Paper : Microprocessor and Computer Programming
Chapter: Microprocessor Programming

Author: SubhasisHaldar
College/ Department: Physics Department, Motilal Nehru College, University of Delhi

Microprocessor Programming for beginners

The target audience of this chapter are those students who has just began learning about computer and microprocessor. Assembly language programming is something which if not handled in proper way scares away the students and can cause permanent detraction towards it. So through this chapter I tried to present the basic concept and logic of assembly language programming in easiest form for the beginner.

I sincerely hope after going through above article students will be motivated to learn microprocessor and assembly language in detail.

I will eagerly wait for any suggestions/modifications from the vivid readers.

Table of Content

1. Basic Components of a computer
2. Program Addition
3. Few lines about memory
4. Assembly language programming
5. Few more lines about memory
6. Program Subtraction
7. Addition and Subtraction (Indirect Method)
8. Program Multiplication
9. Program Division
10. Program HCF of two numbers
11. Program LCM of two numbers
12. Program 16 bit Addition.

Basic components of a computer system: -

These are five basic components of a computer.These are:

1) Input: This is the component through which a user loads the computer with various data and instructions to carry out desired operation/function. Typical and most widely used input device is keyboard.

2) Arithmetic and Logic Unit: This is the main number crunching section. In this unit all the arithmetic and logical operations are performed over the raw data as per instructions which were loaded through input.

3) Control Unit: This unit provides necessary timing and signals, which are required by the ALU section to carry out the desired task in proper chronological order.

4) Memory: The job of this unit is to simply store all the instructions and data required to carry out a given task. It is different from other units in the sense that its size can be varied according to the need. Memory is nothing but piles of registers each of which can store any eight-bit number.

5) Output: This section provides the final outcome of the task carried out by the ALU section to the user. Most widely used output devices are Video Display Unit (VDU) and printers.

Generally the**ALU** section is combined with the control section and together they are called Central Processing Unit **(CPU).** The CPU when fabricated on a single chip is known as microprocessor Unit (MPU) or simply microprocessor.

Likewise the Input and output section can also be combined and is considered as part of peripheral device.

So a typical computers bare minimum requirement is one chip of microprocessor, one or few chips of input/output devices and few chips of memory.

Since the computers are made up of electronic components (mostly transistors) and since these electronic components work in only two states (saturation or cut-off), binary numbers are most suitable number base systems to work with the computers. So all the (i) data to be analyzed, all the (ii) instructions which are required to carry out a given task and all the (iii) memory location where the program has to be written must be in binary form. So computer understands only the binary language. Now the base of the binary number is very small (binary number base is 2) which means large number of digits are required to represent a decimal number. (e.g., decimal 41 is equivalent to 101001 in binary). Thus, it is very difficult to handle such a large numbers and chances of making error are very high. Therefore we look for a number system, which shall be easily convertible into binary number and simultaneously much smaller in size. Remember, it is much difficult to convert a decimal number into a binary number (This is because the base of decimal number is not an integral multiple of power of base of binary number). So we use an unusual number system, namely hexadecimal number system having base 16. Hexadecimal number base is integral multiple of power of binary base **($2^4 = 16$).**
Hexadecimal numbers are much smaller in size and in fact they are smaller than decimal numbers. Also they can be easily converted into binary numbers. What we need to do is to change every hexadecimal numeral into equivalent four bit binary numbers! That's all!

So instead of writing all the codes, instruction and data in binary format in processor what we can do is use hexadecimal numbers and gets them converted into binary format by the machine itself. This definitely is more advantageous as it greatly reduces the possible commencement of errors due to large number handling.

The programs written both in binary languages or hexadecimal languages are known as machine language. Figure1 shows an example of a typical machine language program where addresses, codes and datum are all written in hexadecimal format.

Memory Address	**Stored content of memory address**
2000H	3E
2001H	00
2002H	06
2003H	53
2004H	0E
2005H	2A
2006H	80
2007H	81
2008H	5F
2009H	76

An obvious question – what the program shown in fig 1.do? Well, as long as the person who had written the program doesn't tell, we can only make wild guesses. Just looking at the program it is not at all possible to find what the program is going to do. So merely using hexadecimal codes does not solve our entire problem where our primary aim is to make the programs more users friendly. It should be more clearly understandable. We can tackle this problem by writing some short of mnemonics or abbreviation for the codes of the instruction. Using suitable and meaningful mnemonics we can easily understand the flow of the program. That will certainly make it more user friendly. For instance, the program shown in fig .1 in hexadecimal code depicts simple addition of two numbers (53H and 2AH) stacked in two different registers B and C, with the final result stored in register E. The same program is rewritten along with mnemonics in figure 2 which make it moreunderstandable.

