5 Looping, Counting and Indexing

Let us study these terms first.

Looping : Program loop is the basic structure which forces the CPU to repeat the sequence of instructions for a particular number of times e.g. to add 5 numbers stored in consecutive memory locations, we have to perform addition five times.

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

Indexing : This method allows the programmer to point or refer the data stored in the sequential memory locations one by one.

One can divide the LOOP in four sections. They are :

(1) **Initialization section :** This section establishes the starting value of counter, address registers for indexing which give pointers to memory locations and other variables.

(2) **Processing section :** In this section the actual data manipulation occurs. This is the section which does the work.

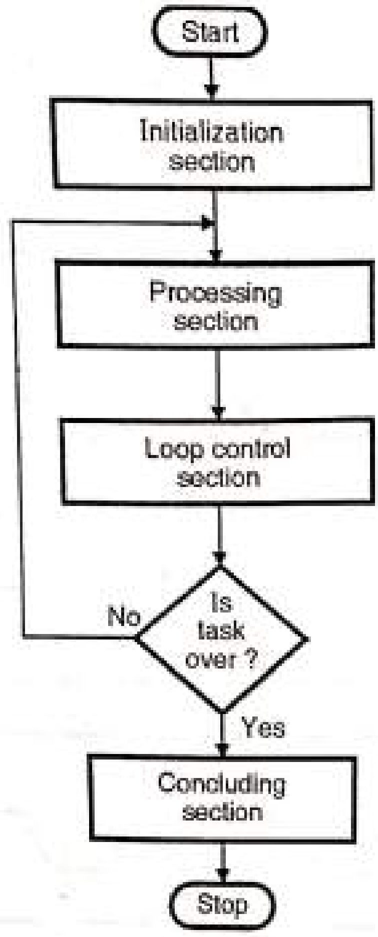
(3) **Loop control section :** This section updates the counters and pointers for next iteration.

(4) **Concluding section :** This section analyzes and stores the result.

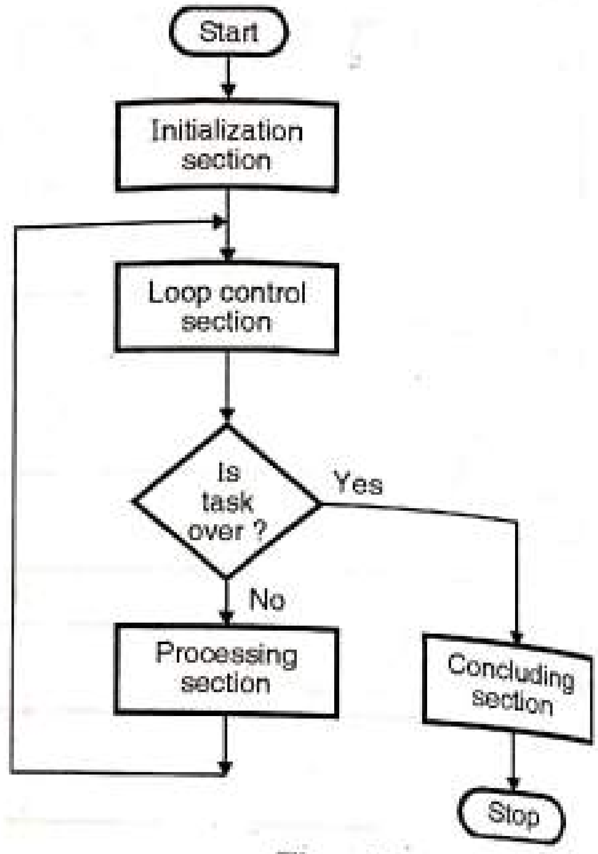
Note : (1) The microprocessor executes section (1) and (4) once.

(2) Microprocessor executes section (2) and (3) many times. The execution time mainly depends on sections (2) and (3).

Flowcharts 1 and 2 shows the ways of arranging these four sections.



Flowchart 1



Flowchart 2

1. Processing section is always executed once.	1. Processing section may not be executed at all.
2. Less efficient.	2. More efficient.

- The loop structure can process entire blocks of data. To achieve this, the program must increment address register (pointer), after each iteration so that address register, points to next element in data block.
- The next iteration will then perform the same operations on the data in the next memory location. Thus, microprocessor is capable enough to handle the blocks of any length with same set of instructions.

Ex. 8.5.1 : Program to calculate the sum of series of 8 bit numbers assuming sum to be 8 bit.

