LABORATORY MANUAL

M.P.M.C. Lab.
(B. Tech.)

SUIIT

Department of Electronics and Communication Engineering

Sambalpur University

Institute of Information Technolgy, Burla

ODISHA-768019

Department of Electronics & Communication Engineering-I
Curriculum of B. Tech (Electronics & Communication Engineering)

Credit: 2 **L-T-P:0-0-3** **Course Code: ECL248**

Objective: introduction to 8085 Microprocessor (architecture, addressing modes, instruction set)
Outcome: ability to program 8051, 8085 in assembly language.

*Introduction to 8085: Architecture, Addressing Modes, Instruction Set

a. Programming model of 8085 Microprocessor
b. Registers in 8085
c. Machine language/ Assembly Language
d. Assembly language commands/Instructions
e. Program format
f. Assembler introduction

1. Find the 1's and 2's complement of an 8-bit number stored in Memory.
    a. 8-bit number
    b. 16-bit number
2. Addition of two 8-bit number stored in Memory:
    a. Result is 8-bit,
    b. Result is 16-bit
3. Subtraction of two 8-bit number stored in Memory
    a. Using SUB instruction
    b. Without using SUB instruction
4. Addition of two 16-bit number
    a. Using DAD instruction
    b. Without using DAD instruction
5. Decimal Addition of two number stored in memory location
    a. Decimal Addition of two 8-bit numbers
    b. Decimal Addition of two 16-bit numbers
6. Move a Block of data from one section of memory to another section of memory
7. Multiplication and Division of two 8-bit numbers
    a. Multiplication of two 8-bit number stored in memory locations
    b. Division of two 8-bit number stored in memory locations
8. Finding the largest and smallest number in an array of 8-bit number
    a. Largest number in the array of 8-bit number
    b. Smallest number in the array of 8-bit number
9. Arrange the array of 8bit numbers in ascending /descending order
    a. Arrangement in ascending order
    b. Arrangement in descending number
10. Generation of Fibonacci Series of a specified length
11. Factorial of an8-bit number

12. Hexadecimal to BCD conversion and vice-versa
    a. Binary Code (Hexadecimal) to BCD conversion of an 8-bit number
    b. BCD to Binary Code (Hexadecimal) conversion of an 8-bit number
13. Finding the Square of a number using lookup table
14. 8051 programming
    a. Addition and subtraction of two 8-bit number stored in memory
    b. Multiplication and Division of two 8-bit number stored in memory
15. Speed control of DC motor.
16. (i) Square wave generator.
    (ii) Sawtooth wave generator.
17. Analog to digital conversions.

Reference Book:

1. Fundamentals of Microprocessors and Microcontrollers by B. Ram, Dhanpat Rai Publications [5.1, 5.2, Chapter-6, 9.9]
2. Microprocessor Architecture, Programming and Applications with the 8085, Ramesh Gaonkar, Penram International Publishing [Chapter-2, 6, 7, 8.1, 8.4, 10.1, 10.2,

MICROPROCESSOR & MICROCONTROLLER LAB

Category: Programme Core Course

Prerequisite: Knowledge on Digital Circuits and System

Course Objective: To provide knowledge on the 8085 microprocessor and their applications.

Course Outcome:

CO-1: Remember and understand the basic concepts/ Principles of Microprocessor and Microcontroller Lab

CO-2: Analyze the various concepts to understand them through case studies

CO-3: Apply the knowledge in understanding practical problems

CO-4: Execute / Create the project or field assignment as per the knowledge gained in the course

* Introduction to 8085 microprocessors (architecture, addressing modes, instruction set)

1. Addition of two 8-bit numbers, result is 8-bits and 16-bits.
2. Subtraction of two 8-bit numbers using sub instruction and without using sub instruction.
3. Addition of two 16-bit numbers.
4. (i) Decimal addition of two 8-bit numbers.
   (ii) Decimal addition of two 16-bit numbers.
5. (i) Find the multiplication of two 8-bit numbers.
   (ii) Find the division of two 8-bit numbers.
6. (i) 1's and 2's complement of an 8-bit number.
   (ii) 1's and 2's complement of a 16-bit number.
7. (i) Find the largest number in an array of 8-bit numbers.
   (ii) Find the smallest number in the array of 8-bit numbers.
8. (i) Arrange the array of 8-bit numbers in ascending order.
   (ii) Arrange the array of 8-bit numbers in descending order.
9. Find the square of an 8-bit number using look-up table.
10. Find the factorial of an 8-bit number.
11. Move a block of data from one section of memory to another section of memory.
12. Fibonacci series.
13. (i) Binary to BCD code conversion.
    (ii) BCD to binary code conversions.
14. Speed control of DC motor.
15. (i) Square wave generator.
    (ii) Saw-tooth wave generator.
16. Analog to digital conversions.
17. Microcontroller- addition, subtraction, division, multiplication

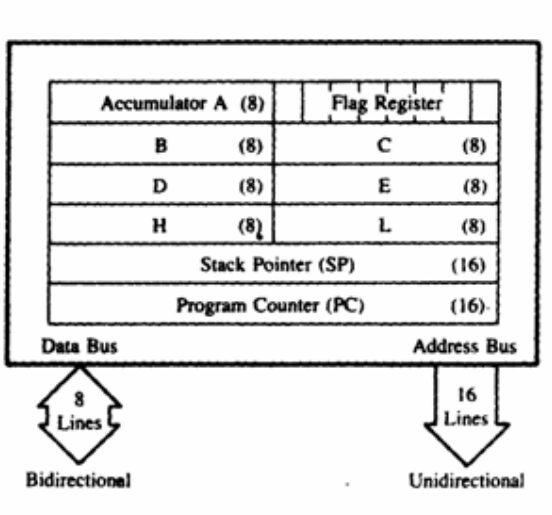
8085 Programming Model

What is a Microprocessor?

A Microprocessor is a multi-purpose programmable Digital VLSI chip that read instruction from memory and processes data as specified in the instruction.
So,

- It can understand a fixed set of basic commands known as instruction set.
- It has a limited set of *on-chip* memory locations, known as *registers,* to hold information (Data to be processed and intermediate results).
- The set of instruction (program) and data sets are stored in external memory.
- It can generate signals to synchronize & control external devices.

Programming Model



1. General Purpose Registers

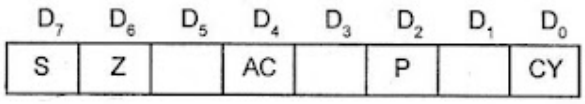
- The 8085 has six general purpose registers to store 8 bit data; these are identifies as B, C, D, E, H, L.
- They can be combined as register pairs - BC, DE and HL to store 16-bit information like address, data. These are represented as B, D and H respectively.
    - The 16-bit register pairs are used to hold 16-bit data or 16-bit address.
    - HL-pair is normally used to holds the address of a data memory location used as data pointer to store/read data from/to memory
- The programmer can use these registers to store or copy data into the registers by using data copy instructions.

2. Accumulator

- It is identified as register A.
- Itis an 8-bit register that is a part of arithmetic/logic unit(ALU).
- 8085 is an Accumulator based Micro-processor.
    - This register is used to store the 1st operand in arithmetic and logical operations.
    - The result of an operation is also stored in the accumulator.
- If not in use an 8-bit data can be stored in the register.

3. Flags

- 8085 has five Status flags and no control flags. Five flip flopsholds the status flag value arranged in an 8-bit register known as Flag register. The arrangement is shown in the figure below.



Bit positions of various flags in the flag register of 8085

The Flags are updated (Value is modified) after the execution of an arithmetic and logical instructions (with few exceptions).

Sign Flag (S): Sets or Resets based on the result stored in the accumulator.If the result stored is positive, the flag resets else if the result stored is negative the flag is set.

Sign bit is D7 bit of the result stored in accumulator. Sign flag is used in signed arithmetic to determine the result type.

0: Result is +ve
1: Result is negative

Zero Flag (Z): Sets or Resets based on the result stored in the accumulator. If the result stored is zero the flag is set else it is reset.

0: The result is not zero after the operation
1: The result is zero after the operation

Auxiliary Carry Flag (AC): This flag is set if there is a carry from lowernibble (lowest 4 bits) to uppernibble (upper 4 bits) or a borrow from upper nibble to lower nibble

0: No carry from D3 to D4 during an operation
1: A carry generated from D3 to D4 during operation

AC flag along with Carry flag are used in BCD arithmetic.

Parity Flag (P): This flag is set if there is even parity else it resets.

0: Odd number of 1 in the result stored in Accumulator.
1: Even number of 1 in the result stored in Accumulator

Carry Flag (CY): This flag is set if there is a carry bit else it resets.

0: No carry generated from D7 or no borrow to D7
1: A carry generated from D7 a borrow is taken to D7

4. Program Counter (PC)

- This 16-bit register deals with sequencing the execution of instructions this register is a memory pointer.
- Memory locations have 16 bit addresses and that is why this is a 16 bit register.
- The function of the PC is to point to the memory address from which the next byte is to be fetched.
- When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

5. Stack Pointer (SP)

- The stack pointer is also a 16-bit register used as a memory pointer.
- It always points to a top of the stack.
- The beginning of the stack is defined by loading 16-bit address in the stack pointer.
- It is automatically updated after any stack operation.

5. Address Bus (16-bit)

The width of the address bus is 16-bit. These lines are used by Microprocessor to send address of a memory location or Input out device to select it during data transfer.

6. Data Bus (8-bit)

The width of data bus is 8-bit. Data are transferred to the microprocessor or by the Microprocessor through these lines.

Aim of the experiment: Write an 8085 ALP to find 1's and 2's complement of an 8-bit number.

Instrument/Software required

PC with 8085 Simulator (GNU 8085)

Assumptions

The following memory locations are used to store data and results:

2000H : 8-bit data
2006H : 1's complement Result
2007H : 2's complement Result

Theory

1's Complement

The simplest way to obtain 1's complement is to toggle each bit of the number. In 8085 we can complement an 8-bit number present in Accumulator by using CMA instruction.

2's Complement

2's complement is obtained by adding 1 to 1's complement. The efficient way to adding 1 is to increment it by 1. 8085 support increment of an 8-bit number.

Algorithm/ Flow Chart

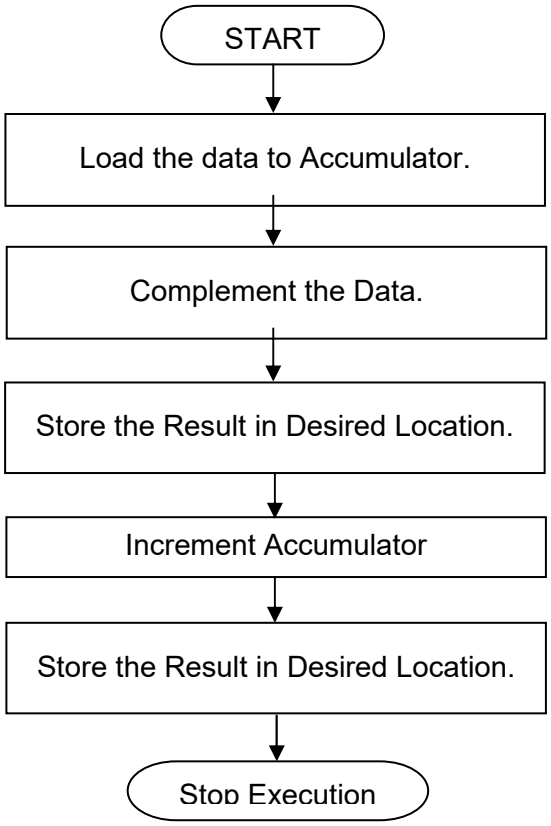


Fig: Flow chart of 1's and 2's complement of an 8-bit number

Procedure

1. Algorithm was developed from the program.
2. Each step of the algorithm is implemented with 8085 assembly language.
3. The program was assembled and syntax errors were corrected.
4. The program was tested with different types of data.
    a. Data wasmanually stored in the required memory location.
    b. Program was executed.
    c. The results were verified manually for its correctness.

Program

Experiment No. 1.a 1's and 2's complement of an 8-bit number
Name : Students Name
Roll No. : Complete Roll No.

Address	Opcode/ Operand	Label	Mnemonics	Comment
4200H	3A 00 20		LDA 2000H	; Data to accumulator
4203H	2F		CMA	; 1's complement
4204H	32 06 20		STA 2006H	; Store 1's complement Result
4207H	3C		INR A	; Compute 2's complement from 1's complement
4208H	32 07 20		STA 2007H	; Store 2's complement Result
420BH	76		HLT	; Stop Execution

Testing/Observations

Sl.	Data (2000H)	Results	
		1's Complement (2006H)	2's Complement 2007H
1.	23H(35)	DCH(220)	DDH(221)
2.	0FH(15)		
3.	F0H(240)		
4.	FFH(255)		
5.	00H(0)		

Verification

8-bit DATA	23H :	0 0 1 0 0 0 1 1
1's Complement	DCH :	1 1 0 1 1 1 0 0
2's Complement	DDH :	1 1 0 1 1 1 0 1

Conclusion

We learn how to write an assembly language program. We learn how to read and write data from memory. The program was assembled and tested with different data and found to be correct.

New Instructions used:

LDA $addr_{16}$: Load data to accumulator from specified memory address
STA $addr_{16}$: Load data from accumulator to specified memory address
CMA : Complement the Accumulator
INR r : Increment the specified register

Signature & Roll No. of the student

Aim of the experiment

Write an 8085 ALP to find 1's and 2's complement of 16-bit number stored in Memory.

Instrument/Software required

PC with 8085 Simulator (GNU 8085)

Assumptions

The following memory locations are used to store data and results:

2000H :	Data (LB)
2001H :	Data (UB)
2006H :	1's complement Result (LB)
2007H :	1's complement Result (UB)
2008H :	2's complement Result (LB)
2009H :	2's complement Result (UB)

Theory

8085 does not have any instruction to complement of a 16-bit number. So, 1's complement can be obtained by complementing each byte separately.

The 2's complement can be obtained by incrementing the 16-bit 1's complement number using INX instruction.

Algorithm/ Flow Chart

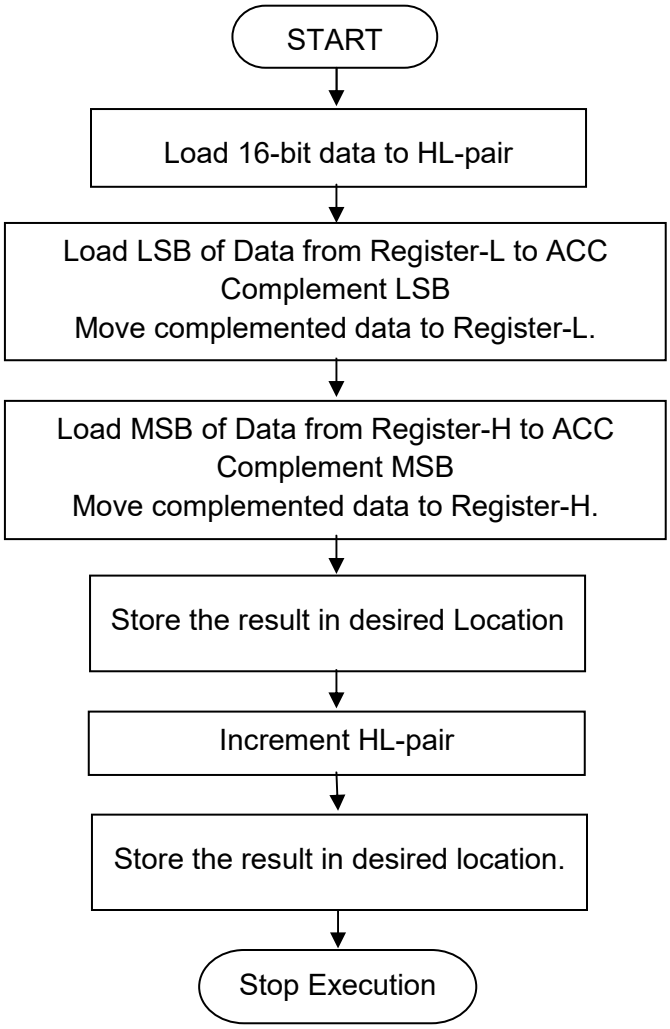


Fig: Flow chart for 1's and 2's complement of a 16-bit number.

Procedure

1. Algorithm was developed from the program.
2. Each step of the algorithm is implemented with 8085 assembly language.
3. The program was assembled and syntax errors were corrected.
4. The program was tested with different data.
    a. Data is stored in the required memory location.
    b. Program was executed.
    c. The results were manually verified for its correctness.

Program

Experiment No. 1.b 1's and 2's complement of a 16-bit number
Name : Students Name
Roll No. : Complete Roll No.

Address	Opcode/ Operand	Label	Mnemonics	Comment
4200H	2A 00 20		LHLD 2000H	; 16-bit Data loaded to HL-pair

4203H	7D		MOV A,L	; LB to Accumulator
4204H	2F		CMA	; 1’s complement of LB
4205H	6F		MOV L,A	
4206H	7C		MOV A,H	; UB to Accumulator
4207H	2F		CMA	; 1's complement of UB
4208H	67		MOV H,A	; 1’s complement in HL-pair
4209H	22 06 20		SHLD 2006H	; Store 1's complement Result
420CH	23		INX H	; Compute 2'S complement from 1'S complement
420DH	22 08 20		SHLD 2008H	;Store 2's complement Result
4210H	76		HLT	; Stop Execution

Testing/Observation

Sl.	Data		Results			
			1’s Complement		2’s Complement	
	Upper Byte (2001H)	Lower Byte (2000H)	Upper Byte (2007H)	Lower Byte (2006H)	Upper Byte (2009H)	Lower Byte (2008H)
1.	23H (35)	59H (89)	DCH (220)	A6H (166)	DCH (35)	A7H (167)
2.	00H (35)	FFH (89)				
3.	FFH (255)	00H (0)				
4.	00H (0)	00H (0)				
5.	FFH (255)	FFH (255)				

Verification

16-bit DATA	23 59H : 0 0 1 0	0 0 1 1	0 1 0 1	1 0 0 1
1’s Complement	DC A6H : 1 1 0 1	1 1 0 0	1 0 1 0	0 1 1 0
2’s Complement	DC A7H : 1 1 0 1	1 1 0 0	1 0 1 0	0 1 1 1

Conclusion:

New Instructions used:

LHLD $addr_{16}$: 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.

SHLD $addr_{16}$: 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.

INX rp: The contents of the designated register pair are incremented by 1 and the result is stored in the same place. The register pair may be B, D, H or SP.

Signature & Roll No. of the student

Aim : To write an 8085 ALP to add two 8-bit number that results in an8-bit number.