Program 1 (Addition)

Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	MVI A,00	3E
2001H		00
2002H	MVI B,53	06
2003H		53
2004H	MVI C,2A	0E
2005H		2A
2006H	ADD B	80

```
2007H           ADD C           81
2008H           MOV E,A              5F
2009H           HLT                  76
```

The programme written in mnemonics is called assembly language and the equipment attached with the microprocessor to convert these mnemonics to binary language is called assembler.

In case microprocessor does not have any assembler, we find out the codes of the mnemonics by looking at the prescribed table; what we call as hand assembling. Both machine language and assembly language are considered to be low level languages and mostly they are machine dependent.

Machine language is the most basic form of writing a program and it gives much deeper insight of how all the logical instructions are carried out step by step by the computer. Remember, whenever we write any program in any language, it is the logic behind the program which have to be translated in that language to carry out the specific task. Whatever be the language, logic of calculation remains the same. Forinstance, if I want to add two numbers, the methodology of addition will remain same irrespective of whether I am doing it in English or Spanish. So the important part of any program is writing the proper logic to carryout the particular task. The best way of expressing this logic is to draw the flowchart depicting all the basic instructions to be carried out by the computer in order to complete the task.Once flowchart is complete, the rest of the thing is very trivial where one has to convert all those basic instructions into a particular language, to be used.

Few lines about memory:

The memory capacity of **8085** microprocessor is **64K** or **65536**. In each of these location one can store any 8 bits number which may be data or instruction. In order to identify these locations uniquely we require separate addresses of all these locations. Now any single binary digit can be associated with two distinct locations (bit 0 can be considered as address of one location and bit 1 can be considered as address of the second location). Similarly different combinations of two bits can uniquely identify four different locations as shown below:

00	-	Location 1
01	-	Location 2
10	-	Location 3
11	-	Location 4

For two bits the number of combination which can be associated with memory locationsis **2^2 (=4).** Similarly if we extend this to three bits the number of combinations will be **2^3(= 8)** as given below

000	-	Location 1
001	-	Location 2
010	-	Location 3
011	-	Location 4
100	-	Location 5
101	-	Location 6
110	-	Location 7

111 - Location 8

So in order to identify 65536 different locations the number of bits required in the address can be calculated as below $2^n = 65536$ or **n** = 16.

So all the possible different combinations of 16 bit number will uniquely, identify all the **65536** different memory locations. (First address being all sixteen zeros and last address as all sixteen ones). Since sixteen bit addresses in binary format is difficult to handle, we convert these addresses in hexadecimal format with starting address as
0000H and last address being **FFFFH**.

The entire 65536 memory can be differentiated in two parts. Memory position starting with address 0000H and up to certain level depending on the make is called ROM (Read only memory) where generally the monitor program is stored.

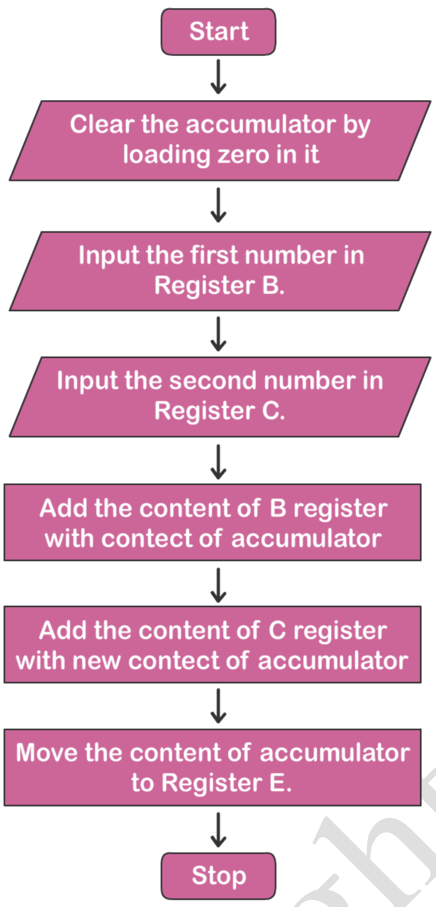
Program written in ROM cannot be altered and is permanent. After ROM, read/write location (Random Access memory – RAM) starts. These locations are available for the user to write different programs.You are free to use any address location in the RAM for writing the program.