➤➤ Program statement :

Calculate the sum of series of numbers. The length of series is in memory location D000 H. The series begins from D001 H. Assuming the sum to be an 8 bit number so that carry can be ignored. Store the result in memory locations E000 H.

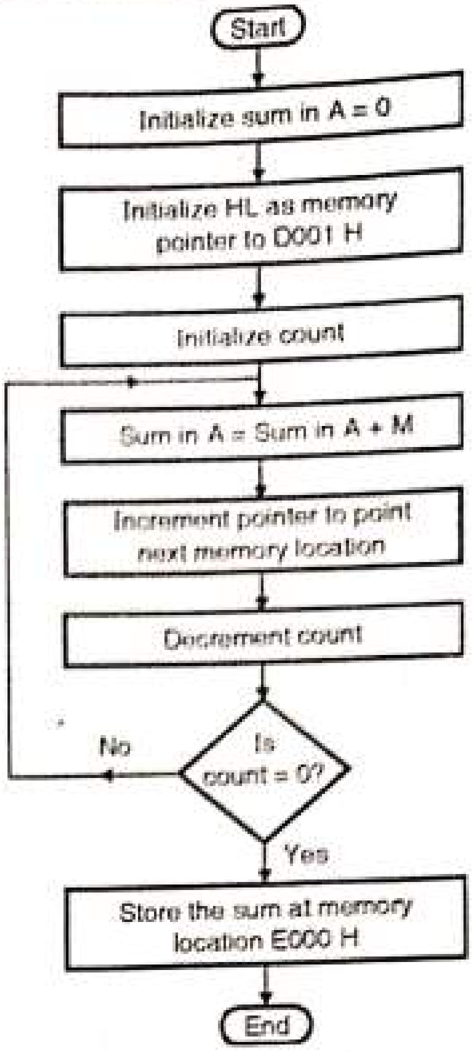
➤➤ Explanation :

- We are given a series of numbers. The length of series is stored at memory location D000 H. We initialise register C as counter with the length of the series.
- We will initialise the accumulator with 00 H, so that the sum can be stored in the accumulator.
- The series begins at D001 H. So, we will initialise HL the register pair as memory pointer to point series.

Using add instruction, add the contents, of the accumulator with the contents of memory location pointed by HL register pair. The result of this addition will be stored in the A register.

Then we will increment HL to point to next memory location of the series. Decrement the count in register C. Continue this process till the count is zero i.e. all the numbers in the series are added.

Store the result at memory location E000 H.



Flowchart 3

Example : Let D000 = 05 H i.e. series is of 5 numbers.

D001 = 08 H

D002 = 12 H

D003 = 74 H

D004 = 34 H

D005 = 04 H

Result = 08 H + 12 H + 74 H + 34 H + 04 H = C6 H

E000 H = C6 H.

Flowchart : Refer Flowchart 3.

Program :

Label	Instruction	Comment	Operation
	LDA D000H	A = contents of location D000 H	A = 05 H
	MOV C, A	Initialize counter	C = 05 H
	XRA A	sum = 0	A = 00 H
	LXI H, D001H	Initialize HL as memory pointer for the series	H = D0 H L = 01 H
L1 :	ADD M	SUM in A= SUM in A + M	A = A + M
	INX H	Increment pointer	HL = HL + 1
	DCR C	Decrement counter	C = C – 1
	JNZ L1	If counter ≠ 0 repeat	
	STA E000H	store the result at memory location E000 H	E000H : C6H Result
	HLT	Terminate program execution	Stop

Ex. 8.5.3 : Program to sort the numbers in ascending order.

(PTU, Dec. 2005)