Equipment and software used :

1. Personal Computer
2. 8085-simulator

Assumption:

The data are stored in the memory locations as follows

2500_H➔ Augend
2501_H➔ Addend
2506_H➔ Sum

NOTE: The sum should be less than 255 (FF_H) to represent it by 8-bit.

Theory:

8085 Micro-processor supports 8-bit addition. To perform the addition operation one of the operands must be in Accumulator and the other one may be an immediate data, in a register or in a memory location. The result is stored in the Accumulator.

The 1st number is brought to Accumulator and added with the in a memory location.

Algorithm Flow Chart:

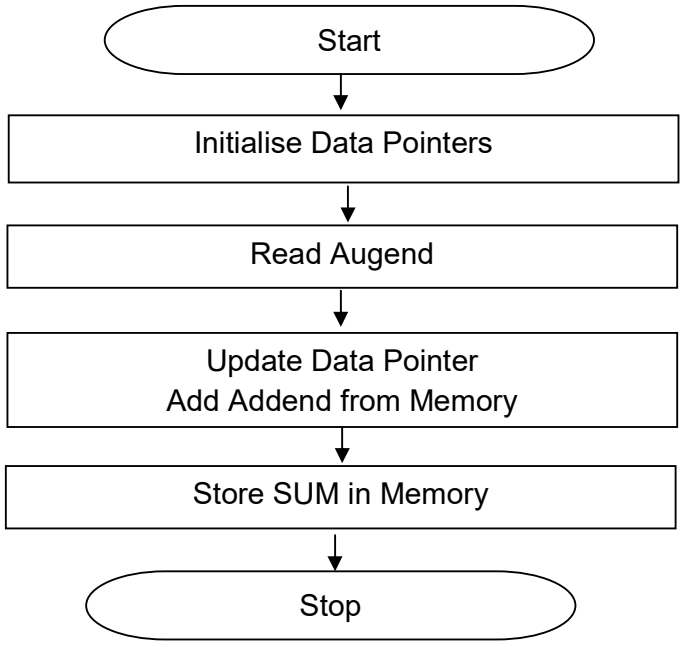


Fig: Flow chart of Addition of two 8-bit numbers Result 8-bit

Program :
Addition of two 8-bit number result 8-bit.

Address	opCode /Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H,2000H	; Data loaded to accumulator
4203	7E		MOV A,M	; Read Augend
4204	23		INX H	; Points to Addend
4205	86		ADD M	; Perform Addition
4206	32 06 020		STA 2006H	; Store the 8-bit SUM
4209	76		HLT	; Stop Execution

Testing of the program:

8-bit numbers to be added aremanuallystored in desired memorylocation.
Assembledthe program and run it.
Verify the results for different set of data.

Test results:

Sl.	Augend (2500_H)	Addend (2501_H)	Result (22506_H)
1			
2			
3			
4			
5			

Data	Decimal	Hex	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Augend	45	2D	0	0	1	0	1	1	0	1
Addend	99	63	0	1	1	0	0	0	1	1
Result	144	90	1	0	0	1	0	0	0	0

Conclusion:

The program was tested with different set of data and resultswere verifiedmanually and found correct. So, the ALP is working satisfactorily. We are now aware of some data movement instructionswith immediate addressing, register indirect addressing and direct addressing mode. We also became aware how to use addition instruction.

New Instruction used:

ADD r/M : Add the content of specified register or memory with the content of Accumulator and stored the sum in Accumulator. HL-register pair points the memory location where data is available(i.e.; the address of memory location must be available in HL-pair.)

Signature & Roll No. of the student

Aim: To write an 8085 ALP to add two 8-bit number that may results in a 16-bit number.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

The numbers to be added are stored in the following memory location.

2500_H➔ Augend
2501_H➔ Addend

2600_H➔Result (lower byte)
2601_H➔Result (upper byte)

Theory:

8085 Micro-processor supports 8-bit addition. To perform the addition operation one of the operand must be in Accumulator and the other one may be an Immediate Data, lies in a register or memory location. The result is stored in the Accumulator. When the sum is large enough to store the sum value in an 8-bit register, the least significant 8-bits are stored in the Accumulator, carry flag is set to 1. The carry represents the 9^{th} bit of the sum.

As the carry that represents the upper byte of sum can't be stored directly in memory. So, a register isused to store the upper byte of the sum. Initially it is initialised with "0" assuming no carry. If a carry is generatedduring the arithmetic operation the value of the register is modified to 1. If the carry is 0, the register value is not modified. The content of the register is stored as upper byte of SUM.

Algorithm:

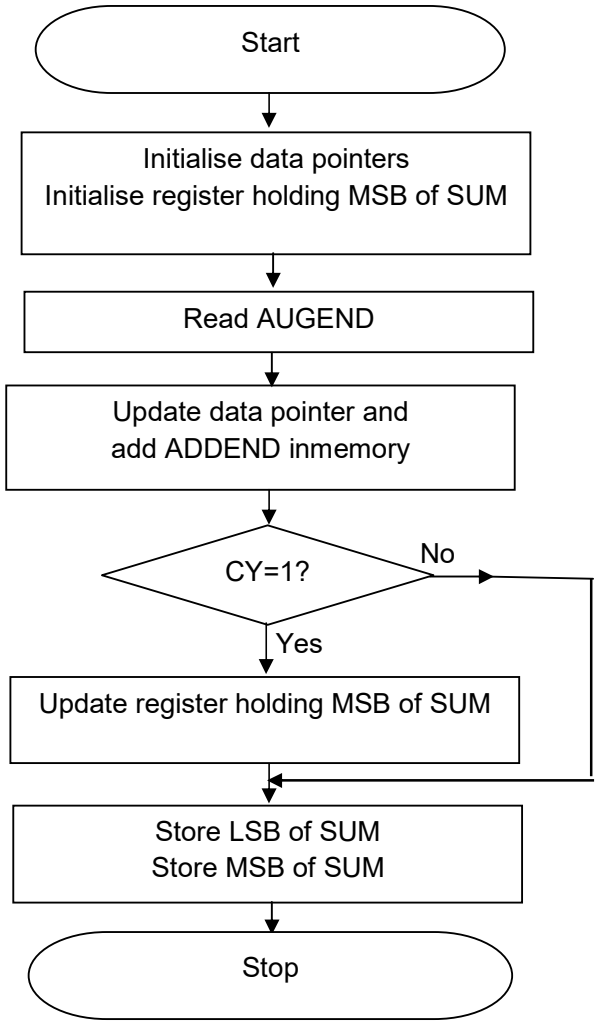


Fig: Addition of two 8-bit numbers, result may be 16-bit

Program:

Addition of two 8-bit numbers, result may be in 16-bit

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H,2000H	; HL to point data in memory
4203	11 06 020		LXI D, 2006H	; DE pair to point result location
4206	06 00		MVI B, 00H	; Holds the MSB of result
4208	7E		MOV A,M	; Read Augend
4209	23		INX H	; Points to Addend
420A	86		ADD M	; Perform addition and updates flags
420B	D2 0F 042		JNC SKIP	;Skip next Instruction if no carry generated
420E	04		INR B	; Update MSB of result
420F	12	SKIP:	STAX D	; Store LSB of Result from A
4210	13		INX D	; Points to next memory location
4211	78		MOV A,B	; MSB of result to Accumulator
4212	12		STAX D	; Store MSB of result
4213	76		HLT	; Stop Execution

Testing of the program:

- 8-bit numbers to be added aremanually stored in memory by editing memory content.
- Assembledthe program and run it.
- Verified the result for different set of data producing both 8-bit and 16-bit result.

Observations and verification:

Sl.	Augend (2500H)	Addend (2501H)	Sum	
			UB (2007H)	LB (2006H)
1.				
2.				
3.				
4.				
5.				

Verification:

Data	Decimal	Hex	D_{12-15}	D_{8-11}	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Augend	180	B4			1	0	1	1	0	1	1	1
Addend	108	6C			0	1	1	0	1	1	0	0
Result	288	01 20	0000	0001	0	0	1	0	0	0	0	0

Conclusion:

The program was tested with different set of data and results were verified and found correct. So, the ALP is working satisfactorily. We are now aware how to use conditional jump instructions to change the program flow sequence. We also learned to use pointers for data reading and result storing.

New Instruction used:

JNC Label: The execution of program branch to a location named **Label**(symbolic Address) if the carry flag is 0 i.e., Cy=0, otherwise executes sequentially.

Signature & Roll No. of the student

Aim: Write an ALP to subtract two 8-bit numbers using “SUB Instruction”

Equipment and software used:

1. Personal Computer
2. 8085-Simulator (GNUSim8085)

Assumptions:

3. The numbers to be added are stored in the following memory location.

2000_H➔Minuend

2001_H➔Sutrahend

2006_H➔Difference

Note:

Minuend ≥ Subtrahend to avoid a borrow.

Theory:

The instruction ***“SUB r/M”*** subtracts the content of specified register or memory location from the Accumulator. If the content in Accumulator is smaller, a borrow occurs and the carry flag is set to 1 else zero. All flags are affected according to the result in accumulator. If the 2^{nd} operand lies in memory, it must be pointed by the data pointer register *HL-Pair*.

Flow Chart:

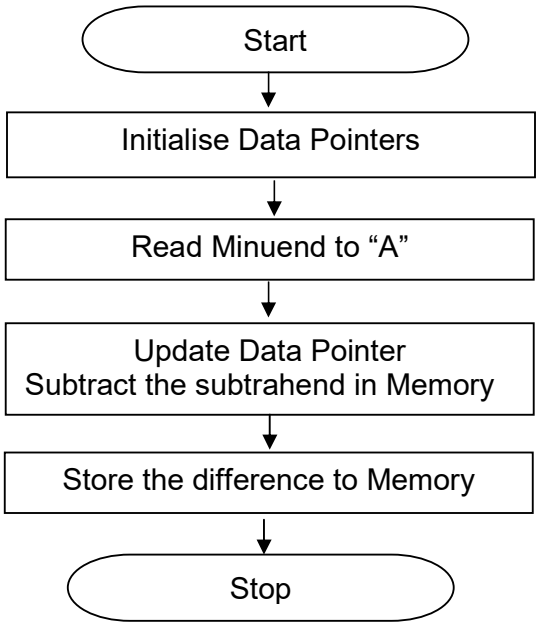


Fig: Subtraction of two 8-bit numbers using “SUB” instruction.

Program:

Subtraction of two 8-bit numbers using “SUB” instruction.

Address	OpCode /Operand	Label	Mnemonics	Remarks
4200	21 00 020		LXI H,2000H	; Initialise Data Pointer
4203	7E		MOV A,M	; Read Minuend
4204	23		INX H	; update to point Subtrahend
4205	96		SUB M	; Subtract subtrahend in memory
4206	32 06 020		STA 2006H	; Store Difference
4209	76		HLT	; Stop Execution

Testing of the program:

- 8-bit minuend and subtrahend are stored in the required memory locationsby editing memory content.
- Assembled the program and run it.
- Verified the result with different set of data.

Observations and Verification:

Sl.	Minuend (2000_H)	Subtrahend (2001_H)	Difference (2006_H)
1	64_H (100)	$2C_H$ (44)	38_H(66)
2			
3			
4			
5			

Data	Decimal	Hex	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Minuend	100	64	0	1	1	0	0	1	0	0
Subtrahend	44	2C	0	0	1	0	1	1	0	0
Difference	66	38	0	0	1	1	1	0	0	0

Conclusion:

The program was tested with different set of data and results were manually verified and found correct. So, the ALP is working satisfactorily.

New Instruction used:

SUB r/M: Subtract the content of specified register/ Memory from accumulator. All flags are affected. Cy=1 if a borrow is taken to MSB (D2). Memory location is pointed by HL-pair.

Signature & Roll No. of the student

Aim: Write an ALP to subtract two 8-bit numbers without using “SUB” instruction

Equipment and software used:

1. Personal Computer
2. 8085-simulator (GNUSim8085)

Assumptions:

1. The numbers to be added are stored in the following memory location.

2000_H➔Minuend

2001_H➔Sutrahend

2006_H➔Difference

> **Note:**
>
> Minuend ≥ Subtrahend (to avoid a borrow to MSB bit.).

Theory:

The subtraction of a number can also be performed by

i. Adding the 2’s complement of subtrahend to minuend.
ii. Decrementing Minuend repeatedly for subtrahend number of times.

In the 1st method, the 2’s complement is determined and added to Minuend. In 2nd method the minuend is repeatedly decremented for subtrahend number of times. So, we have to create a loop and in each iteration the minuend is decremented. The subtrahend is loaded to another register that acts as a counter. The counter register is also decremented in each iteration of the loop. Thus,minuend and subtrahend are decremented by one in each iteration. When counter register becomes zero, the Minuend has been decremented required number of times. So, loop ends when the value of counter register becomes zero, making *Zero Flag=1*. Now the content of accumulator will be the difference between the two numbers. The logic is shown in the flowchart.

Flow Chart:

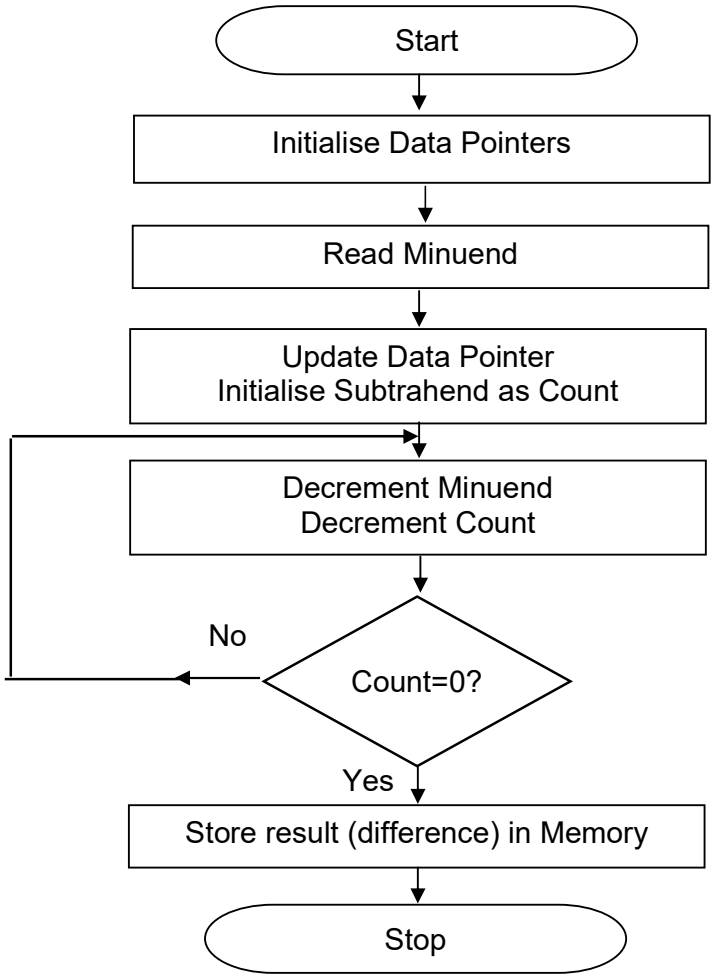


Fig: Subtraction of two 8-bit number without using "SUB" instruction

Program:

Subtraction of two numbers without using "SUB" instruction.

Address	OpCode /Operand	Label	Mnemonics	Remarks
4200	21 01 020		LXI H,2000H	; Data loaded to accumulator
4203	7E		MOV A,M	; Load Minuend
4204			INX H	; Update pointer to point subtrahend
4206	4F		MOV C, M	; Count initialised with subtrahend
4207	3D	LOOP:	DCR A	; Decrement Minuend
4208	0D		DCR C	; Count value updated
4209	C2 07 042		JNZ LOOP	; Repeat till count=0
420C	32 06 020		STA 2006H	; Store 1's complement Result

420D	76		HLT	; Stop Execution

Testing of the program:

- 8-bit minuend and subtrahend are stored in the required memory locationsby editing memory content.
- Assembled the program and run it.
- Verified the result with different set of data.

Observations and verification:

Test results:

Sl.	Minuend (2000_H)	Subtrahend (2001_H)	Difference (2006_H)
1	64_H(100)	$2C_H$(44)	38_H(66)
2			
3			
4			
5			

Data	Decimal	Hex	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Minuend	100	64	0	1	1	0	0	1	0	0
Subtrahend	44	2C	0	0	1	0	1	1	0	0
Difference	66	38	0	0	1	1	1	0	0	0

Conclusion:

An alternate method of subtraction. Here we learned the formation and ending of loop. In this program we also learned the use of default data pointer HL.

New Instruction used:

DCR r/M: Decrement the content of specified register and memory location. The result is stored in the source location. All flags except carry is modified,

JZ label: Jump to the "*address label*" if the zero flag is set to 1, otherwise continue execution sequentially.

Signature & Roll No. of the student

Aim: To write an 8085 ALP to add two 16-bit number using DAD instruction.

Equipment and softwareused:

1. Personal Computer
2. 8085-simulator

Assumptions:

The numbers to be added are stored in the following memory location.

2500_H➔ Augend (LB)
2501_H➔ Augend (UB)
2502_H➔ Addend (LB)
2503_H➔ Addend (UB)

2600_H➔SUM (LB/1st Byte)
2601_H➔SUM (2nd byte)
2601_H➔Result (UB/3rd Byte)

Theory:

8085 Microprocessor supports the addition of two 16-bit numbersusing "DAD r_p" instruction. The instruction uses the register HL-pair to store the 1st operand as well as theresult. The second operand may be stored in a register pair BC-, DE-. HL-pair or SP. Only carry flag is affected and CY=1 if sum is greater than 16-bit.