Assembly Language Programming:

Now I am going to present few basic programmes and analyze them wherever it is needed. Looking at any table/or manual all the assembly language instruction can be converted into machine language. For student help I am going to write these programs in assembly language along with their machine codes.

Program-1: Addition (Direct Addressing)

Already I have written the program for addition. The flowchart of the program is presented below:


Start
Clear the accumulator by loading zero in it
Input the first number in Register B.
Input the second number in Register C.
Add the content of B register with contect of accumulator
Add the content of C register with new contect of accumulator
Move the content of accumulator to Register E.
Stop

Flowchart for addition

The program shows the simple addition of two numbers which are directly stored in two temporary registers B and C of the microprocessor. The first column in the above program shows the memory addresses where the program is stored beginning with address 2000H.

Few more lines about memory

Since all the arithmetic and logical operations are carried out with the content of accumulator it is customary to clear the content of accumulator at the very beginning. Note that I could have shortened the program by putting one of the numbers directly in the accumulator and adding the other number through any other register, but generally we avoid this method as after addition the final result will reside in the accumulator and thus initial data written in accumulator will be lost forever.

Instruction written in memory addresses 2000H, 2002H and 2004H are all two byte instructions which load the individual registers A (Accumulator) with zero and B and C with the numbers to be added. Instruction ADD B add the content of B register to the content of register A (which is zero)and it remains in register A. Then in the next instruction content of register C is added with the content of register A (which actually now contain the value of register B) and final result again resides in A. We can find the answer in register A but again it is customary to move the final result in any other register which is right now not in use (In this case the answer is moved to register E). Finally with 'HLT' instruction program terminates.

Program -2 Subtraction (Direct addressing). After the addition subtraction is very trivial. What we have to do is simply replace the command ADDC in memory address 2007H by SUBC. Rest of the instruction remains same.

One important point about subtraction is that if one subtract larger number from a smaller number the answer will be displayed in 2's complement form. The sign flag in the flag register will indicate that the result is negative.

Program -3 Addition and subtraction (Indirect addressing) In the above two programs the numbers to be added/ subtracted are directly stored in the registers (register B and register C). Addition and subtraction programs can be rewritten by storing the numbers in some memory address locations rather than storing them directly in temporary registers.

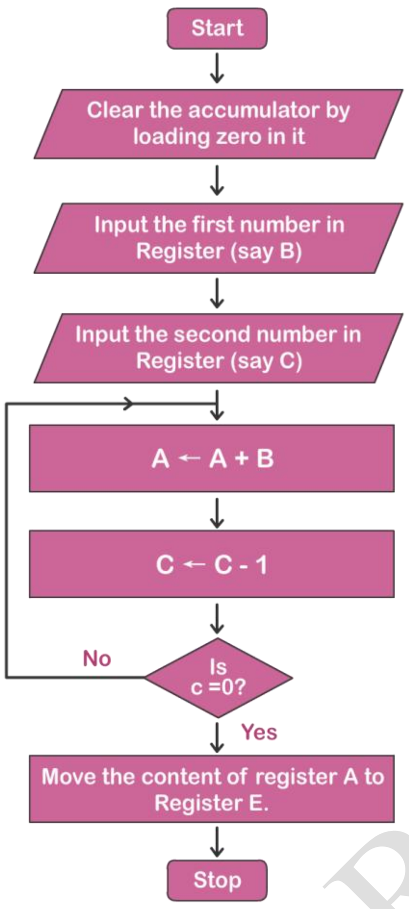
Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	LXI H, 2009	21
2001H		09
2002H		20
2003H	MOV A,M	7E
2004H	INX H	23
2005H	MOV B,M	46
2006H	ADD B/ SUB B	80/90
2007H	MOV E,A	5F
2008H	HLT	76
2009H	Data	
200AH	Data	

For subtraction again the instruction ADD B in memory location 2006 is replaced with SUB B. The advantage of the above type of program is that the main program is not altered. If we want to change the data for addition and subtraction it can be changed in the memory location without interfering the main program.

