack
2010  POP H             ; store data from top of the stack to HL pair
```

For PUSH H

The stack pointer is decremented by one to 2098H, and the contents of the h register are copied to memory location 2098H.The stack pointer register is again decremented by one to 2097H,and the contents of the L register are copied to memory location 2097H.The contents of the register pair HL are not destroyed.

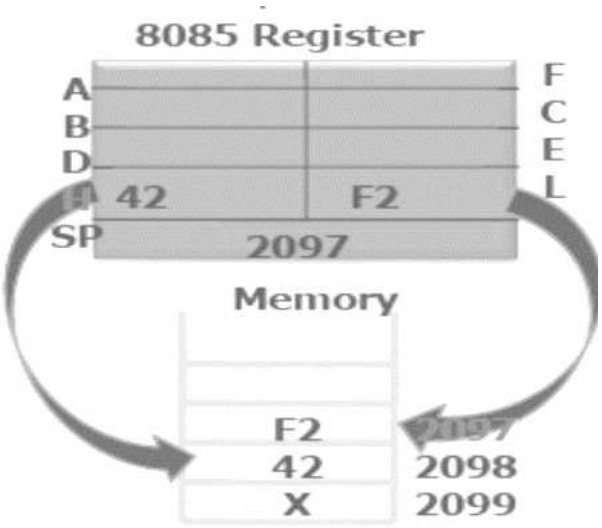


Figure 2.7.1 PUSH H operation

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-238]

For POP H

The contents of the top of the stack location shown by the stack pointer are copied in the L register and the stack pointer register is incremented by one to 2098 H. The contents of the top of the stack (now it is 2098H) are copied in the H register, and the stack pointer is incremented by one. The contents of memory location 2097H and 2098 are not destroyed until some other data bytes are stored in these location.

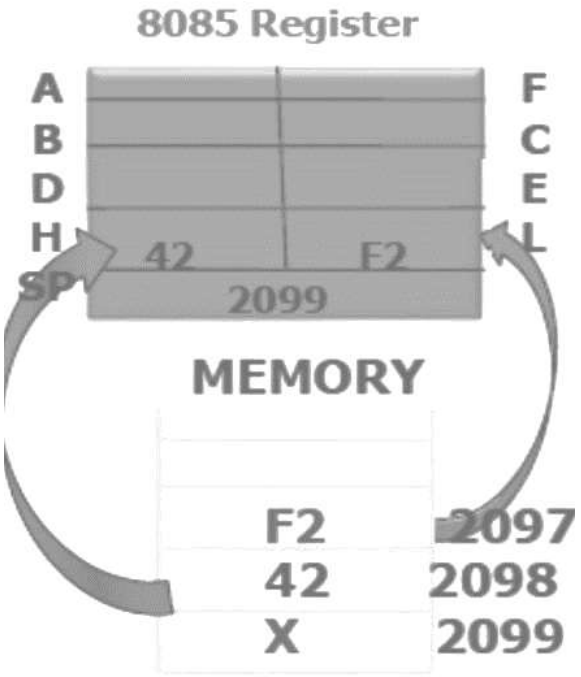


Figure 2.7.1 POP H operation

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-238]

2.6 SUBROUTINE INSTRUCTIONS

A subroutine is a group of instructions that will be used repeatedly in different locations of the program. Rather than repeat the same instructions several times, they can be grouped into a one program which is called subroutine.

When main program calls a subroutine the program execution is transferred to the subroutine.After the completion of the subroutine, the program execution returns to the main program.The microprocessor uses the stack to store the return address of the subroutine.

The 8085 has two instructions for dealing with subroutines.

- The CALL instruction is used to CALL the subroutine.
- The RET instruction is used to return to the main program at the end of the subroutine.

Subroutine process is shown in figure below.

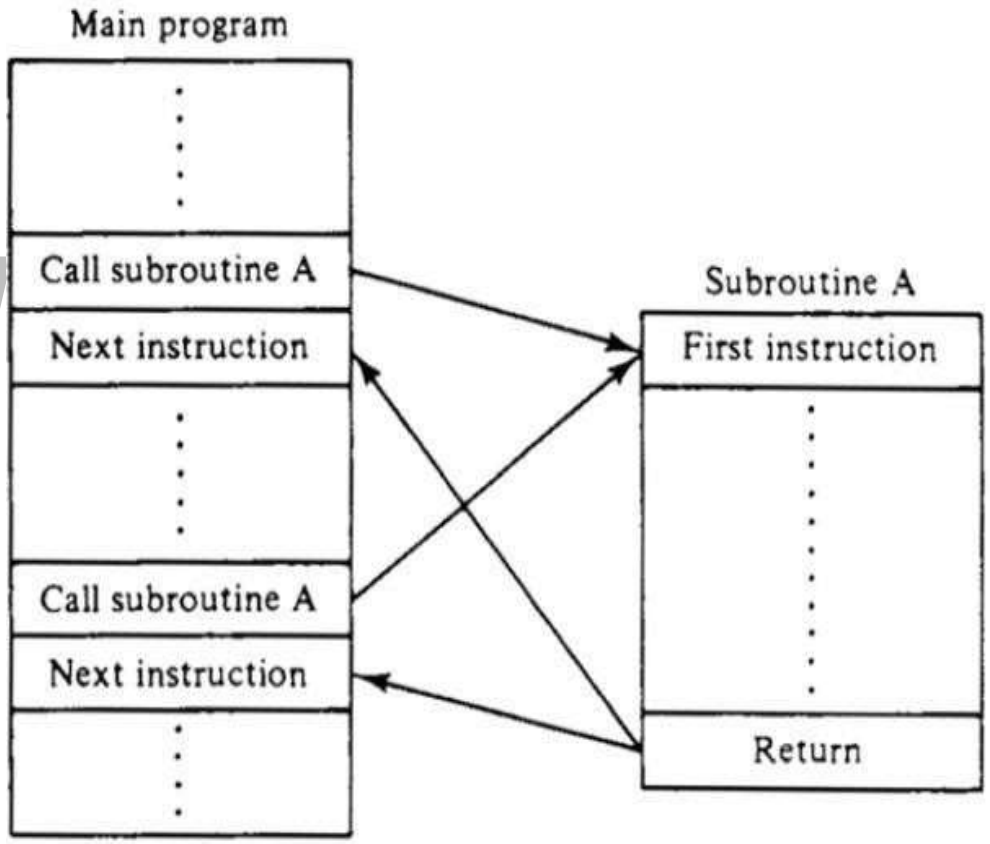


Figure 2.6.1 Subroutine process

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-249]

The CALL Instruction

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

We can also call the subroutine by using conditional CALL instruction. For Example,

CC 16-bit address Call on if CY = 1

CNC16-bit address Call on no Carry CY = 0

CP16-bit address Call on positive S = 0

CM16-bit address Call on minus S = 1

CZ 16-bit address Call on zero Z = 1

CNZ16-bit address Call on no zero Z = 0

CPE16-bit address Call on parity even P = 1

CPO16-bit address Call on parity odd P = 0

RET Instruction

RET 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

We can also return from the subroutine by using conditional RET instruction. For Example,

RC 16-bit address Return if CY = 1

RNC16-bit address Return if CY = 0

RP16-bit address Return if S = 0

RM16-bit address Return if S = 1

RZ 16-bit address Return if Z = 1

RNZ16-bit address Return if Z = 0

RPE16-bit address Return if P = 1

RPO16-bit address Return if P = 0