Flow Chart:

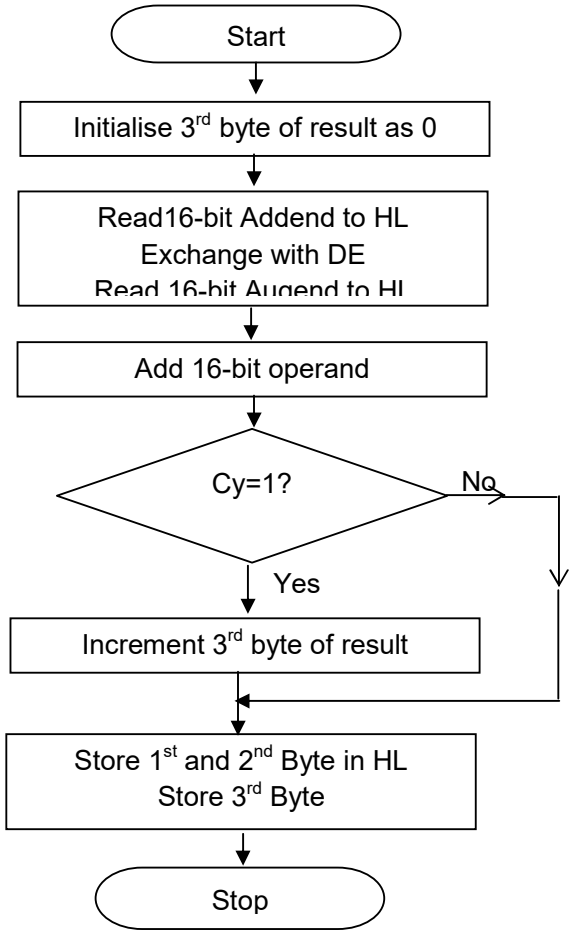


Fig: Addition of two 16-bit numbers using DAD instruction

Program:

Addition of two 16-bit numbers, result may be in 24-bit

Address	Opcode/ Operand	Label	Mnemonics	Comment
4200	AF		XRA A	; Clear Accumulator and Flags
4201	2A 02 020		LHLD 2002H	; Load Addend (16-bit data) to HL
4204	EB		XCHG	; Addend in DE register pair
4205	2A 02 020		LHLD 2000H	; Augend in HL pair
4208	19		DAD D	; 16-bit addition and carry update
4209	D2 0D 042		JNC NXT	; Skip updating of 3rd byte if CY=0
420C	04		INR B	; Update 3rd Byte
420D	22 06 020	NXT:	SHLD 2006H	; Store 16-bit Results
4210	32 08 020		STA 2008H	; Store 3rd byte of result (SUM)
4213	76		HLT	; Stop Execution

Testing of the program:

- 16-bit numbers to be added are stored in memorylocation by editing memory content.
- Assembledthe program and run it.
- Verified the result with different set of data producing 8-bit, 16-bit and 24-bit results.

Observations and verification:

Sl	Augend		Addend		SUM			Remarks
	UB (2001H)	LB (2000H)	UB (2003H)	LB (2002H)	MSB (2008H)	2nd Byte (2007H)	LSB (2006H)	
1.								
2.								
3.								
4.								
5.								

Verification:

99 86 H: 39302	1001 1001 1000 0110
86 90H: 34448	1000 0110 1001 0000
1 20 16 : 73750	1 0010 0000 0001 01100

Conclusion:

We learned to use the DAD instruction to add two 16-bit numbers at a time. The algorithm is similar to addition of two 8-bit numbers which uses the ADD instruction. We also know the technique of storing the carry flag.

Signature & Roll No. of the student

Aim: To write an 8085 ALP to add two 16 bit number without using DAD instruction.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

The numbers to be added are stored in the following memory location.

2500_H➔ Augend (LB)
2501_H➔ Augend (UB)
2502_H➔ Addend (LB)
2503_H➔ Addend (UB)

2600_H➔Result ()
2601_H➔Result (2^{nd} byte)
2601_H➔Result ()

Note:
LB : Lower Byte
UB : Upper Byte

Theory:

8085 can add two 8-bit numbers with or without carry. To add multi-byte numbers (16-bit, 32-bit etc.), addition is performed byte by byte. The lowermost byte is added without carry. Then the higher bytes are added along with carry, so that carry generated in a byte is added to next upper byte. After the addition of uppermost byte, the carry flag is tested and a register is updated accordingly.

Flow Chart:

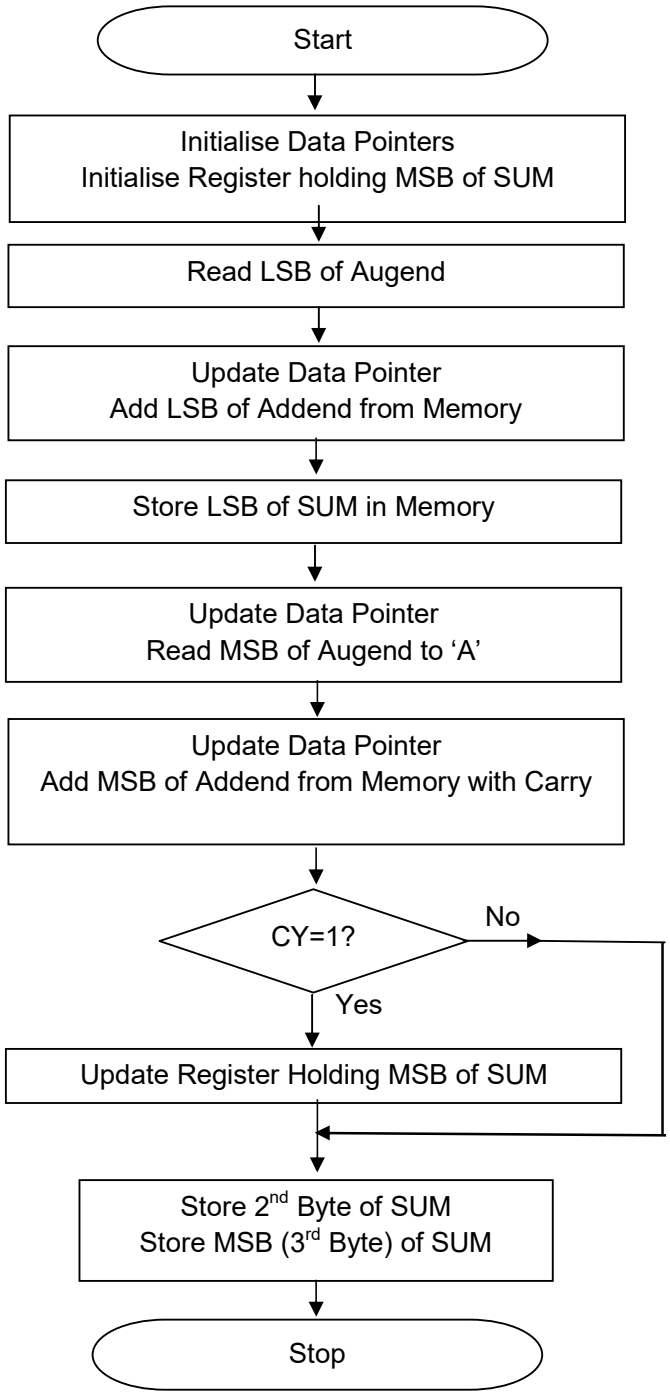


Fig: Addition of two 16-bit numberswithout using DAD instruction

Program:

Addition of two 8-bit numbers, result may be in 16-bit

Address	Opcode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Points data Location
4203	06 00		MVI B, 0	; Initialise the 3rd byte of SUM to '0'
4205	7E		MOV A,M	; Load LSB of Augend
4206	23		INX H	
4207	23		INX H	; Points to LSB of Addend
4208	86		ADD M	; ADD LSB of two number
4209	32 06 020		STA 2006H	; Store LSB of Result
420C	2B		DCX H	; Points MSB of Augend
420D	7E		MOV A,M	; Load MSB of Augend to A
420E	23		INX H	
420F	23		INX H	; Points to MSB of Addend
4210	8E		ADC M	; Add MSB along with Carry
4211	D2 15 42		JNC NXT	; Skip updating of 3rd byte if CY=0
4214	04		INR B	; Update 3rd Byte
4215	32 07 020	NXT:	STA 2007H	; Store 2nd Byte Results
4218	78		MOV A,B	; 3rd byte to Accumulator
4219	32 08 020		STA 2008H	; Store 3rd byte of result (SUM)
421C	76		HLT	; Stop Execution

Testing of the program:

- 16-bit numbers to be added are stored in memorylocation by editing memory content.
- Assembledthe program and run it.
- Verified the result with different set of data producing 8-bit, 16-bit and 24-bit results.

Observations and verification:

Sl	Augend		Addend		SUM			Remarks
	UB (2001H)	LB (2000H)	UB (2003H)	LB (2002H)	UB (2008H)	2nd Byte (2007H)	LB (2006H)	
1.								
2.								
3.								
4.								
5.								

Verification:

99 86 H: 39302	1001 1001 1000 0110
86 90H: 34448	1000 0110 1001 0000
1 20 16 : 73750	1 0010 0000 0001 01100

Conclusion:

We learned to use of data pointer.The same logic can be extended to perform addition of any length of number e.g. 32-bit, 64-bit..

New Instruction used:

ADC r,M : Add Accumulator with the specified register or memory location along with carry. The memory location is pointed by HL-pair.

Signature & Roll No. of the student

Aim: To write an 8085 ALP to add two 8-bit decimal numbers.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H : Augend
2002H : Addend

2006H : SUM (LB)
2007H : SUM (UB)

Theory:

A decimal number are represented in digital system with BCD code. An BCD code digit is represented by 4-bit. Each byte consists of two decimal digits. So, one byte holds two decimal digits. The range of number for 8-bit data is from 00_{10}-99_{10} .

In 8085 DAA instruction is available to adjust the binary sum to correct BCD sum. So, the decimal addition is performed by performing binary addition using any addition instruction (ADD r/M, ADC r/M, ADI $data_8$, ACI $data_8$). After the binary addition is performed, DAA instruction is executed to made necessary correction to convert the binary sum to BCD sum.

NOTE:

The addition of BCD numbers is performed as a binary addition followed by correction by using DAA instruction. If sum of two digit is greater than 9, 6 is added to it else 0 is added to it. So, depending upon the situation, 00, 06, 60 or 66 is added to the binary sum to get the correct decimal (BCD) sum. This is accomplished by executing the DAA instruction after any addition instruction.

The DAD rp instruction can't be used for decimal addition as it does not update AC (Auxiliary Carry) flag that is used by DAA during necessary correction.

This instruction cannot be used to convert the binary number to decimal number (i.e., HEX to BCD conversion).

Flow Chart:

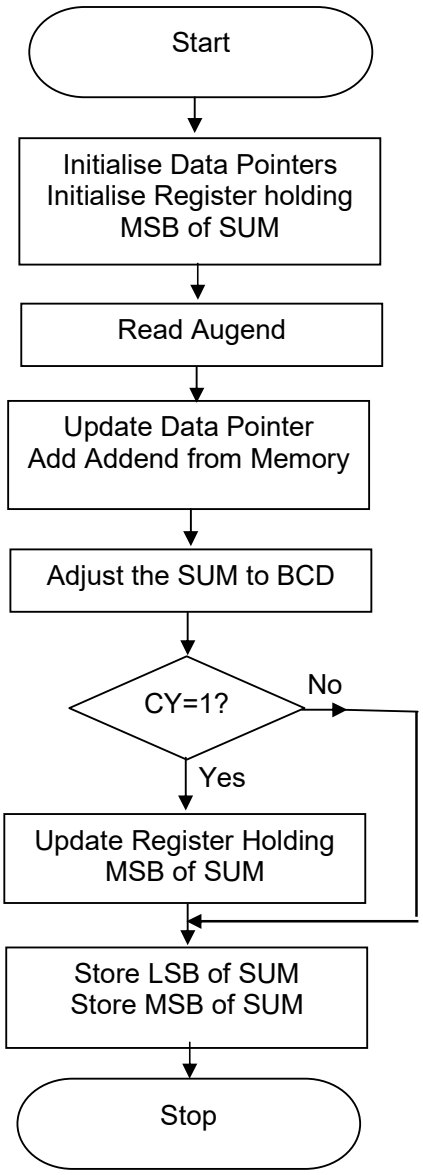


Fig: Addition of two 8-bit Decimal numbers

Program:

Addition of two 16-bit BCD (Decimal)numbers, result may be in 16-bit

Address	OpCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Points to data Location
4203	11 06 20		LXI D,2006H	; Initialise destination pointer
4206	06 00		MVI B,00H	; Most Significant Byte of result
4208	7E		MOV A,M	; Load LB of Augend
4209	23		INX H	; Points to LB of Addend
420A	86		**ADD M**	; ADD LB
420B	27		**DAA**	; Convert the result to correct BCD
420C	D2 10 42		JNC NEXT	; Skip Updatingof UB if Cy=0
420F	04		INR B	; Update UB (3rd Byte)
4210	12	NEXT:	STAXD	; Store the 2nd byte Result
4211	78		MOV A,B	; Upper; Update destination pointer Byte to Accumulator
4212	13		INX D	; Update destination pointer
4213	12		STAX D	; Store Upper Byte
4214	76		HLT	; Stop Execution

Testing of the program:

- 16-bit Decimal numbers to be added are stored in memorylocation by editing the required memory location.
- Assembledthe program and run it.
- Verified the result with different set of data producing 8-bit, 16-bit and 24-bit Decimal results.

Observation:

Sl	**Augend**	**Addend**	**SUM**		**Remarks**
	(2001H)	(2002H)	MSB (2007H)	LSB (2006H)	
1.	99H(153)	99H(153)	1	98H(152)	
2.					
3.					
4.					

5.					

Verification:

99:		1001 1001	BCD
99:		1001 1001	BCD
1 32 :	1	0011 0010	Binary Addition in ALU
66		0110 0110	Adjustment by DAA (Cy=1 and AC=1)
1 98 :	1	1001 1000	

Conclusion:

In this program we learn the addition of Decimal number. During addition of decimal number instruction DAA is used after any addition instruction. DAA uses the Cy and AC flags to convert the result to true BCD sum.

New Instruction used:

DAA : This stands for Decimal adjust accumulator. This instruction made necessary correction in result in BCD addition. The instruction uses Cy, AC flags and invalid BCD code to make the correction.

Signature & Roll No. of the student

Aim: To write an 8085 ALP to add two 16 bit decimal numbers.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H :	LB of Augend
2001H :	UB of Augend
2002H :	LB of Addend
2003H :	UB of Addend
2006H :	1st byte of SUM (LB)
2007H :	2nd byte of SUM
2008H :	3rd byte of SUM (UB)

Theory:

A decimal number are represented in digital system with BCD code. An BCD code digit is represented by 4-bit. Each byte consists of two decimal digits. So, one byte holds two decimal digits. The range of number for 16-bit data is from 0000_{10}-9999_{10} .

In 8085 DAA instruction is available to adjust the binary sum to correct BCD sum. So, the decimal addition is performed by performing binary addition using any addition instruction (ADD r/M, ADC r/M, ADI $data_8$, ACI $data_8$). After the binary addition is performed, DAA instruction is executed to made necessary correction to convert the binary sum to BCD sum.

NOTE:

The addition of BCD numbers is performed as a binary addition followed by correction by using DAA instruction. If sum of two digit is greater than 9, 6 is added to it else 0 is added to it. So, depending upon the situation, 00, 06, 60 or 66 is added to the binary sum to get the correct decimal (BCD) sum. This is accomplished by executing the DAA instruction after any addition instruction.

The DAD rp instruction can't be used for decimal addition as it does not update AC (Auxiliary Carry) flag that is used by DAA during necessary correction.

This instruction cannot be used to convert the binary number to decimal number (i.e., HEX to BCD conversion).

Flow Chart:

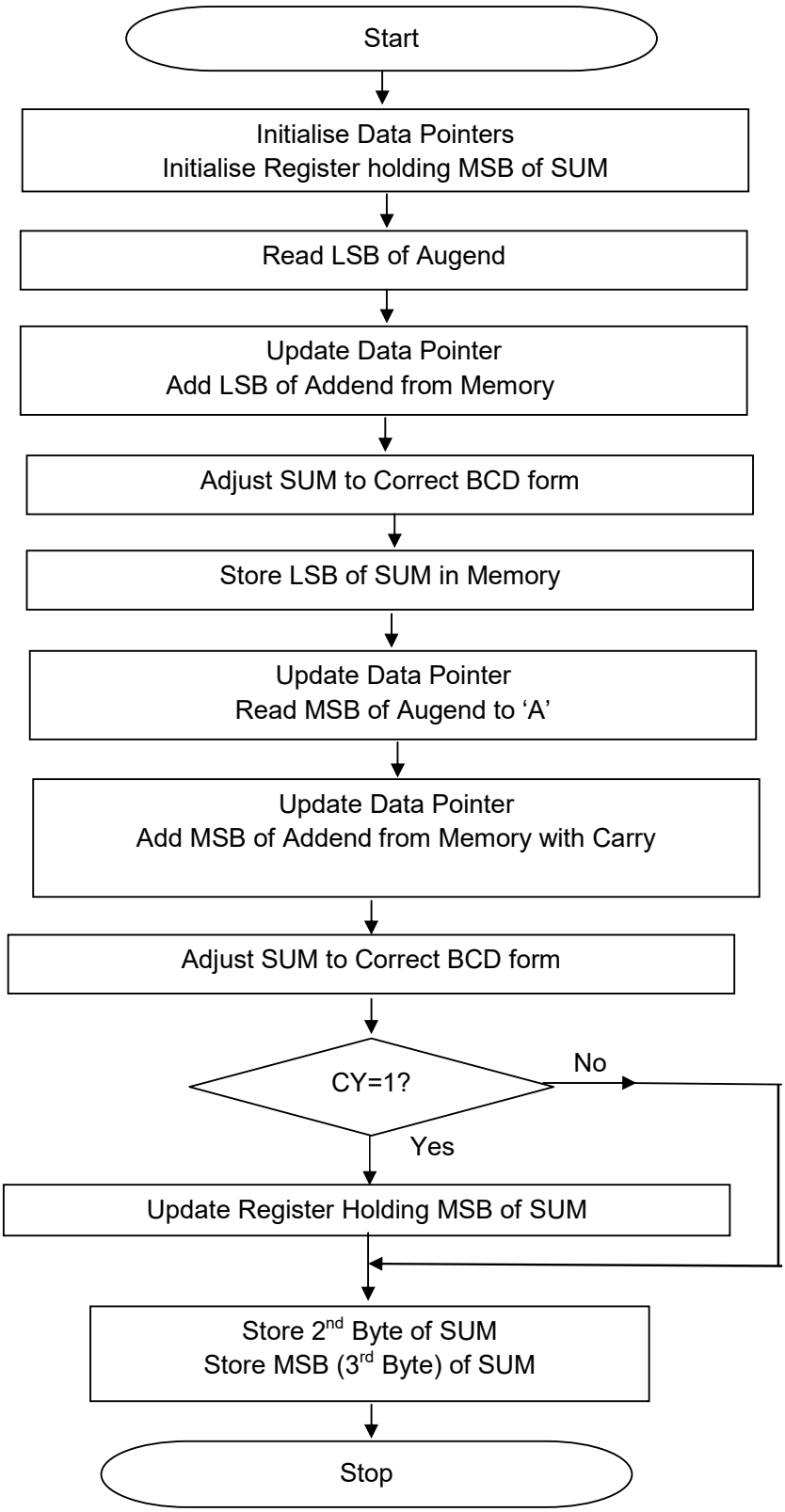


Fig: Addition of two 16-bit Decimal numbers

Program:

Addition of two 16-bit BCD (Decimal)numbers, result may be in 16-bit

Address	OpCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Points to data Location
4203	06 00		MVI B,00H	; Most Significant Byte of result
4205	7E		MOV A,M	; Load LB of Augend
4206	23		INX H	
4207	23		INX H	; Points to LB of Addend
4208	86		**ADD M**	; ADD LB
4209	27		**DAA**	; Convert the result to correct BCD
420A	32 06 020		STA 2006H	; Store LB of Result
420D	2B		DCX H	; Points UB of Augend
420E	7E		MOV A,M	; Load UB of Augend
420F	23		INX H	
4210	23		INX H	; Points to UB of Addend
4211	8E		**ADC M**	; Add UB along with Carry
4212	27		**DAA**	; Convert result to correct BCD
4213	D2 17 42		JNC NEXT	; Skip Updation of UB if Cy=0
4216	04		INR B	; Update UB (3rd Byte)
4217	32 07 020	NEXT:	STA 2007H	; Store the 2nd byte Result
421A	78		MOV A,B	; Third Byte to Accumulator
421B	32 08 020		STA 2008H	; Store 3rd Byte (UB)
421E	76		HLT	; Stop Execution

Testing of the program:

- 16-bit Decimal numbers to be added are stored in memorylocation by editing the required memory location.
- Assembledthe program and run it.
- Verified the result with different set of data producing 8-bit, 16-bit and 24-bit Decimal results.