Instruction LXI H is a three byte instruction where a particular memory address (2009H in this case) is loaded to the HL register pair. In other words content of HL register pair actually points to the memory location where data resides. So the next instruction MOV A, M will automatically load the content of location 2009H to register A. The third instruction INX H will increment the memory pointer (i.e., the content of HL register pair) from 2009H to 200AH. Now the fourth instruction MOV B, M will move the data to register B from new memory location 200AH. Above programs can be slightly modified and we can use LDA (Load the accumulator) instead of LXI instruction. Instruction LDA 2009 will load the content of memory location to register A. Then from register A data can be transferred to any other register.

Program – 4 (Multiplication)

Any multiplication can be carried out by repeated addition, For example if we want to multiply 4 and 3, then what we have to do is simply either add 4 repeatedly three times or add 3 four times with itself. The flow chart of the multiplication and its assembly language program is written below:


Start
Clear the accumulator by loading zero in it
Input the first number in Register (say B)
Input the second number in Register (say C)
A ← A + B
C ← C - 1
Is c =0?
No
Yes
Move the content of register A to Register E.
Stop

Flowchart for multiplication

Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	MVI A,00	3E
2001H		00
2002H	MVI B,xx	06
2003H		xx
2004H	MVI C,yy	0E
2005H	ADD B	80
2006H	DCR C	0D
2007H	JZ 200D	CA
2008H		0D
2009H		20
200AH	JMP 2005	C3
200BH		05
200CH		20
200DH	MOV E,A	5F

```
200EH          HLT            76
```

The instruction JZ (Jump of zero) checks the zero flag of flag register and if it is set then instruction sequence will alter and will jump to instruction location 200DH. Otherwise, simple unconditional jump instruction (**JMP 2005**) will execute and addition process will be repeated.

Note that two jump instructions of 2007H (JZ) and 200A (JMP) can be combined by a single conditional branch instruction (JNZ 2005). It is left for the reader as an exercise.

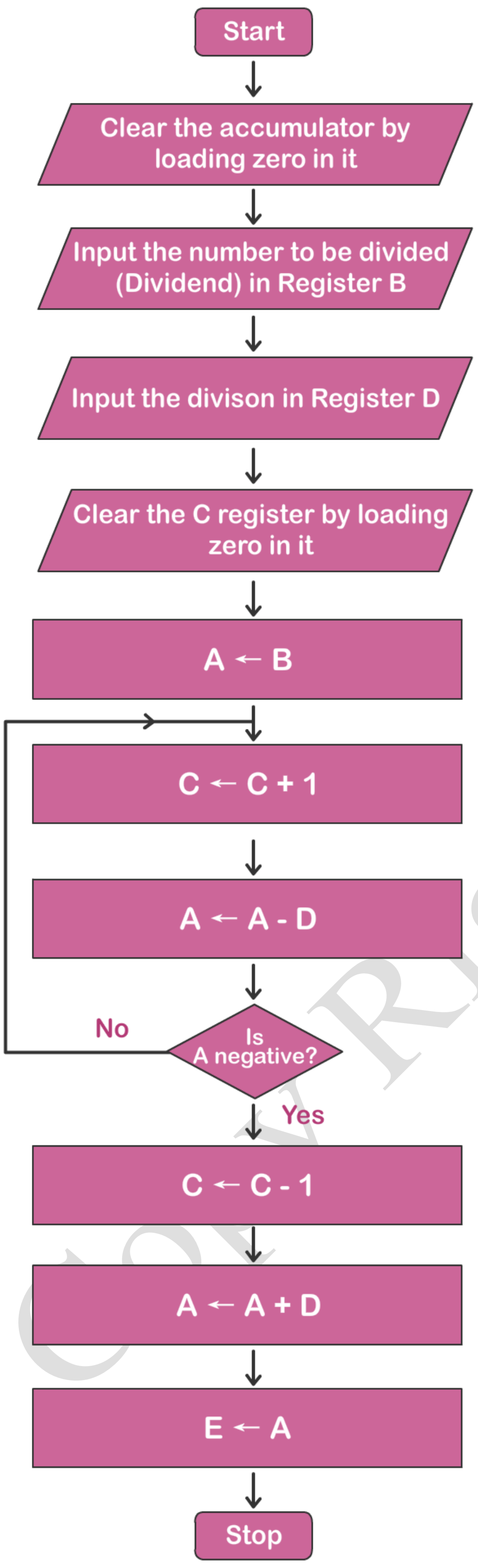
Program 5 (Division)

Division can be carried out by repeated subtraction. For example if we want to divide 15 by 4, then we subtract 4 repeated from 15 until we reach a number less than 4. The number of times 4 is subtracted gives the quotient and the remaining residue number is the remainder.