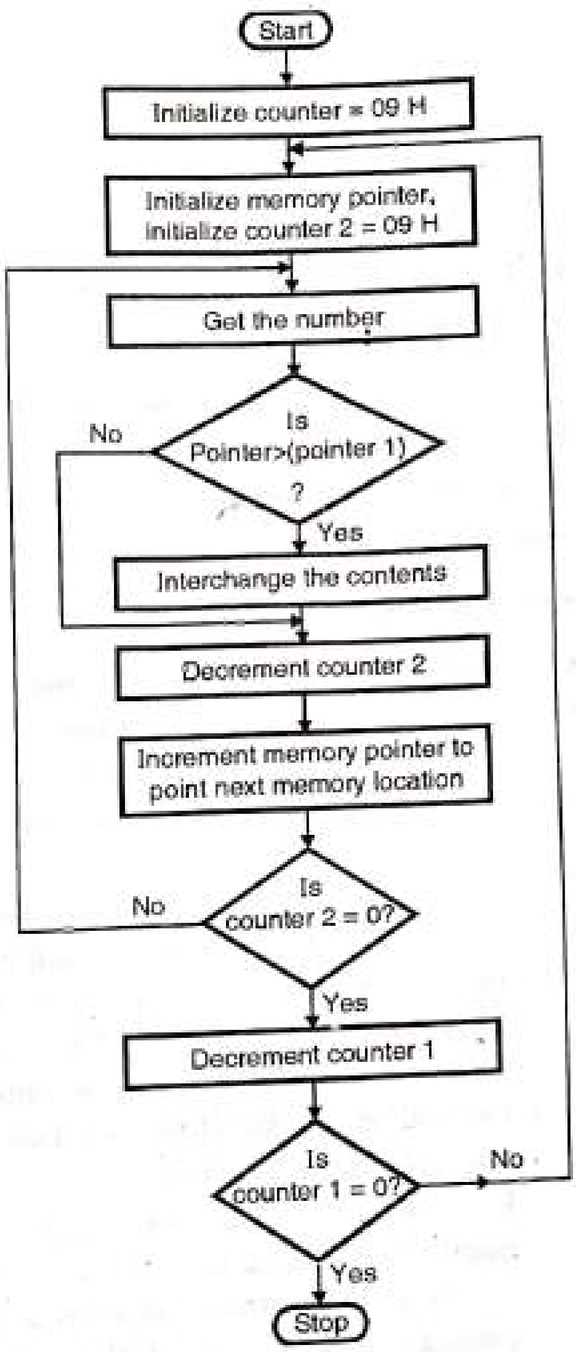
Program statement :

Write a program in assembly language of 8085 to sort the given N numbers from a block in ascending order. Assume that the memory block begins at D000 H.

Explanation :

- Consider that a block of N words is present. Now we have to arrange these N words in ascending order, Let N = 4 for example. We will use HL as pointer to point the block of N words.
- Initially in the first iteration we compare first number with the second number. If first number < second number, don't interchange the contents, otherwise if first number > second number swap the contents.
- In the next iteration we go on comparing the first number with third number. If first number < third number, don't interchange the contents. If first number > third number then swapping will be done.
- Since the first two numbers are in ascending order the third number will go to first place, first number in second place and second number will come in third place in the second iteration only if first number > third number.
- In the next iteration first number is compared with fourth number. So comparisons are done till all N numbers are arranged in ascending order. This method requires approximately n comparisons.

Flowchart : Refer Flowchart 5.



Flowchart 5

Program :

Label	Instruction	Comment	Operation
	MVI B, 09	Initialize counter 1 06	B = 09 H
START :	LXI H, D000H	Initialize memory pointer 21	H = D0 H, L = 00 H
	MVI C, 09H	Initialize counter 2 0E	C = 09 H
BACK :	MOV A, M	Get the number in accumulator 7E	A ← M
	INX H	Increment memory pointer 23	HL = HL + 1

Label	Instruction	Comment	Operation
	CMP M	Compare number with next number BE	Compare
	JC SKIP	If less, don't interchange DA	If A < M don't interchange
	JZ SKIP	If equal, don't interchange CA	If A = M don't interchange
	MOV D, M	Otherwise swap the contents 56	D ← M
	MOV M, A	77	M ← A
	DCX H	Interchange numbers 2B	HL = HL – 1
	MOV M, D	72	M ← D
	INX H	Increment pointer to next memory location 23	HL = HL + 1
SKIP :	DCR C	Decrement counter 2 0D	C = C – 1
	JNZ BACK	If not zero, repeat C2	
	DCR B	Decrement counter 1 05	B = B – 1
	JNZ START	If not zero, repeat C2	
	HLT	Terminate program execution 76	Stop

Ex. 8.5.5 : Multiply two 8 bit numbers using successive addition method. (PTU - Dec. 2005)

>> **Program statement :**

Multiply two 8-bit numbers stored in memory locations D000 H and D001 H. Store the result in memory locations E000 H and E001 H.

>> **Explanation :**

- Consider that a byte is present at the memory location D000 H and second byte is present at memory location D001 H.
- We have to multiply the bytes present at the above two memory locations.
- We will multiply the numbers using successive addition method.
- In successive addition method, one number is accepted and other number is taken as a counter. The first number is added with itself, till the counter decrements to zero.
- Result is stored at memory locations E000 H and E001 H.