Observation:

Sl	**Augend**		**Addend**		**SUM**			**Remarks**
	UB (2001H)	LB (2000H)	UB (2003H)	LB (2002H)	MSB (2008H)	2nd Byte (2007H)	LSB (2006H)	
1.	23H (35)	50H (80)	21H (33)	10H (16)	00H (0)	44H (68)	60H (96)	$2350_{10}+2110_{10}=4460_{10}$

2.	99H (153)	86H (134)	86H (134)	90H (144)	01H (01)	86H (134)	76H (118)	$9986_{10}+8690_{10}$ $=18676_{10}$
3.								
4.								
5.								

Verification:

99 86 H:		1001 1001 1000 0110	
86 90H:		1000 0110 1001 0000	
1 20 16 :	1	0010 0000 0001 01100	
66 60		0110 0110 0110 0000	Adjustment by DAA
1 86 76 :	1	1000 0110 0111 01100	

Conclusion:

In this program we learn the addition of Decimal number. During addition of decimal number instruction DAA is used after any addition instruction. DAA uses the Cy and AC flags to convert the result to true BCD sum.

Signature & Roll No. of the student

Aim of Experiment:

To write an 8085 ALP to move a Block of Data from one section of memory to another section of memory.

Equipment and software used:

1. Personal Computer
2. 8085-simulator (GNUSim8085)

Assumptions:

2000H : Length of the memory block

2001H : 1st element of data in the array

3000H : Length of the memory block

3001H : 1st element of data moved

The memory location of last element in the memory block depends upon the length of array. If 5 is stored in 2000H location, memory block to be transferred is from 2001H to 20005H. If 0AH than memory block is from.2001H to 200AH. The size of the memory block available in 2000H location is also transferred to the location prior to destination location of memory block.

Theory:

The data block is preceded by the length of data to be moved. The length as well the data in the specified memory block has to be moved to the new memory location. So, two data pointers are required, one to point to source location and another to point destination locations.

The length of data block is read to an internal register of the processor (using the source data pointer) and stored it prior to destination location)using destination pointer). This length of memory block is also used to initialize the counter.

Then the data are transferred one by one using loop. In the loop, the source pointer and destination data pointer are incremented by 1 to point the next memory location. The content of counter is decremented when a byte of data is transferred to desired location. The content of counter shows the number of data bytes remains for transfer.

The counter is checked whether all data elements are moved or not. when all data elements are transferred, counter reaches to zero and zero flags become 1. So, the program control goes out of the loop and the program terminates.

.

Flow Chart:

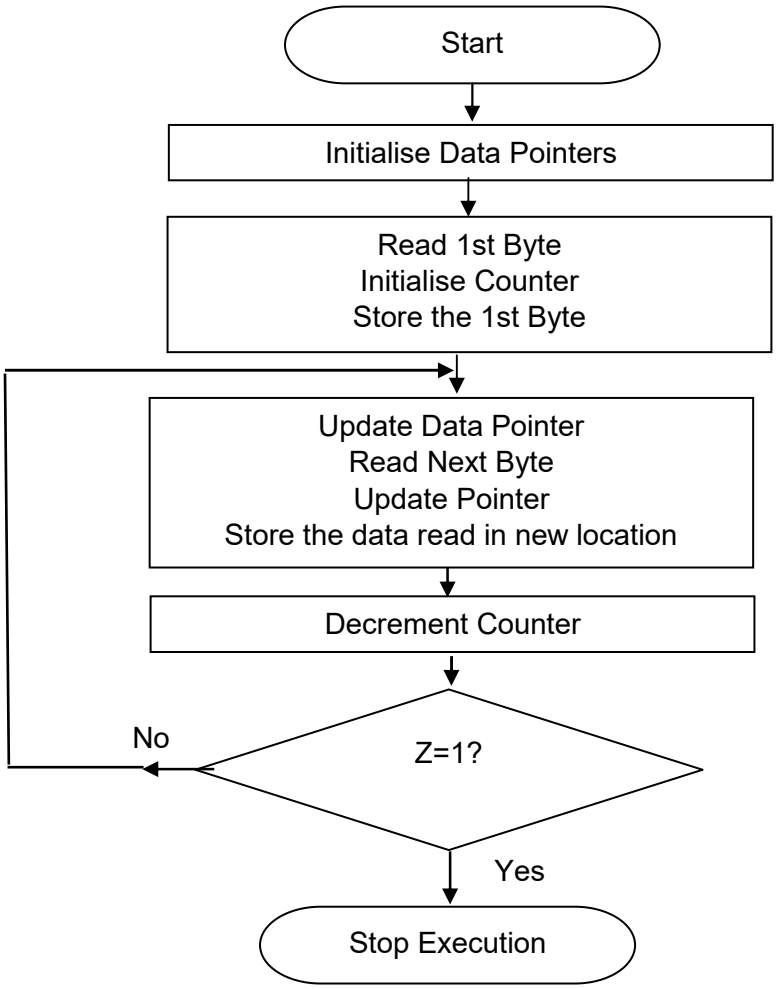


Fig: Flow chart to move a block of data from one memory locations to another memory locations

Program:

Program for "Movement of a block of Data from one memory location to another memory location".

Address	OpCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Source data pointer
4203	11 00 030		LXI D,3000H	; Destination Data Pointer
4206	4E		MOV C,M	; COUNT = Length of Memory Block
4207	79		MOV A,C	; Length of data block to 'A'
4208	12		STAX D	; Store length of data byte in Destination location
4209	23	Loop:	INX H	; Points to next source location
420A	13		INX D	; Points to next Destination location
420B	7E		MOV A,M	; Read Next Element
420C	12		STAX D	; Store the data into destination location

420D	0D		DCR C	; Update counter
420E	C2 09 042		JNZ Loop	; Continue till all elements are moved
4211	76		HLT	; Stop Execution

Testing of the program:

- Program was assembled.
- Source Data and length of memory block are manually stored in the desired memory locations.
- Program was executed.
- Destination memory locations are verified for data transfer to appropriate location and for required length.
- The sequence is repeated for different size of memory block.

Observations and verification:

Sl.	Source data and length of memory block	Source data and length of memory block
1.	2000H: 5 2001H-2005H:	3000H: 5 3001H-2005H:
2.	2000H: 0A H 2001H-200AH:	3000H: 0AH 3001H-200AH:
3.	2000H: 0DH 2001H-200DH:	3000H: 0DH 3001H-200DH:

Conclusion:

Learn the data transfer of any length using two different data pointers.

Signature & Roll No. of the student

Aim:

Generation of Fibonacci series of specified length.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H : Length of the series

2001H : 1st element of the series

The length of series is restricted to 14 to keep the data size one byte only.

Theory:

Fibonacci sequence is a series of numbers in which each number (Fibonacci number) is the sum of the two preceding numbers. Beginning with the first two numbers of the sequence 0,1 the remaining numbers can be computed by using the formulae

$$D(n+1)=D(n)+D(n-1)$$

Where, D is the element in the Fibonacci series.

An example of Fibonacci sequence of 10 element is given below as reference:

0H (0), 01H (1), 01H (1), 02H (2), 03H (3), 05(5), 08H (8), 0DH (13), 15H (21), 22H (34)

To generate the sequence, the length of the sequence is initiated as count. Than the first two elements were initialised and stored in desired memory location. The count is updated by decrementing it by two. Then, previous two elements are used to compute the next element. The computed value is stored and the count value is updated by reducing it by 1. The computation is repeated till all the element has been computed and stored. The count tells the number of elements remaining to be computed. So, when count becomes zero, Zero flag is set and the control goes out of the loop.

Flow Chart:

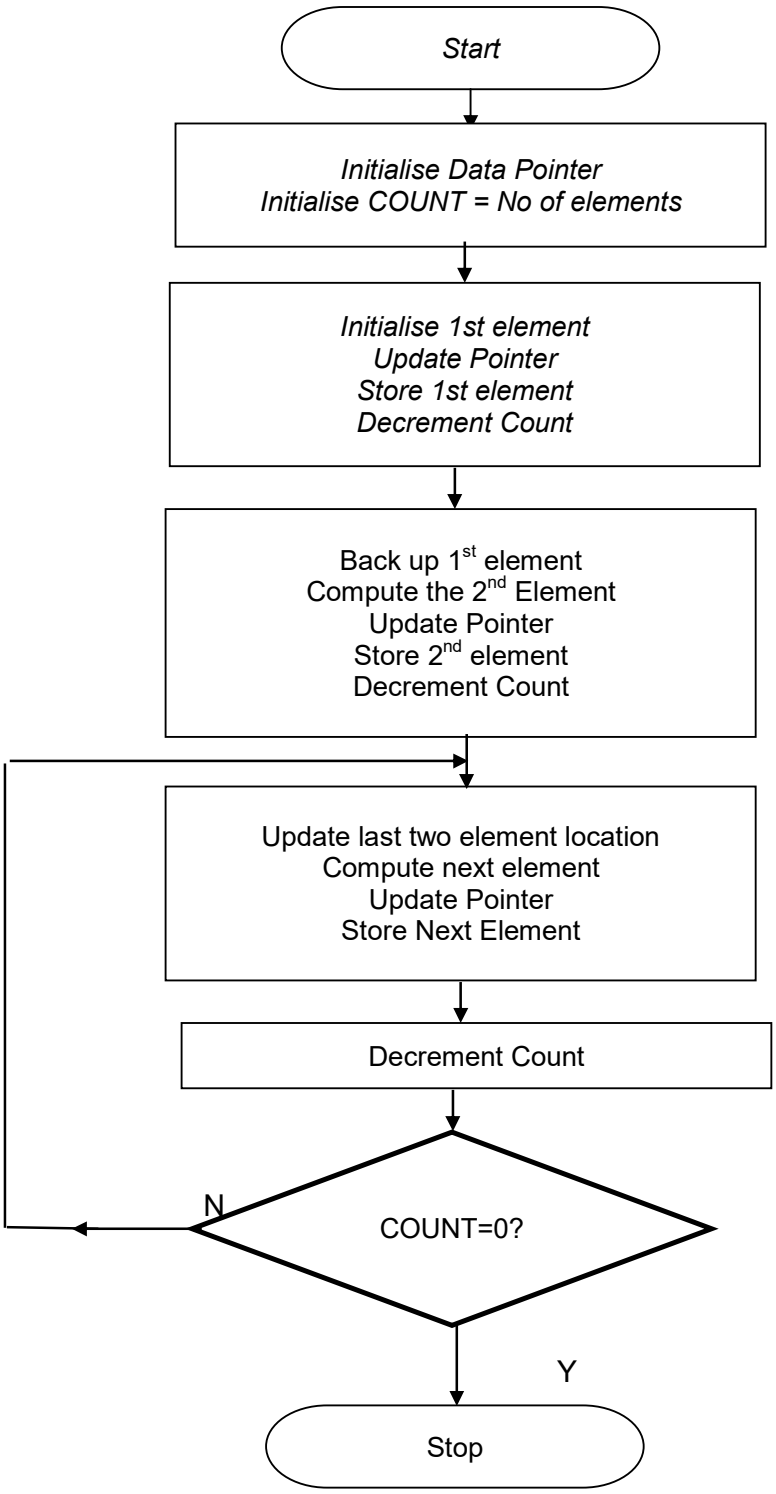


Fig: Generation of Fibonacci Series

Program:

Generation of Fibonacci series of desired length.

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 20		LXI H, 2000H	; Initialise DATA pointer
4203	4E		MOV C, M	; COUNT = Length of sequence
4204	AF		XRA A	; Clear Accumulator
4205	23		INX H	; Update pointer
4206	77		MOV M, A	; Store 1st element '0'
4207	0D		DCR C	; Update Count
4208	47		MOV B, A	; Update ith element in B
4209	3C		INR A	; 2nd element of the sequence
420A	23		INX H	; Update Pointer
420B	77		MOV M, A	; Store 2nd element '1'
420C	0D		DCR C	; Update Count
420D	50	LOOP:	MOV D, B	; Update (n-1)th element in D
420E	47		MOV B, A	; Update (n)th element in register B
420F	82		ADD D	; (n+1)th element computed
4210	23		INX H	; Points to next memory location
4211	77		MOV M, A	; Store next element of the sequence
4212	0D		DCR C	; Update Counter and find
4213	C2 0D 042		JNZ LOOP	; Repeat loop to compute all the numbers
4216	76		HLT	; Stop execution

Testing of the program:

- Program was assembled.
- The length of Fibonacci series is initialized and program was executed.
- Verified the result with different length.

Observation:

Sl.	Length of the sequence (2000H)	Fibonacci Sequence 2001H onwards
1	5	
2	10	2001H-200AH: 0H(0), 01H(1), 01H(1), 02H(2), 03H(3), 05(5), 08H(8), 0DH(13), 15H(21), 22H(34)
3.	14	

Conclusion:

Learn to handle and create a new series using loops. The Sequence was created for different length. However it was limited to 14, so that the number can be represented with one byte only.

Signature & Roll No. of the student

Aim: Multiplication oftwo8-bit numbers.(Bit rotation or Shift and add method)

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H : Multiplicand
2001H : Multiplier

2006H : Lower byte of product (LB)
2007H : Upper byte of product (UB)

Theory:

There are different algorithms to multiply two numbers. The simplest way to implement is to add the multiplicand repeatedly multiplier number of times. To multiply X with Y, X is added to "ZERO" Y number of times. To implement we initialise a 16-bit register as 0. Initialise the multiplier as count. Count is decremented by 1, when the multiplicand is added to the sum (partial product). So, when count becomes 0, multiplicand is multiplied by Multiplier times. The sum is the final product value. When this algorithm is used, the execution time is quite high when multiplier value is large. This method is known as **repeated addition method**.
Another method is the standard method used for multiplication in pen and paper. This method is known as **shift and adds method or bit rotation method**.

Multiplication of binary number is performed in the same way as multiplication of digital numbers. The multiplication of binary numbers is carried out by multiplying the multiplicand by one bit of the multiplier at a time. Each such multiplication forms a partial product. The result of the partial product for each bit is placed in such a manner that the LSB is under the corresponding multiplier bit. Finally, the partial products are added to get the complete product.

In the case of binary multiplication there are certain advantages. The multiplication is actually the addition of multiplicand with itself with some suitable shift. Thus, multiplication is actually a process of shifting and adding. In the case of binary multiplication, since the digits are 0 and 1, each step of the multiplication is simple. If the leftmost multiplier bit is 1, a copy of the multiplicand is shifted and added to partial product but if the multiplier bit is 0, no addition is required.

To implement the algorithm

1. Initialise a 16-bit product as 0. This is called partial product.
2. Shift the partial product to left. As there is no shift instruction for 16-bit number, shifting to left is obtained by adding the partial product to it. This is equivalent to shifting the multiplicand right once.
3. The multiplier is rotated to left, so that CY= MSB of the multiplicand.
4. If Cy =1 add the multiplicand to partial product to get the new partial product. If Cy=0, skip the addition.
5. Repeat step 2, 3 and 4 for 8-times for an 8-bit multiplicand.
6. Now the partial product is the actual product value.

Flow Chart:

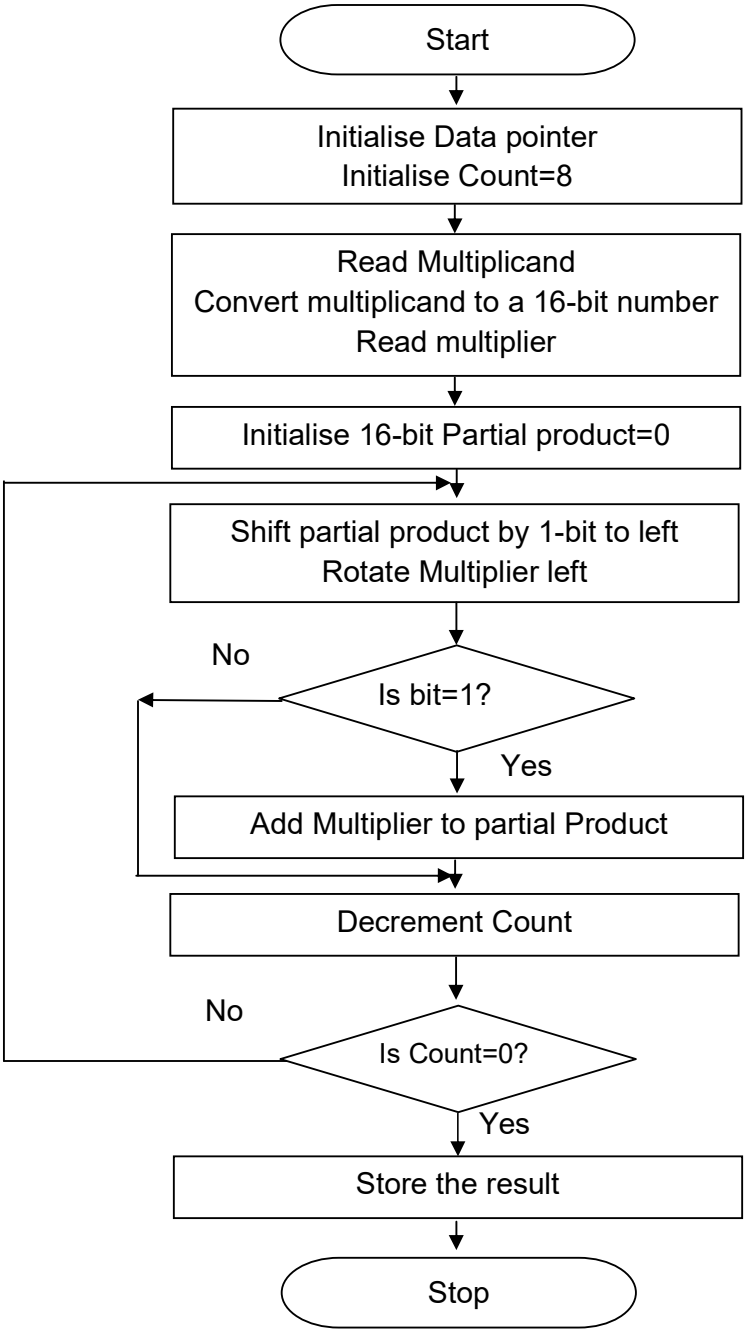


Fig: Multiplication of two 8-bit numbers

Note:

- HL holds the Partial Product
- HL is shifted by adding HL to it.
- DE holds the extended Multiplicand
- A holds the multiplier

Program:

Multiplication of two 8-bitnumbers, result may be in 16-bit.