Dividend	Divisor	Counter for quotient
15	4	0
(15-4=) 11	4	1
(11-4=) 7	4	2
(7-4=) 3	4	3

The quotient is 3 and remainder is also 3.

So what we have to do is to set a counter and increment it every time we subtract the divisor from the dividend. The value of the counter at the end will be the quotient.


Start
Clear the accumulator by loading zero in it
Input the number to be divided (Dividend) in Register B
Input the divison in Register D
Clear the C register by loading zero in it
A ← B
C ← C + 1
A ← A - D
Is A negative?
No
Yes
C ← C - 1
A ← A + D
E ← A
Stop

Flowchart for division

Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	MV1 A, 00	3E
2001H		00
2002H	MV1 B, xx	06
2003H		xx
2004H	MV1 D, yy	16
2005H		yy
2006H	MV1 C, 00	0E
2007H		00
2008H	MOV A, B	78
2009H	INR C	0C
200AH	SUB D	92
200BH	JM 2011	FA
200CH		11
200DH		20
200EH	JMP 2009	C3
200FH		09
2010H		20
2011H	DCR C	0D
2012H	ADD D	82
2013H	MOV E, A	5F
2014H	HLT	76

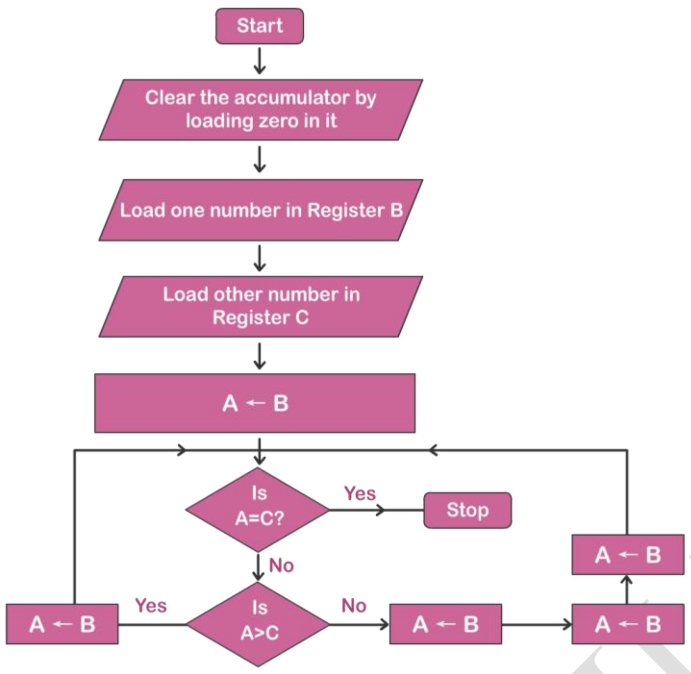
The content of register C will give quotient and content of register E will be the remainder. Instruction JM (Jump if minus) checks the sign flag after every subtraction is carried out in instruction 200A. If it is not set the subtraction is repeated again. If sign flag is set, the program comes out from the loop. Remember, we have to stop subtraction the moment dividend is smaller than divisor. But unfortunately we do not have any means to check it. So we carry on the subtraction as long as the dividend does not change sign. The moment dividend becomes negative we stop.At this juncture we actually carried out an extra subtraction. This is rectified by adding the divisor and decrementing the counter once as is done in the address location 2011 and 2012.

Program 6 HCF of two numbers

HCF means highest common factor. That is the largest number with which both the numbers can be divided. The simplest way to calculate HCF of any two numbers is given by subtraction method. What we have to do is to subtract the smaller number from the larger number and replace the larger number by the new number as long as the numbers are not equal.
Example : HCF of 16 and 24 can be obtained as follows :

16	24
16	24-16=8
16-8=8	8
8	**8**

So HCF of 16 and 24 is 8.