For example : D000 H = 12 H, D001 H = 10 H

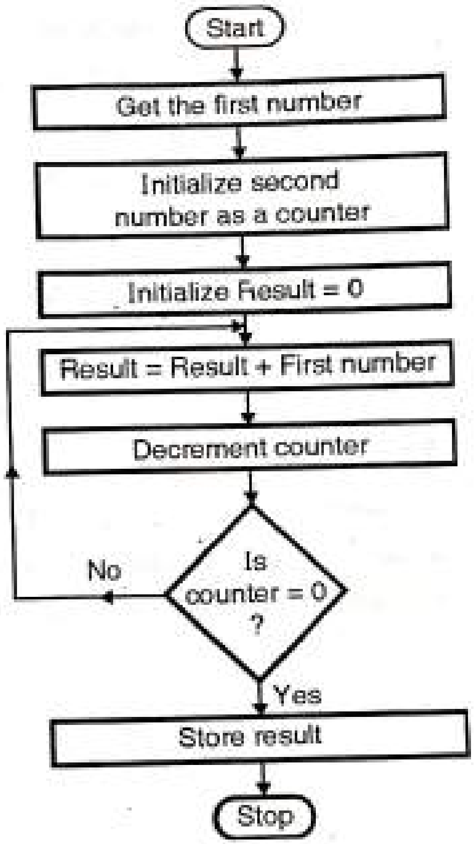
Result = 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H

Result = 0120 H

E000 H = 20 H

E001 H = 01 H

>> **Flow chart :** Refer Flowchart 7.



Flowchart 7

➤➤ **Program :**

Label	Instruction	Comment	Operation
	LDA D000H	A = first number	A = 12 H
	MOV E, A	E = first number	E = 12 H
	MVI D, 00 H	D = 00 H	D = 00 H
	LDA D001H	A = second number	A = 10 H
	MOV C, A	Initialize counter	C = 10 H
	LXI H, 0000H	Result = 0	H = 00H, L = 00H
BACK:	DAD D	Result = result + first number	HL = HL + DE
	DCR C	decrement count	C = C – 1
	JNZ BACK	If counter ≠ 0 repeat	
	SHLD E000H	Store result	E000H : 20H, E001H : 01H Result
	HLT	Terminate program execution	Stop

Ex. 8.5.7 : Divide 16 bit number by an 8 bit number

Program statement :

Divide 16 bit number stored in memory locations D000 H and D001 H by the 8 bit number stored at memory location D002 H. Store the quotient in memory locations E000 H and E001 H and remainder in memory locations E002 H and E003 H.

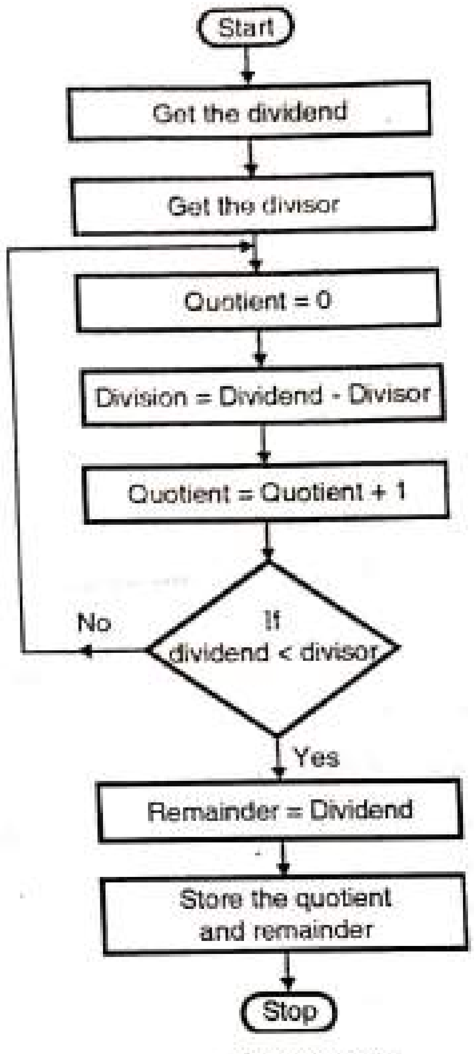
Explanation :

- Get the dividend in the HL register pair. Get the divisor in the accumulator and store it in register C.
- Initialize quotient in register pair DE as 00 H.
- Perform the division by subtracting the divisor from the dividend, till the dividend is greater than the divisor. Increment the quotient every time the dividend is greater than the divisor when the subtraction is performed.
- When the dividend becomes less than the divisor then this dividend is the remainder. Store the quotient and remainder.