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Initialise Pointer
4203	5E		MOV E, M	; Read Multiplicand
4204	16 00		MVI D, 00H	; UB of Dividend = 00H
4206	23		INX H	
4207	7E		MOV A, M	; Read Multiplier
4208	0E 08		MVI C, 08H	; Initialise count to shift 8 times
420A	21 00 00		LXI H, 00H	; Partial product=00H
420D	29	LOOP:	DAD H	; Shift partial product left by 1-bit
420E	17		RAL	; Rotate multiplicand and Check next Bit
420F	D2 13 42		JNC NXT	; Add if bit=0 else skip
4212	19		DAD D	; Add Multiplier to Partial product
4213	0D	NXT:	DCR C	; Update counter
4214	C2 0D 042		JNZ LOOP	; loop 8 times
4217	22 06 020		SHLD 2006H	; Store Result
421A	76		HLT	

Testing of the program:

Multiplicand and Multiplier are stored in memory location
Program is assembled and executed.
Result is verified for correctness with different sets of value.

Observation:

Sl.	Multiplicand (2000H)	Multiplier (2001H)	Product (UB) (2007H)	Product (LB) (2006H)	Remarks
1.					
2.					
3.					
4.					
5.					

Conclusion:

The shift and add method of multiplication is implemented. We learn how to shift a number to left by adding the number to itself i.e., multiplying by two.

Signature & Roll No. of the student

Program for repeated addition Method;Multiplication by repeated addition Method

```
;Assumption:
;       2000H: Multiplicand
;       2001H : Multiplier
;       2006H: Product (LSB)
;       2007H: Product (MSB)
; Register B holds the Product(MSB) and and Accumulator Product (LSB)
; Muliplier is added repeatedly Multiplicand Times
; Register C is used as counter

        LXI H, 2001H        ; Initialise Data pointer
        MVI B, 00H                  ; Initialse Product(MSB)=0
        MOV C, M            ; Initialise Counter with Multiplicand

        MOV A,B             ; Partial Product=00H
        CMP M
        JZ RSLT             ; Skip Computation if Multiplicand=0
        DCX H

LOOP:   ADD M               ; Add repeatedly
        JNC NXT             ; Skip updationod MSB if no carry generated
        INR B               ; Update MSB
NXT:    DCR C               ; Decrement Count
        JNZ LOOP            ; Continue adding if count is not equal to 0
RSLT:   STA 2006H           ; Store the LSB of producrt in desird location
        MOV A,B
        STA 2007h           ; Store MSB of product in desired location
        HLT                 ; Stop execution
```

Aim: Division of an8-bit number with an 8-bit number. (Repeated subtraction method)

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H : Dividend

2001H : Divisor

2006H : Quotient

2007H : Remainder

Theory:

Division can be performed by subtracting the divisor repeatedly from the dividend. The value of quotient is incremented by 1 after the subtraction. This process continues till theremainder is less than the divisor.

The division may also be performed as we divide as we do it in pen and paper. In this process the dividend is made 16-bit by appending 8-zeros to the left. The 16-bit dividend is shifted by one bit to the left. The 8-bit divisor is subtracted from most 8 significant bits of shifted dividend. If there is no borrow the least significant bit of Quotient is set to 1 else set to 0. To line up the subtraction of dividend it is shifted again to left by 1 bit. This process continues for 8-times. The remaining part of dividend is the remainder.

Though the second method is faster we will implement the 1stmethodi.e. repeated subtraction method.

Note:

The divisor is compared with the dividend. If the divisor is greater, Quotient=0 and remainder= dividend. So, the division procedure is aborted and resultsare stored.If dividend is greater than divisor, it is subtracted and quotient is incremented by 1. This process continues till the remainder becomes less than divisor. The results are stored in desired locations.

Flow Chart:

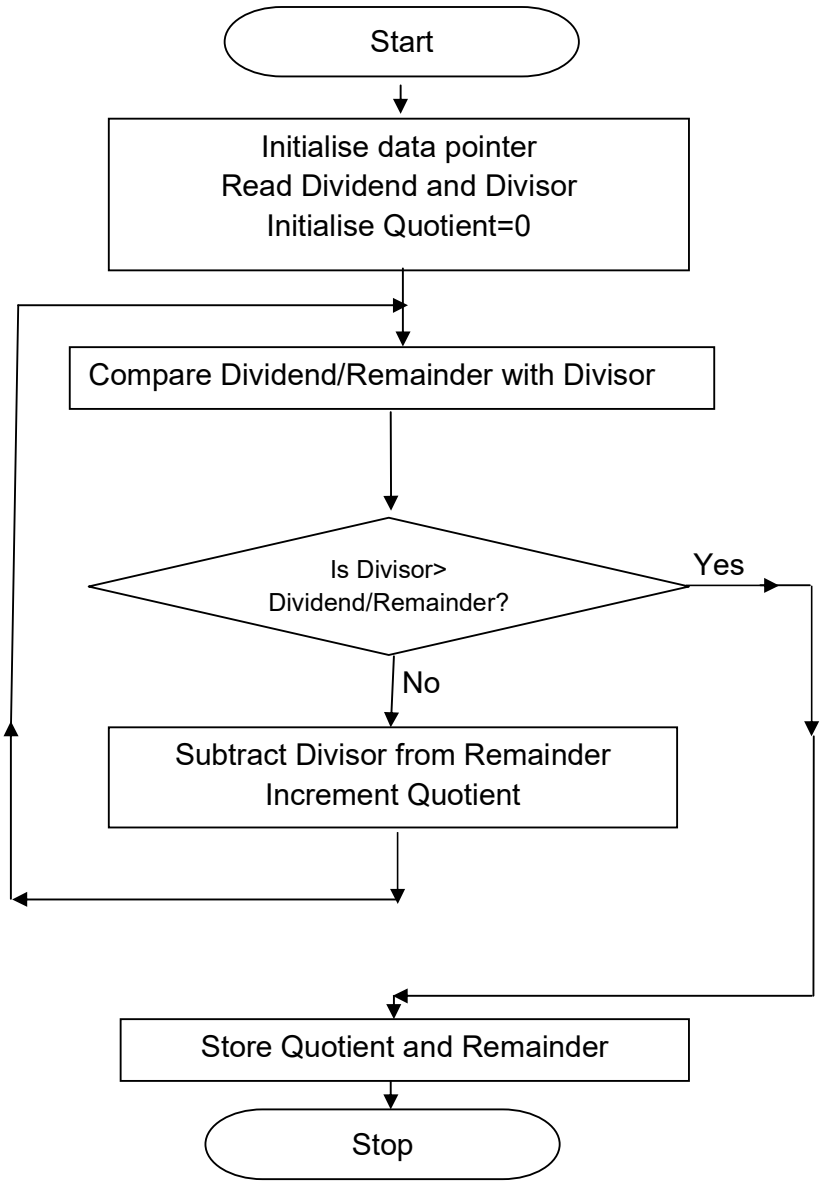


Fig: Division of an 8-bit number

Note:

- HL : Data pointer
- A : Dividend/ Remainder
- B holds the divisor
- D=Quotient

Program:

Division of an8-bit number with an 8-bit number

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Data pointer initialisation
4203	7E		MOV A, M	; Read Dividend (8-bit)
4204	23		INX H	
4205	46		MOV B, M	; Read Divisor (8-bit)
4206	0E 00		MVI C, 00H	; Initialise Quotient=0
4208	B8	RPT:	CMP B	; Compare the divisor with the remainder
4209	DA 11 42		JC RSLT	; Go to result store
420C	90		SUB B	; Subtract divisor from remainder
420D	0C		INR C	; Update Quotient
420E	C3 08 042		JMP RPT	; Repeat the sequence
4211	32 07 020	RSLT:	STA 2007H	; Store Remainder
4214	79		MOV A,C	; Quotient to Accumulator
4215	32 06 020		STA 2006H	; Store Quotient
4218	76		HLT	

Testing of the program:

- Dividend anddivisor are stored in memory location
- Program is assembled and executed.
- Result is verified for correctness with different sets of value.

Observation:

Sl.	Dividend (2000H)	Divisor (2001H)	Quotient (2006H)	Remainder (2007H)	Remarks
1.					
2.					
3.					
4.					
5.					

Conclusion:

The program was tested with various types of data and results are found to be correct.

Signature & Roll No. of the student

Aim:

Write an 80850 ALP .to determine the factorial of an8-bit number.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H --> DATA (Factorial of the number is determined)

2001H --> LSB of Result (Factorial Value)

2002H --> MSB of Result (Factorial Value)

7FFFH down word --> Stack Locations

The largest number for an 8-bit result is 5 where as the highest number for a 16-bit result is 09. So, the number must be less than or equal to 08.

NOTE:

---------- Factorial of different numbers for Reference----------

2!= 02H	(0, 02)	02
3!= 06H	(0, 06)	06
4!= 18H	(0, 24)	24
5!= 78H	(0, 120)	120
6!= 2 D0H	(2,208)	720 > 256 > (highest 8-bit Number)
7!= 13 B0H	(19,176)	5,040
8!= 9D 80H	(157,128)	40,320 (Highest 16-bit number)

Theory:

The factorial of a number $'n'$ is defines as

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \dots\dots\dots\dots (n-2) \times (n-1) \times n$$
$$= n \times (n-1) \times (n-2) \dots\dots\dots\dots 3 \times 2 \times 1$$

So, to compute the factorial partial product is initialized with 1 (a 16-bit number).The partial product is multiplied with $'n'$. The product is the new partial product. The number is decremented to find the next factor. The partial product is multiplied with the new factor till the new factor becomes 0 or 1. As it is easy to test new factor for zero, When the loop is terminated when the new factor reduces to $'0'$. The last partial product is the computed factorial of the number.

Flow Chart:

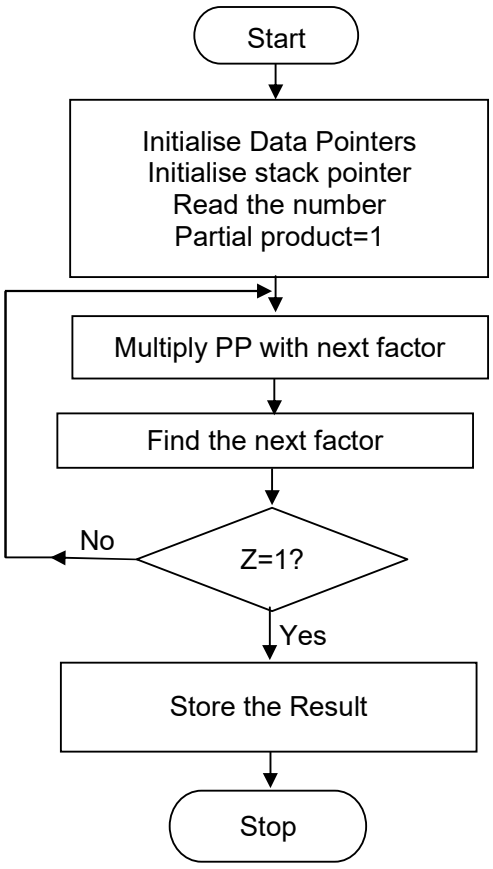


Fig (a)

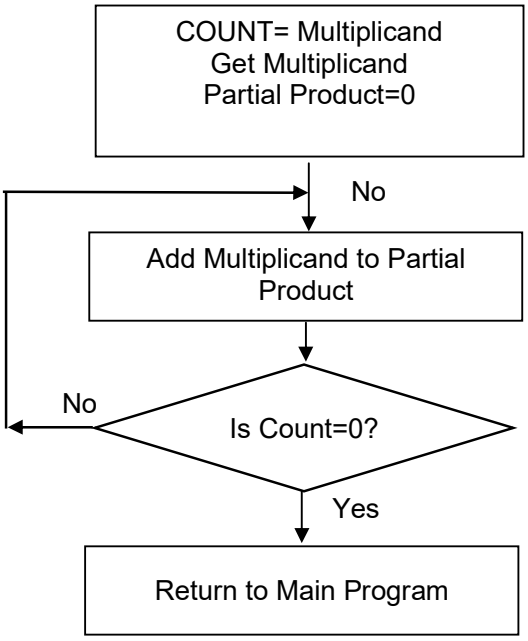


Fig (b)

Fig: Flow chart to compute factorial. (a) Main program (b)Multiplication Sub-routine using repeated addition method.

Program:

Computation of factorial(16-bit)

Address	opCode/ Operand	Label	Mnemonics	Comment
4200H	31 00 080		LXI SP,8000H	; Define stack memory
4203H	21 00 020		LXI H, 2000H	; Initialise Data pointer
4206H	46		MOV B,M	; Read data
4207H	21 01 00		LXI H, 0001H	; Partial Product=01
420AH	CD 15 42	FACT:	CALL MUL	; PP=PP x Factor
420DH	05		DCR B	; Find Next factor
420EH	C2 0A 042		JNZ FACT	; If new factor-0, stop multiplication
4211H	22 01 020		SHLD 2001H	; Store the result
4214H	76		HLT	; Stop Execution.
; -------- Multiplication Subroutine --------				
; Multiplication Subroutine (Result may 16-bit) ; Modify Register HL, DE, C ;Register C holds the multiplier				

;HL holds the partial product				
;Product value returns in DE				
4215H	48	MUL:	MOV C,B	; COUNT= Multiplier
4216H	EB		XCHG	; Multiplicand to DE
4217H	21 00 00		LXI H, 0000H	; Partial product =0
421AH	19	RPT:	DAD D	; Repeatedly ADD to get product
421BH	0D		DCR C	; Update COUNT
421CH	C2 1A 42		JNZ RPT	; Repeatedly add Multiplicand Times
421FH	C9		RET	; Return to main Program

Testing of the program:

- The number whose factor has to be computed is stored in memory.
- Program was assembled and executed.
- Verified the 16-bit result from memory.
- The observations were repeated for different number.

Observation:

Sl.	Number $'n'$	Factorial	
	(2000H)	Upper Byte (2007H)	Lower Byte (2006H)
1	3		
2	5		
3.	8		

Conclusion:

Factorials of different number less than 09 werecalculated and verified. To compute the factorial of a larger number the multiplication subroutine needs to be modified to accommodate all bots of the product. As multiplicand is small (≤ 8), repeated addition algorithm has been used.

In this program we learn how a use a subroutine and call it from the main program

New Instruction used:

CALL Label: The call instruction **transfers the program sequence to the memory address given in the operand**. Before transferring, the address of the next instruction (after CALL) is pushed onto the stack. The content of SP is decremented by 2 to point to the top of the stack.

RET:RET is the instruction used at the end of sub-routine. Value of PC (Program Counter) is retrieved from the memory stack and value of SP (Stack Pointer) is incremented by 2.After execution of this instruction program control is transferred back to main program from where it had stopped.

Signature & Roll No. of the student

Aim:

Write an 80850 ALP .to find the largest number in an array.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H -->Length of the array (No of Elements in the array)

2001H onwards-->Data array

2100H: Largest number

Assumes the array length less than 255, else destination address to be modified.

Theory:

The first element of the array is read to the accumulator. Then the number in accumulator is compared with next element of the array in memory. If the number in larger than accumulator , the larger number is read to accumulator then compared with the next element. . If the number in accumulator is larger only comparison with the next element of array is carried out.

When an element is read or compared the count is decremented by 1. When the count=0, the process is stopped and the data in accumulator is the largest number that may be stored in desired location.

Flow Chart:

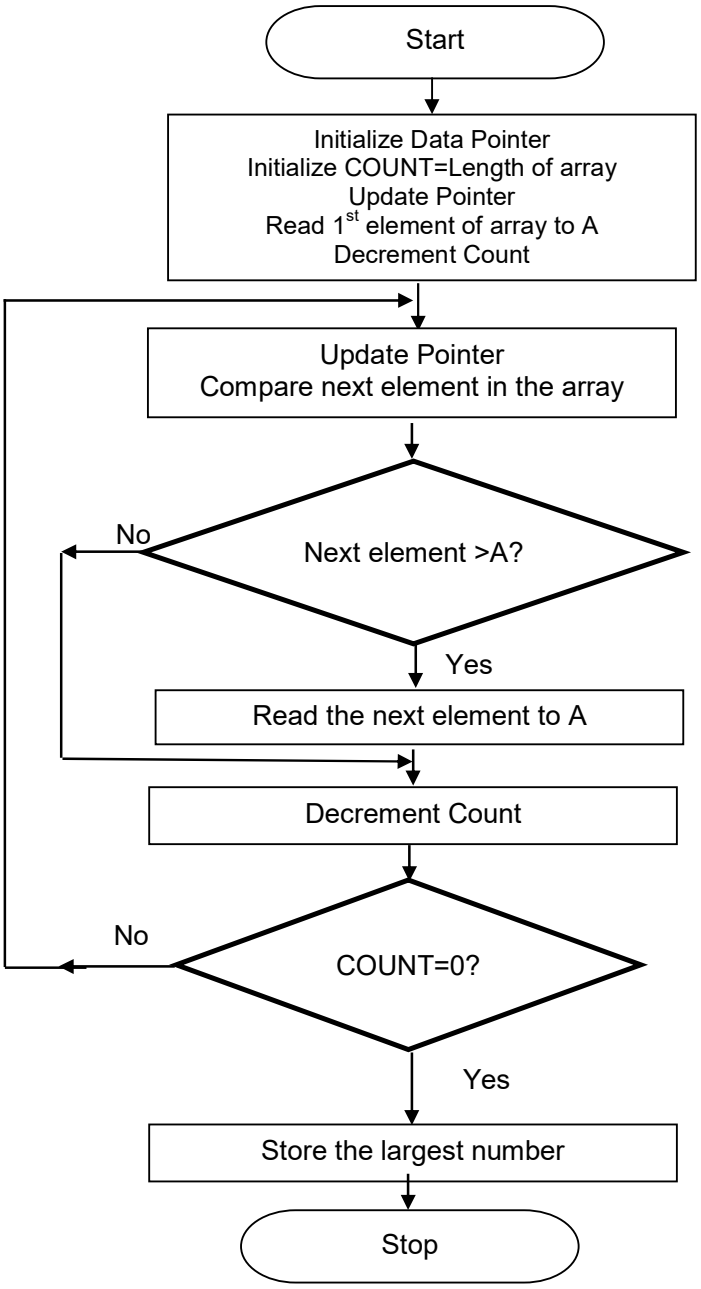


Fig: Flow chart to find the largest number in an array .