Start
Clear the accumulator by loading zero in it
Load one number in Register B
Load other number in Register C
A ← B
Is A=C?
Yes
Stop
No
Is A>C
Yes
A ← B
No
A ← B
A ← B
A ← B

Flowchart for HCF of two numbers

Memory Address	Mnemonics	Stored content of memory address
2000H	MV1 A, 00	3E
2001H		00
2002H	LDA 2500	3A
2003H		00
2004H		25
2005H	MOV B,A	47
2006H	LDA 2501	3A
2007H		01
2008H		25
2009H	SUB B	90
200AH	JZ 201AH	CA
200BH		1A
200CH		20
200DH	JM 2013H	FA
200EH		13
200FH		20
2010H	JMP 2009H	C3
2011H		09
2012H		20
2013H	ADD B	80
2014H	MOV C,B	48
2015H	MOV B,A	47
2016H	MOV A,C	79
2017H	JMP 2009H	C3
2018H		09
2019H		20
201AH	MOV E,B	58
201BH	HLT	76

2500H	data
2501H	data

In the above program we used indirect addressing so that the initial numbers for which we have to calculate HCF remains unchanged.

Exercise: Calculate HCF of any three number.

Program 7: LCM of two numbers

LCM of two number is that smallest number which will be perfectly divisible by both the numbers. So the best way to calculate LCM is to multiply any one number starting with 1 and check that whether the product is perfectly divisible by other number or not. If it is divisible then the product is LCM; if not we repeat the multiplication with next natural number.

Forexample: LCM of 8 and 12 can be calculated as follows:

$\frac{12X1}{8}= \frac{3}{2}$ (not perfectly divisible)

$\frac{12X2}{8}= \frac{24}{8}= \mathbf{3}$ (perfectly divisible)

SoLCM of 8 and 12 is 24.

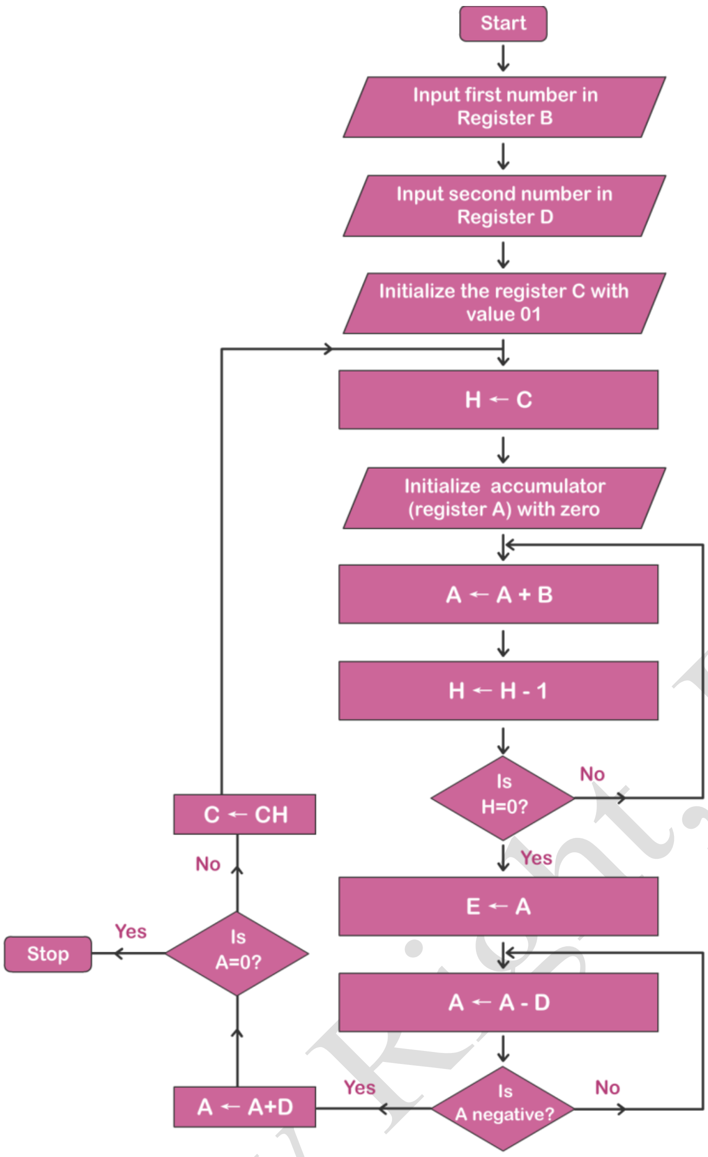
Similarly LCM of 3 and 4 can be calculated as follows:

$\frac{4X1}{3}= \frac{4}{3}$(not perfectly divisible)

$\frac{4X2}{3}= \frac{8}{3}$ (not perfectly divisible)

$\frac{4X3}{3}= \mathbf{4}$ (perfectly divisible)

So LCM of 4 and 3 is 12.


Start
Input first number in Register B
Input second number in Register D
Initialize the register C with value 01
H ← C
Initialize accumulator (register A) with zero
A ← A + B
H ← H - 1
Is H=0?
No
Yes
E ← A
A ← A - D
Is A negative?
No
Yes
A ← A+D
Is A=0?
Yes
Stop
No
C ← CH

Flowchart for LCM of two numbers

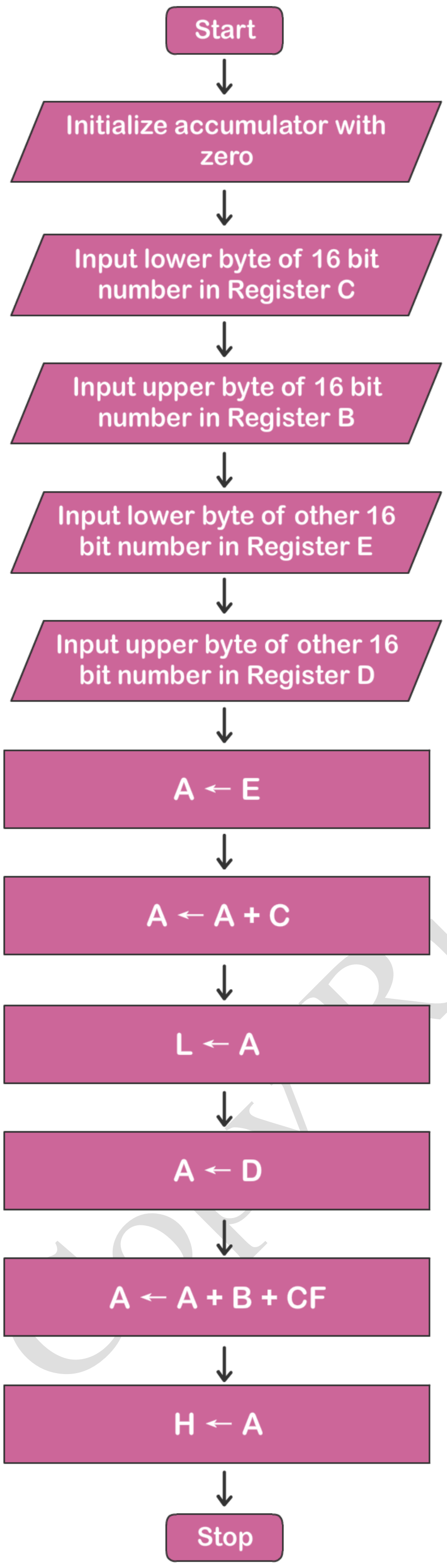
Memory Address	Mnemonics	Stored content of memory address
2000H	MV1 B,xx	06
2001H		xx
2002H	MVI D,yy	16
2003H		yy
2004H	MVI C,01	0E
2005H		01
2006H	MOV H,C	61
2007H	MVI A,00	3E
2008H		00
2009H	ADD B	80
200AH	DCR H	25
200BH	JNZ 2009H	C2
200CH		09
200DH		20
200EH	MOV E,A	5F
200FH	SUB D	92
2010H	JM 2016H	FA

2011H		16
2012H		20
2013H	JMP 200FH	C3
2014H		OF
2015H		20
2016H	ADD D	82
2017H	JZ 201EH	CA
2018H		1E
2019H		20
201AH	INR C	0C
201BH	JMP 2006H	C3
201CH		06
201DH		20
201EH	HLT	76

Exercise: Calculate LCM of three numbers.