Example :

D000 = 05 H
D001 H = 02 H
D002 H = 04 H
Result = 0205H/04H
= 81 H (Quotient)
and 01 H = Remainder
E000 H = 81 H
E001 H = 00 H
E002 H = 01 H
E003 H = 00 H

Flowchart : Refer Flowchart 9.



Flowchart 9

>> Program :

Label	Instruction	Comment	Operation
	LHLD D000H	Get the dividend	H = 02 H, L = 05 H
	LDA D002H	Get the divisor	A = 04 H
	MOV C, A	Store the divisor in C	C = 04 H
	LXI D, 0000H	Initialize Quotient = 0	D = 00 H,E = 00H
BACK:	MOV A, L		A = L
	SUB C	Division = Dividend – Divisor	A = A – C
	MOV L, A		L = A
	JNC SKIP		
	DCR H	Subtract borrow of previous subtraction	H = H – 1
SKIP:	INX D	quotient = quotient + 1	DE = DE + 1
	MOV A, H		A = H
	CPI 00	Check if dividend < divisor	
	JNZ BACK		
	MOV A, L	if no repeat	A = L
	CMP C		
	JNC BACK		
	SHLD E002H	Store the remainder	E002 H = 01 H, E003 H = 004H Result
	XCHG		
	SHLD E000H	Store the quotient	E000H = 81H, E001H = 00H Result
	HLT	Terminate program execution	Stop

Ex. 6.5.16 : Generation of fibonacci series **(PTU - Dec. 2006)**

Soln. :

Program Statement : Write an assembly language program to display fibonacci series

Explanation : The fibonacci series is,

1 1 2 3 5 8 13 21

The first two terms are 0, 1, the third term is computed as 0 + 1 = 1, fourth term = 1 + 1= 2, fifth term = 1 + 2 = 3 ... i.e. n^{th} term = $(n-2)^{th}$ term + $(n-1)^{th}$ term. We will initiatize the count in register D. Initialise register C with first term 01. Register B contains the number.

Add the two terms. Then store the first term. The next term is then computed making result equal previous number. The process is repeated till all the terms are calculated. The result will be,

1 1 2 3 5 8 0D 15 22.

Algorithm :

Step I : Initialize counter

Step II : Initialize variable B to store previous number.

Step III : Initialize variable C to store current number.

Step IV : Add two numbers.

Step V : Make current number = Previous number.

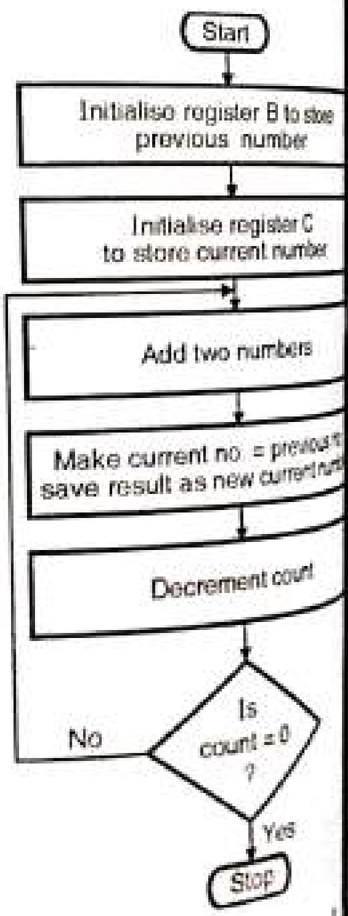
Step VI : Save result as new current number.

Step VII : Decrement count.

Step VIII : If count ≠ 0 go to step IV.

Step IX : Stop

Flowchart : Refer Flowchart 18



Flowchart 18

Program

Label	Instruction	Comments
	MVI D,count	Initialize counter
	MVI B,00	Initialize variable to store precious number.
	MVI C,01H	Initialize variable to store current number
	MOV A,B	
BACK :	ADD C	Add the two numbers
	MOV B,C	Make the current number is the previous number
	MOV C,A	Save result as a new current number
	DCR D	Decrement count
	JNZ BACK	If count ≠ 0 go to BACK
	HLT	Stop