Program:

Finding the largest number in an array.

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Initialise Data Pointer
4203	4E		MOV C, M	; COUNT = Length of the array
4204	23		INX H	; Point to the 1st element of the array
4205	7E		MOV A, M	; Read the 1st element
4206	0D		DCR C	; Update COUNT

4207	23	loop:	INX H	; Point to next byte to be compared
4208	BE		CMP M	; CY=1 if Memory is greater, CY=0 if small or equal
4209	D2 0D 042		JNC nxt	; Skip if content of memory is smaller
420C	7E		MOV A, M	; Read the Large number in memory to A
420D	0D	nxt:	DCR C	; Update COUNT
420E	C2 07 042		JNZ loop	; Continue comparision till all elements are compared
4211	32 00 021		STA 2100H	; Store Largest Number
4214	76		HLT	

Testing of the program:

- The data of the array and its length are stored in desired memory location..
- Program was assembled and executed.
- Verified the result
- The observations were repeated for array of different length.

Observation:

SL	Data	Largest Number
1	Length of the array (2000H): Data element in array (2001H to ****H):	
2	Length of the array (2000H): Data element in array (2001H to ****H):	
3	Length of the array (2000H): Data element in array (2001H to ****H):	

Conclusion:

The experiment was carried out for different length of array and found tb working well. Here we learn to use loop to procees elements in an array.

New Instruction used:

CMP r/M:

Compare register and memory and flags are updated. The number to be compared are subtracted from the accumulator and the flags are updated but the content of acccumuator remain unchanged (result is not stored.) By testing Carry and zero flag it is possible to know which number is larger.

Signature & Roll No. of the student

Aim:

Write an 80850 ALP .to find the smallest number in an array.

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H -->Length of the array (No of Elements in the array)
2001H onwards-->Data array

2100H→Smallest number

Assumes the array length less than 255, else destination address to be modified.

Theory:

The first element of the array is read to the accumulator. Then the number in accumulator is compared with next element of the array in memory. If the number issmaller than accumulator, the small number is read to accumulator then compared with the next element. . If the number in accumulator is smaller only comparison with the next element of array is carried out.

When an element is read or compared the count is decremented by 1. When the count=0, the process is stopped and the data in accumulator is the smallest number that may be stored in desired location.

Flow Chart:

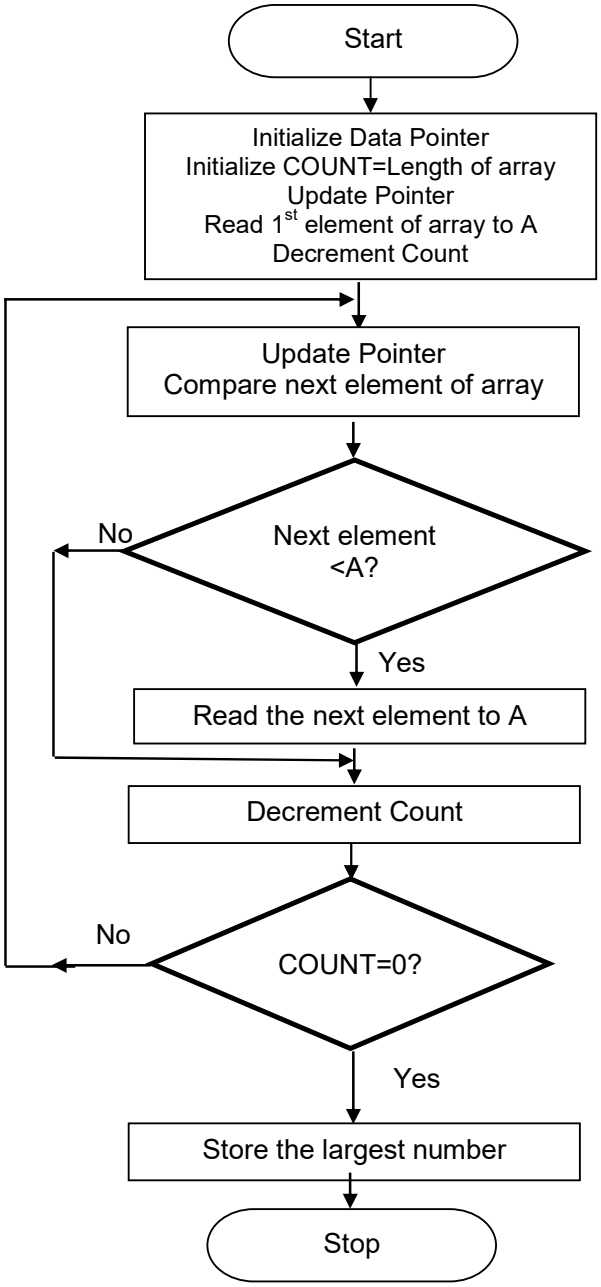


Fig: Flow chart to find the smallest number in an array

Program:

Finding the smallest number in an array.

Address	opCode/ Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Initialise Data Pointer
4203	4E		MOV C, M	; COUNT = Length of the array
4204	23		INX H	; Point to the 1st element of the array

4205	7E		MOV A, M	; Read the 1st element
4206	0D		DCR C	; Update COUNT
4207	23	loop:	INX H	; Point to next byte to be compared
4208	BE		CMP M	; CY=1 if Memory is greater, CY=0 if small or equal
4209	D2 0D 042		JC nxt	; Skip if content of memory is Larger
420C	7E		MOV A, M	; Read the Small number in memory to A
420D	0D	nxt:	DCR C	; Update COUNT
420E	C2 07 042		JNZ loop	; Continue comparison till all elements are compared
4211	32 00 021		STA 2100H	; Store Smallest Number
4214	76		HLT	

Testing of the program:

- The data of the array and its length are stored in desired memory location..
- Program was assembled and executed.
- Verified the result
- The observations were repeated for array of different length.

Observation:

SL	Data	Smallest Number
1	Length of the array (2000H): Data element in array (2001H to ****H):	
2	Length of the array (2000H): Data element in array (2001H to ****H):	
3	Length of the array (2000H): Data element in array (2001H to ****H):	

Conclusion:

The experiment was carried out for different length of array and found to working well. Here we learn to use loop to process elements in an array.

Signature & Roll No. of the student

Aim:

Write an 8085 ALP .to sort an array in ascending order

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H --> Length of the array
2001H onwards --> Data (8-bit) in the array

2100H : Length of the sorted array
2101 onwards : Sorted array data elements

Here it is assumed that the array size is less than 255.

Theory:

An array can be sorted in different way. In one method, . a new array is created and elements are inserted one by one. The elements are obtained by repeatedly finding the smallest element in the unsorted one. The element is inserted in the sorted array as the next element and the smallest element in unsorted array is replaced with the highest number (for 8-bit FFH). So, the algorithm is destructive in nature i.e.; the elements in the array to be sorted will be overwritten by the highest number. If the original array has to retain unmodified, the array may be copied to another location and then modified.

Steps of algorithm:

1. Find the smallest number of the array.
2. Store the smallest number as next data element in a new array.So, the array size goes on increasing.
3. Replace the smallest numberwith FFH (highest 8-bit number)
4. Repeat step 1 to 3 till new array length is same with the source array.
5. New array is in sorted form.

So, here two subroutines are used, one finds the smallest number and other find the location of smallest number in the unsorted array and replace it with FFH.

Flow Chart:

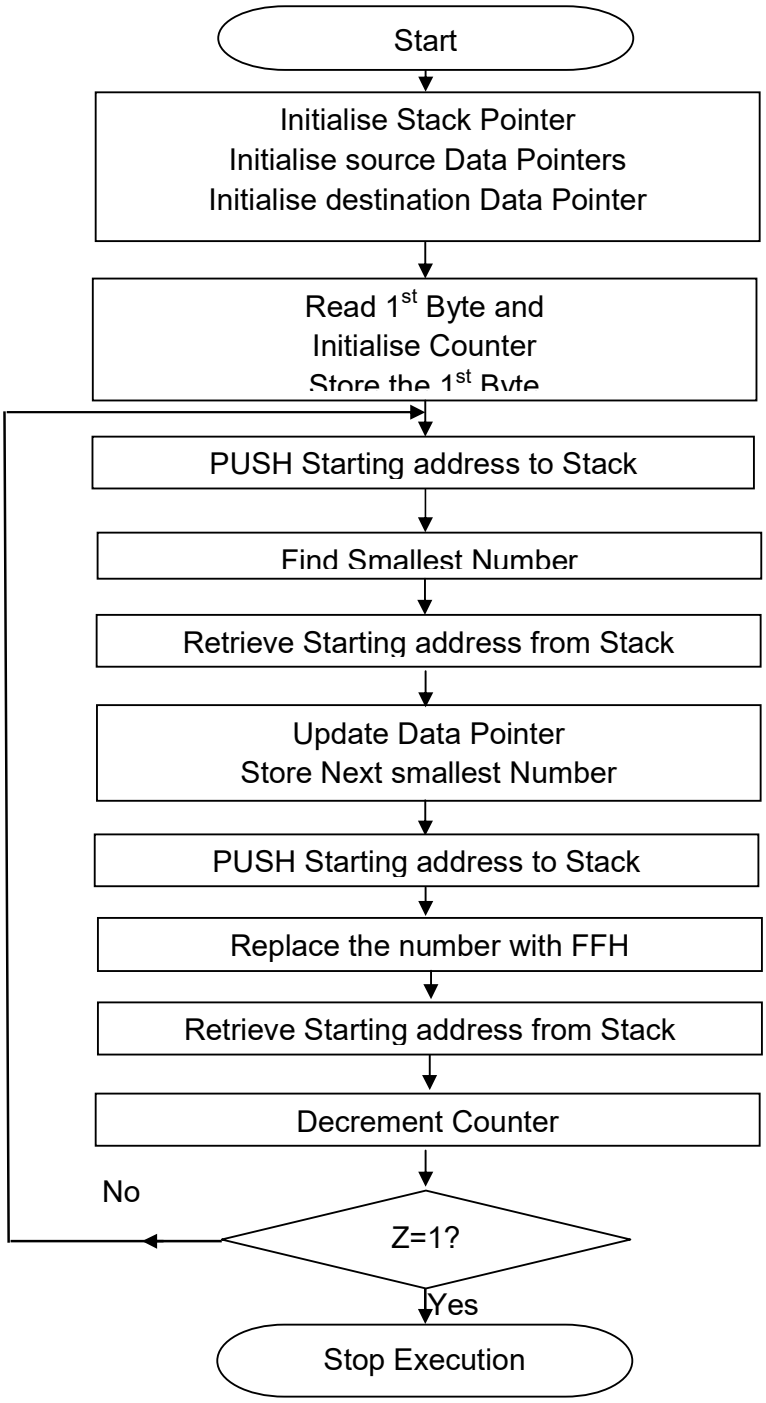


Fig: Main Program: sorting in ascending order

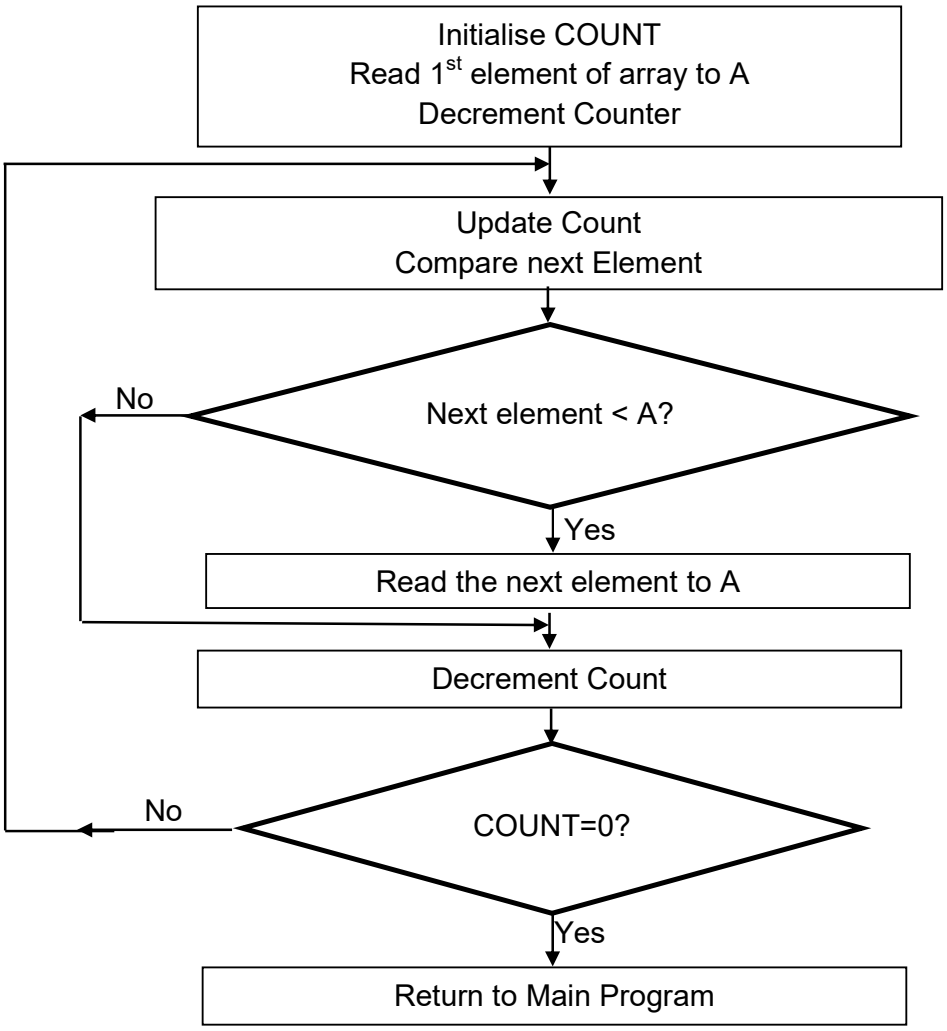


Fig: Flowchart for sub-routine to find the smallest number

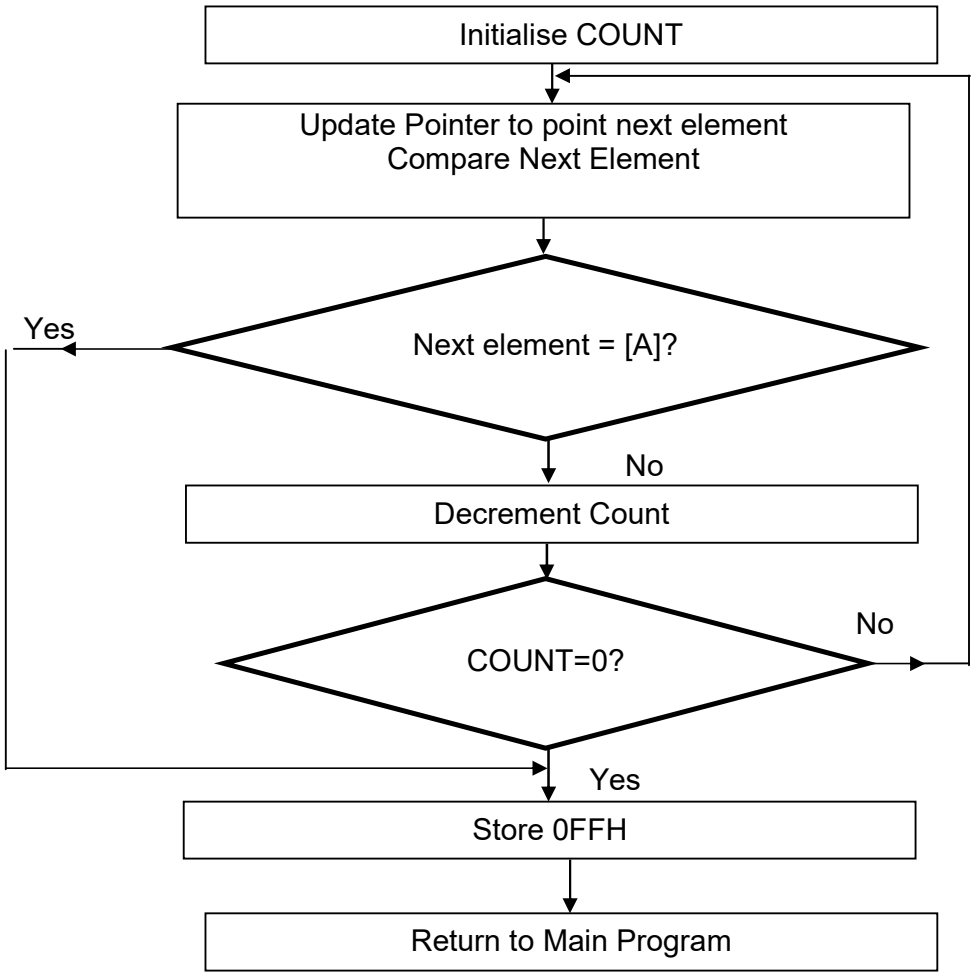


Fig: Flow chat of subroutine that replace the smallest number with FFH

Program:

Sorting of array in ascending order

Address	opCode/ Operand	Label	Mnemonics	Comment
; Main Program ; Register B is used as counter in Main Program as register C is reserved as counter for sub-routine ; Main program intialises data pointers, stack pointers counter, call subroutines and store the data in the new sorted array.				
4200	31 00 080		LXI SP, 8000H	; Define Stack
4203	21 00 020		LXI H, 2000H	; Source Data Location
4206	11 20 20		LXI D, 2100H	; Destination Data Location

4209	7E		MOV A, M	; Length of array to Accumulator
420A	47		MOV B, A	; Count= Length of array
420B	12		STAX D	; Store array length
420C	E5	start:	PUSH H	; Store starting address in Stack
420D	CD 1D 42		CALL SM_NO	; Call subroutine that finds the smallest number in the array/ modified array.
4210	E1		POP H	; Retrieve starting address from stack
4211	13		INX D	; Update destination pointer
4212	12		STAX D	; Store the next smallest number
4213	E5		PUSH H	; Store Starting address in Stack
4214	CD 2C 42		CALL REPL	; Call data replacement subroutine
4217	E1		POP H	; Retrieve Starting address from Stack
4218	05		DCR B	; Update count
4219	C2 0C 042		JNZ start	; repeat if all element of array is not sorted
421C	76		HLT	; Stop Execution
; Subroutine to find the smallest number ; Modifies HL, C and A ; Smallest number is returned in register A				
421D	4E	SM_NO:	MOV C, M	; COUNT= Length of the array
421E	23		INX H	
421F	7E		MOV A, M	; Read 1st element of array
4220	0D		DCR C	; Update COUNT
4221	23	loop:	INX H	; Point to next element of array
4222	BE		CMP M	; Compare next number
4223	DA 27 42		JC ahd	; Skip if the number in memory is larger
4226	7E		MOV A, M	; Read the smaller number to Accumulator
4227	0D	ahd:	DCR C	; Update COUNT
4228	C2 21 42		JNZ loop	; Continue if all elements are not compared
422B	C9		RET	; Return to main Program
; Replace the smallest number with FFH (1-Byte Largest Number) ; Searches the location and replacethe 1st match with FF				

; Modifies HL, C and A ; The smallest number to be replace must be in A				
422C	4E	REPL:	MOV C, M	; COUNT = Length of array
422D	23	loop1:	INX H	
422E	BE		CMP M	; Compare Next element
422F	CA 36 42		JZ next	; Abort loop when data location is found
4232	0D		DCR C	; Update count
4233	C2 2D 42		JNZ loop1	; Continue comparing with next element
4236	36 FF	next:	MVI M, 0FFH	; Replace the element with 0FFH
4238	C9		RET	; Return to main program

Testing of the program:

- Create the array manually.
  - The array to be is stored in desired memory locations.
- Assemble the program and rectify syntax error, if any.
- Execute the program.
- Verified the new array whether sorted or not.If any error
- is observed, debug and modify the program for its logical error.
- The observations were repeated for different numbers.