Program 8: 16 bit addition

Intel 8085 microprocessor is an 8-bit microprocessor meaning that its data capacity is of 8 bits. So whatever arithmetic we have done so far deals with only one byte numbers. Now mathematics is extremely limited with 8-bit number as its range is too small to carry out any bigger problem (range of 8-bit signed number is from -128 to +127). To increase the range we can combine the contents of two registers. (Range of 16-bit number is 65536). Any 16-bit number is called a word. Two important commands, ADC (Add with carry) and SBB (Subtract with borrow) are required.


Start
Initialize accumulator with zero
Input lower byte of 16 bit number in Register C
Input upper byte of 16 bit number in Register B
Input lower byte of other 16 bit number in Register E
Input upper byte of other 16 bit number in Register D
A ← E
A ← A + C
L ← A
A ← D
A ← A + B + CF
H ← A
Stop

Flowchart for 16 bit addition

Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	MVI A,00	3E
2001H		00
2002H	MVI B,uu	06
2003H		uu
2004H	MVI C,vv	0E
2005H		vv
2006H	MVI D, xx	16
2007H		xx
2008H	MVI E,yy	1E
2009H		yy
200AH	MOV A,C	79
200BH	ADD E	83
200CH	MOV L,A	6F
200DH	MVI A,00	3E
200EH		00
200FH	MOV A,B	78
2010H	ADC D	8A
2011H	MOV H,A	67
2012H	HLT	76

In the above program we added one sixteen bit number uuvv with another sixteen bit number xxyy. The result has been transferred to register pair H and L. ADC instruction add the content of the register with accumulator along with the carry flag (CF) which reflects the carry condition from previous addition.

In a similar way the subtraction can be carried out using SBB instruction instead of ADC and is left as an exercise.

In 8085 microprocessor a special instruction DAD (Double addition) is available to carry out 16 bit additions that will make the above program much easier and shorter. But this instruction is available for addition only. For this instruction accumulator does not play any role and the content of register pair BC and DE can be directly added to the content of register pair HL.

Memory Address	**Mnemonics**	**Stored content of memory address**
2000H	MVI B,uu	06
2001H		uu
2002H	MVI C,vv	0E
2003H		vv
2004H	MVI D,xx	16
2005H		xx
2006H	MVI E, yy	1E
2007H		yy
2008H	MVI H,00	26
2009H		00
200AH	MVI L,00	2E
200BH		00
200CH	DAD B	09
200DH	DAD D	19
200EH	HLT	76

Addition of register pair BC and DE are added and answer will be available in HL register pair.

Questions

1. How many memory locations can be identified with 20 bit address lines?

2. In the multiplication program can we interchange the instructions written in memory locations 2005H and 2006H ? Give reasons to your answer.

3. What is the difference between LXI and LDA instruction?

4. Explain one byte, two byte and three byte instructions.

5. What is the difference between ROM and RAM?

6. Explain conditional and unconditional branching.

7. Explain the basic components of a microcomputer.

8. What is monitor program?

9. Can we perform any 16 bit operation with 8085 microprocessor? If yes how?

10. What do you mean by peripheral devices?

11. What is the difference between MPU and CPU?

12. What is the difference between machine language and assembly language ?

13. Why hexadecimal number system is advantageous over binary number system?

14. Why we use the unusual hexadecimal number system rather than conventional decimal number system?

15. What is the advantage of DAD instruction?

16. What is the significance of indirect addressing?

17. What is the difference between byte and word?

18. Why computer works only with binary numbers?

19. Can we use fractional number in 8085 microprocessor? Give reasons.

20. Write a program to calculate the largest number in the given array of numbers.

21. Write a program to calculate sum of n numbers present in some memory location.

22. Write a program to calculate HCF of three numbers.
23. Write a program to calculate LCM of three numbers.

24. Write a program to calculate $|x - y|$.

25. Write a program to separate odd numbers and even numbers in a given array.

Suggested Books

1. Digital Computer Electronics - An introduction to microcomputer by A.P. Malvino, Gregg Division, McGraw-Hill, 1983, Tata McGraw hill publ. con. Limited, 1985
2. Microprocessor Architecture, Programming and Applications with the 8085 byRamesh S. Gaonkar, Prentice Hall, 2002