Observation:

SL.	Array (to be sorted)	Array (Sorted)	Remarks
1	2000H: Length of array (x) ---- 2001H: 200xH: ----, ----,-----	2000H: Length of array (x) ---- 2001H: 200xH: ----, ----,-----	
2			
3			
4			

Conclusion:

The program was tested with data array of different length and fount working satisfactorily. The program can be extended for a 16-bit number, where upper byte will be compared for larger or smaller. If MSB is equal, LSB is tested to determine smallest number.

New Instruction used:

CALL Label: The call instruction **transfers the program sequence to the memory address given in the operand**. Before transferring, the address of the next instruction (after CALL) is pushed onto the stack. The content of SP is decremented by 2 to point to the top of the stack.

RET: RET is the instruction used at the end of sub-routine. Value of PC (Program Counter) is retrieved from the memory stack and value of SP (Stack Pointer) is incremented by 2.After execution of this instruction program control is transferred back to main program from where it had stopped.

Signature & Roll No. of the student

Aim:

Write an 8085 ALP .to sort an array in descending order

Equipment and software used:

1. Personal Computer
2. 8085-simulator

Assumptions:

2000H --> Length of the array
2001H onwards --> Data (8-bit) in the array

2100H : Length of the sorted array
2101 onwards : Sorted array data elements

Here it is assumed that the array size is less than 255.

Theory:

An array can be sorted in different way. In one method, . a new array is created and elements are inserted at the end of the array. The elements to be added are obtained by repeatedly finding the smallest element in the unsorted(modified)array. The element is inserted in the sorted array as the next element and the largestelement in unsorted array is replaced with the smallestpossible 8-bit number (for 8-bit 00H). So, the algorithm is destructive in nature i.e.; the elements in the array to be sorted will be overwrittenwith 0. If the original array has to retain unmodified, the array may be copied to another location and then modified.

Steps of algorithm:

1. Find the largest number of the array.
2. Store the largest number as next data element in a new array.So, the array size goes on increasing.
3. Replace the largest numberwith “0”(smallest8-bit number)
4. Repeat step 1 to 3 till new array length is same with the source array.
5. New array is in sorted form.

So, here two subroutines are used, one finds the smallest number and other find the location of smallest number in the unsorted array and replace it with “0”.

Flow Chart:

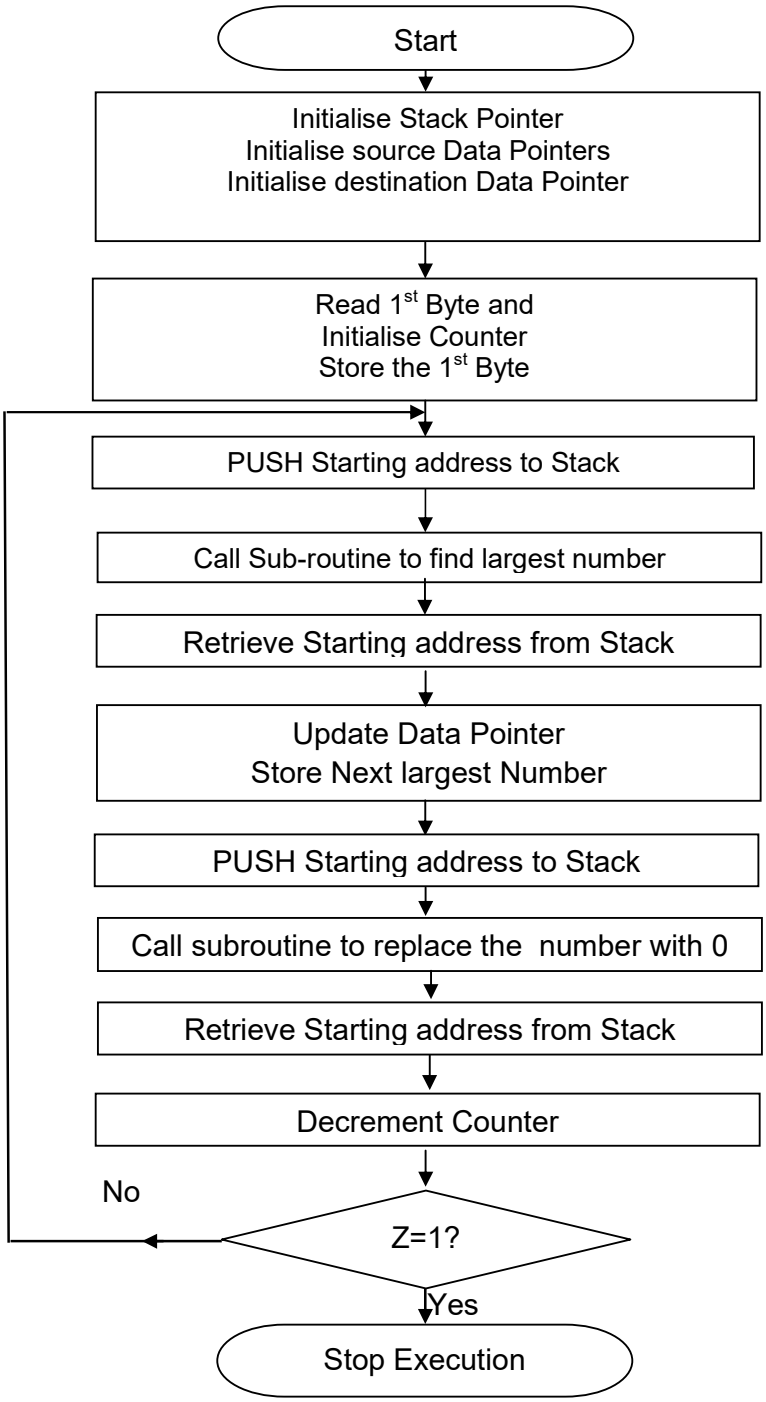


Fig: Main Program: sorting in descending order

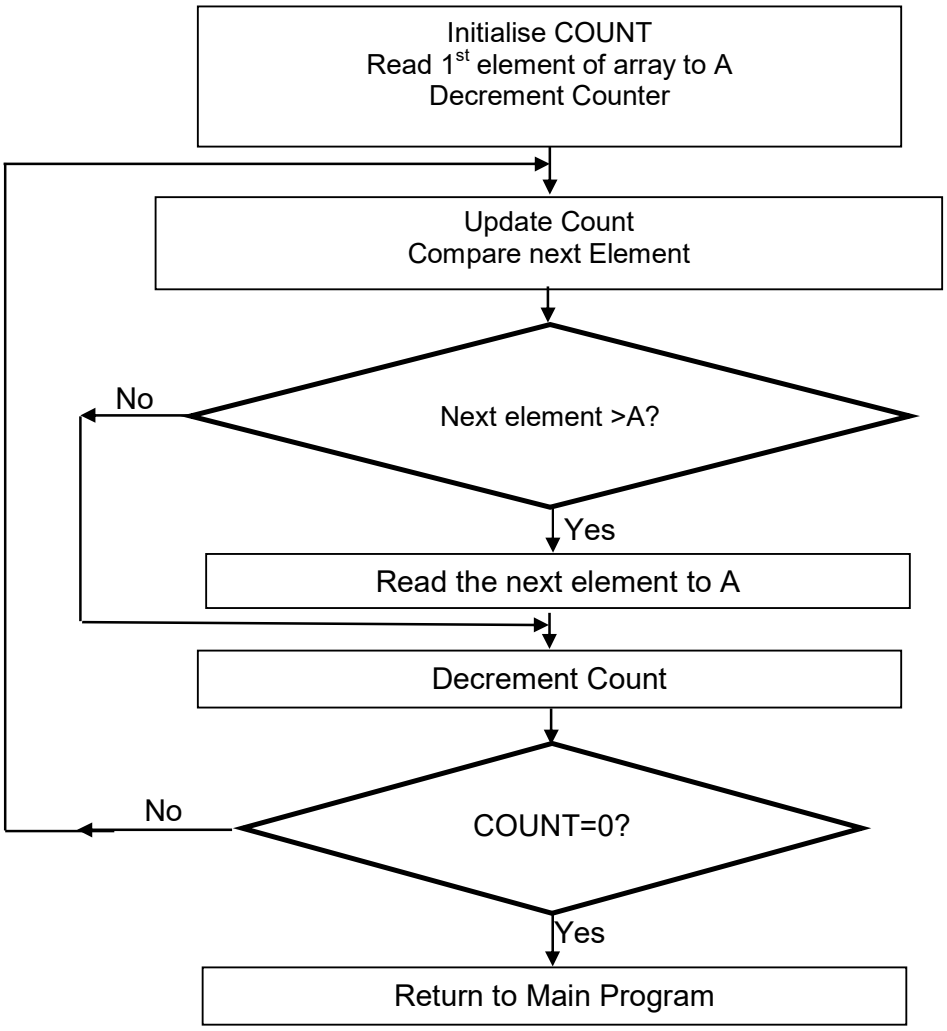


Fig: Flowchart for sub-routine to find the largest number

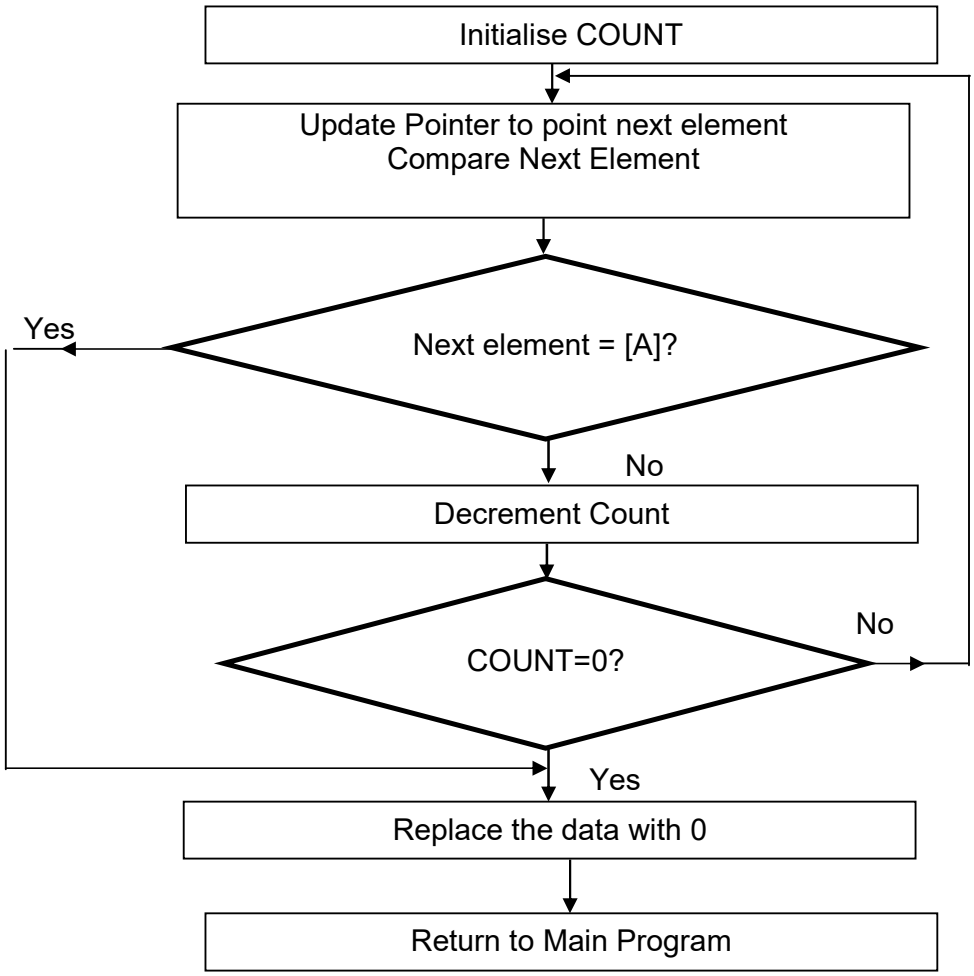


Fig: Flow chat of subroutine that replace the largest number with 00H

Program:

Sorting of array in descending order.

Address	opCode/ Operand	Label	Mnemonics	Comment
; Main Program ; Register B is used as counter in Main Program as register C is reserved as counter for sub-routine ; Main program intialises data pointers, stack pointers counter, call subroutines and store the data in the new sorted array.				
4200	31 00 080		LXI SP, 8000H	; Define Stack
4203	21 00 020		LXI H, 2000H	; Source Data Location
4206	11 20 20		LXI D, 2100H	; Destination Data Location
4209	7E		MOV A, M	; Length of array to Accumulator

420A	47		MOV B, A	; Count= Length of array
420B	12		STAX D	; Store array length
420C	E5	start:	PUSH H	; Store starting address in Stack
420D	CD 1D 42		CALL LR_NO	; Call subroutine that finds the smallest number in the array/ modified array.
4210	E1		POP H	; Retrieve starting address from stack
4211	13		INX D	; Update destination pointer
4212	12		STAX D	; Store the next smallest number
4213	E5		PUSH H	; Store Starting address in Stack
4214	CD 2C 42		CALL REPL	; Call data replacement subroutine
4217	E1		POP H	; Retrieve Starting address from Stack
4218	05		DCR B	; Update count
4219	C2 0C 042		JNZ start	; repeat if all element of array is not sorted
421C	76		HLT	; Stop Execution
; Subroutine to find the largest number ; Modifies HL, C and A ; Smallest number is returned in register A				
421D	4E	LR_NO:	MOV C, M	; COUNT= Length of the array
421E	23		INX H	
421F	7E		MOV A, M	; Read 1st element of array
4220	0D		DCR C	; Update COUNT
4221	23	loop:	INX H	; Point to next element of array
4222	BE		CMP M	; Compare next number
4223	DA 27 42		JNC ahd	; Skip if the number in memory is larger
4226	7E		MOV A, M	; Read the smaller number to Accumulator
4227	0D	ahd:	DCR C	; Update COUNT
4228	C2 21 42		JNZ loop	; Continue if all elements are not compared
422B	C9		RET	; Return to main Program
; Replace the smallest number with 0 (1-Byte smallest Number) ; Searches the location and replacethe 1st match with FF ; Modifies HL, C and A ; The smallest number to be replace must be in A				

422C	4E	REPL:	MOV C, M	; COUNT = Length of array
422D	23	loop1:	INX H	
422E	BE		CMP M	; Compare Next element
422F	CA 36 42		JZ next	; Abort loop when data location is found
4232	0D		DCR C	; Update count
4233	C2 2D 42		JNZ loop1	; Continue comparing with next element
4236	36 FF	next:	MVI M, 00H	; Replace the element with 0FFH
4238	C9		RET	; Return to main program

Testing of the program:

- Create the array manually.
    - The array to be is stored in desired memory locations.
- Assemble the program and rectify syntax error, if any.
- Execute the program.
- Verified the new array whether sorted or not.If any error
- is observed, debug and modify the program for its logical error.
- The observations were repeated for different numbers.

Observation:

SL.	Array (to be sorted)	Array (Sorted)	Remarks
1	2000H: Length of array (x) ---- 2001H: 200xH: ----, ----,-----	2000H: Length of array (x) ---- 2001H: 200xH: ----, ----,-----	
2			
3			
4			

Conclusion:

The program was tested with data array of different length and fount working satisfactorily. The program can be extended for a 16-bit number, where upper byte will be compared for larger or smaller. If MSB is equal, LSB is tested to determine smallest number.

New Instruction used:

CALL Label: The call instruction **transfers the program sequence to the memory address given in the operand**. Before transferring, the address of the next instruction (after CALL) is pushed onto the stack. The content of SP is decremented by 2 to point to the top of the stack.

RET: RET is the instruction used at the end of sub-routine. Value of PC (Program Counter) is retrieved from the memory stack and value of SP (Stack Pointer) is incremented by 2.After execution of this instruction program control is transferred back to main program from where it had stopped.

Signature & Roll No. of the student

Aim of the Experiment:

Finding the Square of a number using lookup Table

Equipment and software used:

1. Personal Computer
2. 8085-simulator (GNUSim8085)

Assumptions:

2000H : Data input
2006H : Square of a number

Theory:

A lookup table is an array of data that maps input values to output values, thereby approximating a mathematical function. Given the input value(s), a lookup operation retrieves the corresponding output values from a table. So, when a lookup table is used the output value is not actually computed but read from a reference table stored in ROM.

Flow Chart:

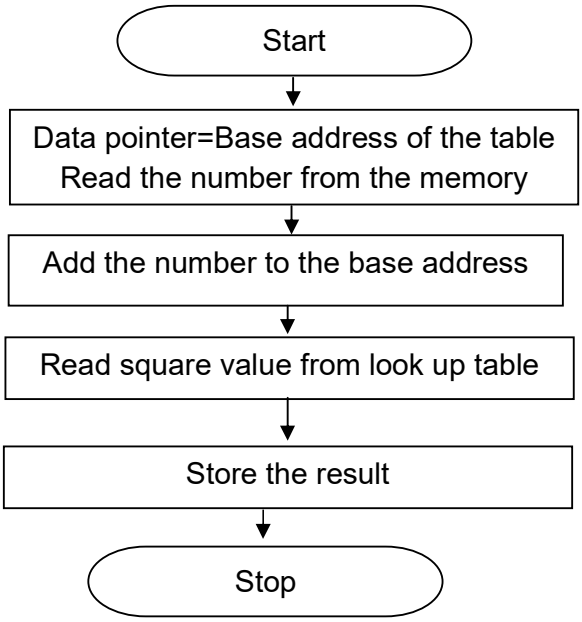


Fig: Finding square of a number using lookup table

Program:

Square of a number using lookup table

Address	Opcode/ Operand	Label	Mnemonics	Comment
4200	21 00 060		LXI H, 6000H	; Base of the look-up Table
4203	3A 00 020		LDA 2000H	; Read the number
4206	6F		MOV L, A	; Addition of base and number
4207 7E			MOV A, M	; Read the square value from look-up Table
4208	32 06 020		STA 2006H	; Store the result
420B	76	HLT		; Stop execution

Testing of the program:

- Program is written.
- Look table is manually created
- Data is stored in memory location
- Program is assembled and executed.
- Result is verified for correctness with different sets of value.

Observation:

Sl.	Data (2000H)	Square (2006H)	Remarks
1.			
2.			
3.			
4.			
5.			

Conclusion:

Learned how to use lookup table.

Signature & Roll No. of the student

Aim of the experiment:

Write an 8085 ALP to convert 8-bit Hexadecimal number(Binary) to BCD (Decimal).

Instrument/Software required

PC with 8085 Simulator (GNU 8085)

Assumptions

The following memory locations are used to store data and results:

2000H : 8-bit data

2006H : BCD Number (LB)

2007H : BCD Number (UB)

Theory

The maximum value of one byte HEX number is 255. This required two bytes of data to store the result.

In a simple form, the HEX number is decremented repeatedly till it becomes zero. The partial BCD sum is added with 1 for every decrement of HEX number. As the decrement perform is in Binary and Addition is BCD the result obtained is in BCD having the same magnitude .

Another algorithm may be used to convert the HEX to BCD. In this algorithm, the lower digit is converted to BCD by simply using DAA instruction. The upper digit is multiplied (multiplication using repeated Decimal addition). The lower converted result is added to product value to obtain the BCD equivalent of HEX number.

Here we will implement using the 1st algorithm.

Algorithm/ Flow Chart

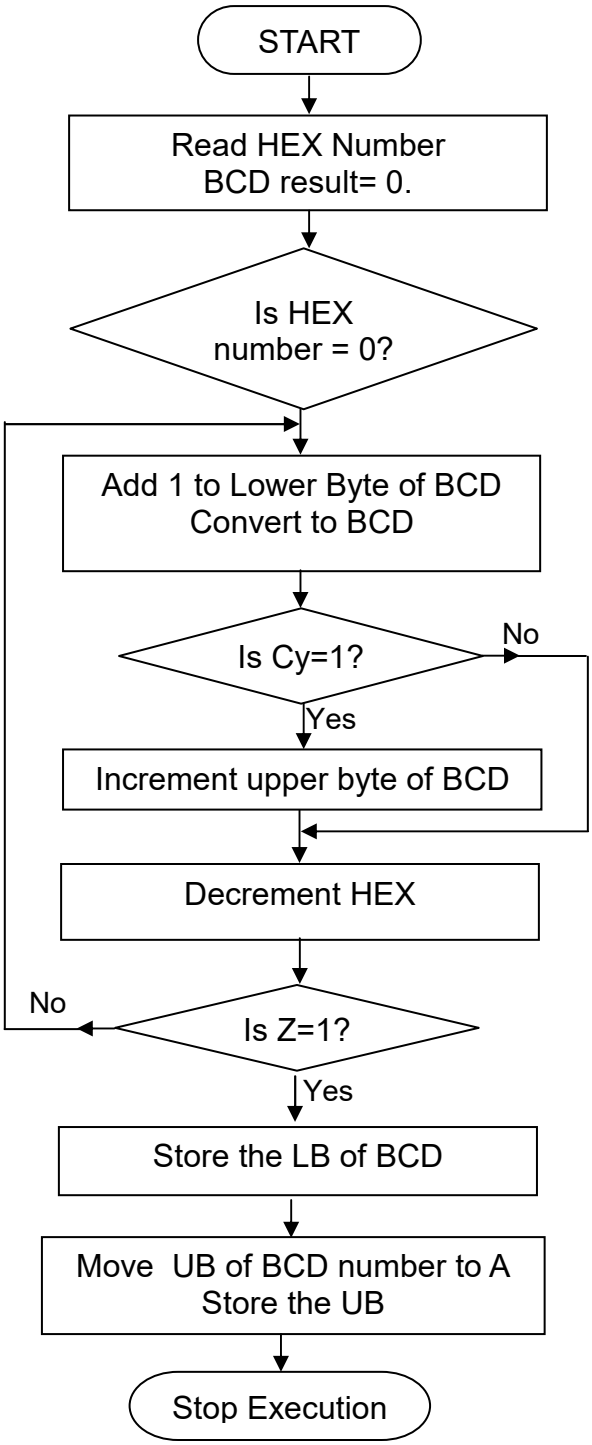


Fig: Conversion of HEX-to-BCD

Procedure

1. Develop the algorithm.
2. Each step of the algorithm with 8085 assembly language.
3. Assemble the program and correct syntax errors if any.
4. Test the program with different types of data.
    a. Manually store the data in memory location.
    b. Execute the program.
    c. Verify the result manuallyfor its correctness. If required modify the algorithm .

Program

HEX-to-BCD conversion

Address	opcode/Operand	Label	Mnemonics	Comment
4200	21 00 020		LXI H, 2000H	; Initialise Data pointer
4203	AF		XRA A	; Initialise LB of BCD to 0
4204	47		MOV B, A	; InitialiseUB of BCD to 0
4205	4E		MOV C, M	; Read HEX Data
4206	B9		CMP C	; Verify whether data input is 0
4207	CA 15 42		JZ str	; Skip calculation id BCD=0
420A	C6 01	rpt:	ADI 01	; Add 1 to Partial Sum
420C	27		DAA	; Convert to correct BCD result
420D	D2 11 42		JNC skp	; If no carry no change to UB of BCD
4210	04		INR B	; Update UB of BCD
4211	0D	skp:	DCR C	; Decrement Hexadecimal Number
4212	C2 0A 042		JNZ rpt	; Continue Decimal addition till HEX value is 0
4215	32 06 020	str:	STA 2006H	; Store LB
4218	78		MOV A, B	; UB to Accumulator
4219	32 07 020		STA 2007H	; Store UB
421C	76		HLT	; Stop Execution

Program for other method.

4200	31 00 080		LXI SP, 8000H	; Define Stack
4203	26 00		MVI H,0	; Upper Byte of Result=0
4205	3A 00 020		LDA 2000H	; Read the Hexadecimal data
4208	4F		MOV C, A	; Decimal Data to C
4209	E6 0F		ANI 0FH	; Mask Lower Nibble
420B	C6 00		ADI 00H	
420D	27		DAA	
420E	5F		MOV E, A	; Lower Nibble
420F	79		MOV A, C	; Decimal Data to A
4210	E6 F0		ANI 0F0H	; Mask Upper Nibble
4212	0F		RRC	; Rotate right without carry
4213	0F		RRC	
4214	0F		RRC	
421	5 0F		RRC	; Swap Nibblerotating A 4-times
4216	57		MOV D, A	; Upper nibble to D
4217	0E 16		MVI C, 16H	; Multiplier=16
4219	CD 27 42		CALL MUL	; Multiply
421C	83		ADD E	; Add Lower Nibble to LB of Product
421D	27		DAA	; For Decimal Addition
421E	D2 22 42		JNC NXT	; Skip modifying UB if Cy=0.
4221	24		INR H	; Update Upper Byte (when Carry=0)
4222	6F	NXT:	MOV L,A	; Lower Byte to L
4223	22 06 020		SHLD 2006H	; Store the result
4226	76		HLT	; Stop Execution

; Decimal Multiplication Subroutine (Using repeated addition Method) ; AddMultiplier repeatedly Multiplicand Times ; Multiply the content of B and D ;Multiplier and Multiplicand in register C and D ; Multiplication result returns in register A ; Register A, D are Modified ; C remains unmodified				
4227	7A	MUL:	MOV A, D	; Multiplier to A
4228	FE 00		CPI 00H	; Check Multiplicand for 0
422A	CA 38 42		JZ AHD	; Skip Repeated addition if 0
422D	AF		XRA A	; Initialise Partial Product
422E	81	ML:	ADD C	; PP = PP + Multiplier
422F	27		DAA	; To perform Decimal Addition
4230	D2 34 42		JNC SKIP	; In no carry, continue without modifying UB
4233	24		INR H	; Update UB if CY=1
4234	15	SKIP:	DCR D	; Decrement COUNT (Multiplicand)
4235	C2 2E 42		JNZ ML	; Repeat till added multiplicand times
4238	C9	AHD:	RET	; Return product in A

Testing/Observations

Sl.	HEX input (2000H)	BCD output		Remarks
		UB (2007H)	LB (2006H)	
1.	12H(35)			
2.	66H(102)			
3.	08H(08)			
4.	FFH(255)	02H(02)	55H(85)	$(FF)_{16}=(255)_{10}$
5.	00H(0)			

Conclusion

The program was tested with various types of data and the program is found to be correct.

Signature & Roll No. of the student

Aim of the experiment:

Write an 8085 ALP to convert 8-bit BCD number (Decimal) to Hexadecimal (Binary).

Instrument/Software required

PC with 8085 Simulator (GNU 8085)

Assumptions

The following memory locations are used to store data and results:

2000H : 8-bit BCD data

2006H : HEX Number (LB)

Theory

One byte of BCD number produces one byte of Hex Number. So, the result is Again for a particular symbol the magnitude in BCD and HEX are same. So, one digit BCD number can be considered as a hexadecimal number without conversion.

To convert the 8-bit BCD (Packed BCD), the two BCD digits are separated (unpacked). The upper digit is multiplied with 0AH(10) and the magnitude of lower digit is added. As the multiplication and addition are in binary not in decimal, the result is the Hexadecimal number.

Procedure

1. Develop the algorithm.
2. Each step of the algorithm with 8085 assembly language.
3. Assemble the program and correct syntax errors if any.
4. Test the program with different types of data.
    a. Manually store the data in memory location.
    b. Execute the program.
    c. Verify the result manually for its correctness. If required modify the algorithm .

Algorithm/ Flow Chart

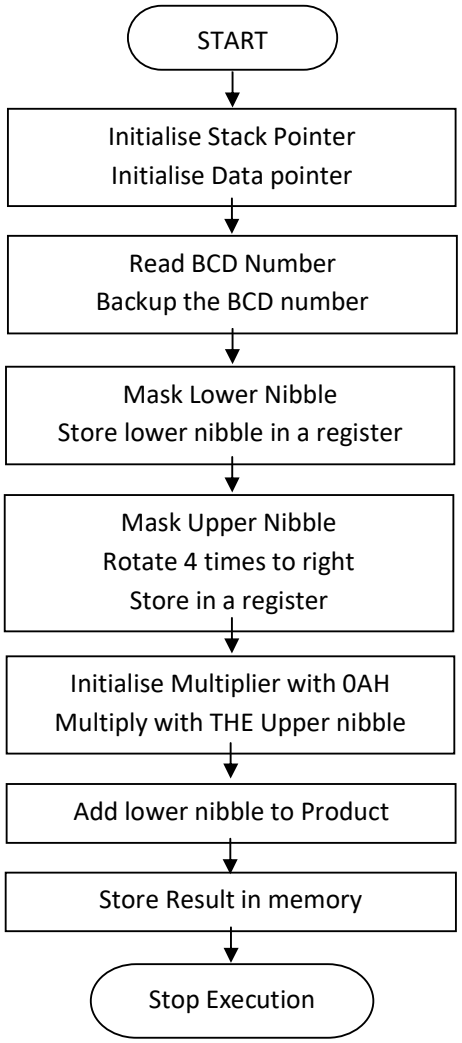


Fig: Flowchart for converting 8-bit BCD number to HEX

Program

Conversion of 8-bit BCD number to HEX

Address	opcode/ Operand	Label	Mnemonics	Comment
4200	31 00 080		LXI SP, 8000H	; Define Stack
4203	21 00 020		LXI H, 2000H	; Initialise Data Pointer
4206	46		MOV B, M	; Read the Decimal Data
4207	78		MOV A, B	; Data to Accumulator
4208	E6 0F		ANI 0FH	; Mask Lower Nibble
420A	5F		MOV E, A	; Lower Nibble
420B	78		MOV A, B	; Decimal Data to A

420C	E6 F0		ANI 0F0H	; Mask Upper Nibble
420E	0F		RRC	; Rotate Right with through Carry
420F	0F		RRC	
4210	0F		RRC	
4211	0F		RRC	; Swap Nibble by rotating 4-times
4212	57		MOV D, A	; Upper nibble to D
4213	06 0A		MVI B, 0AH	; Multiplier=10D
4215	CD 1D 42		CALL MUL	; Multiply
4218	83		ADD E	; Add Lower Nibble to Product
4219	32 06 020		STA 2006H	; Store Result
421C	76		HLT	
; Multiplication Subroutine (Using repeated addition Method) *; Multiply the content of B and D* *; Modify register A, D* *; B remains unmodified* *; Multiplier and Multiplicand in register B and D* *; Multiplication result returns in register A*				
421D	7A	MUL:	MOV A, D	; Multiplier to A
421E	FE 00		CPI 00H	; Check Multiplicand for 0
4220	CA 29 42		JZ AHD	; Skip Repeated addition if 0
4223	AF		XRA A	; Initialise Partial Product
4224	80	ML:	ADD B	; PP = PP + Multiplier
4225	15		DCR D	; Decrement COUNT (Multiplicand)
4226	C2 24 42		JNZ ML	; Repeat till added multiplicand times
4229	C9	AHD:	RET	; Return product in A

Testing/Observations

Sl.	bcd input (2000H)	HEX output (2006H)	Remarks
1.			
2.			
3.			
4.			
5.			

Conclusion

The program was tested with various types of data and the program is found to be correct.

Signature & Roll No. of the student

Aim of the Experiment :

Write assembly language programs for addition, subtraction, division, multiplicationin micro-controller.

Equipment and software used :

1. Personal Computer
2. 8051-simulator (edsim51)

Assumption:

The data are stored in the memory locations as follows

0x60➔ Augend
0x61 ➔ Addend

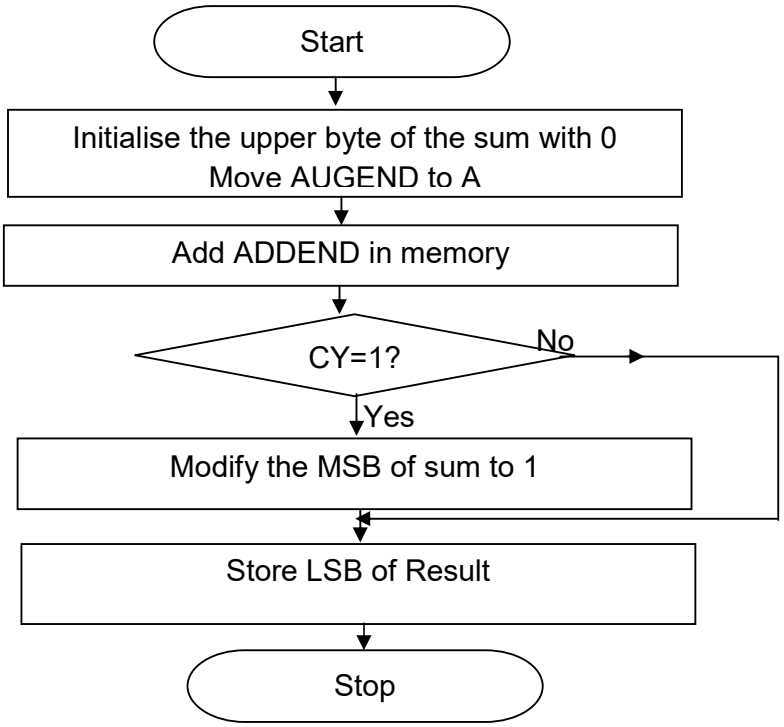
0x70➔ Sum (LB)
0x71➔ Sum (UB)
0x72➔ Difference
0x73 ➔ Borrow
0x74 ➔ Product (LB)
0x75➔ Product (UB)
0x76 ➔ Quotient
0x77➔ Remainder

Theory:

8051 supports 8-bit addition, subtraction, multiplication and division. In addition, and subtraction, the 1st operand lies in Accumulator and the 2nd operand may be in a register R7-R0, in data memory or is an immediate 8-bit data. 8051 supports addition as well as addition with carry. For decimal addition, instruction DAA is used to convert the result to BCD form. However, 8051 does not support subtraction without borrow. So, it is necessary to clear the carry flag when we want to subtract without borrow.

8051 supports 8-bit multiplication and division. The 1st operand lies in Accumulator and the 2ndoperant in register B. After multiplication, Accumulator holds the lower byte of product and B holds the upper byte of product. After division Accumulator holds the Quotient part and B holds the remainder part.

Flow Charts:



Addition of two 8-bit number stored in internal memory

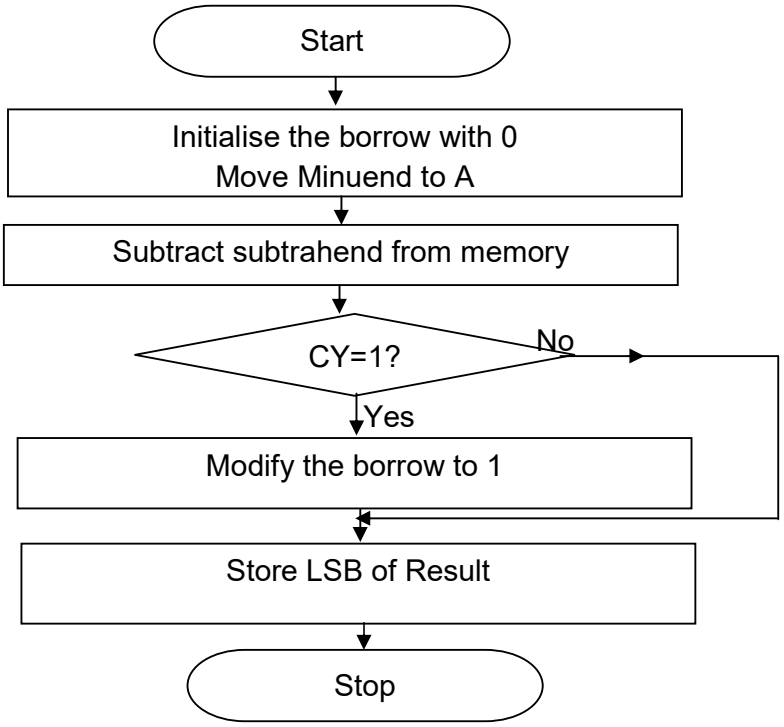


Fig: Subtraction of two 8-bit number stored in internal memory

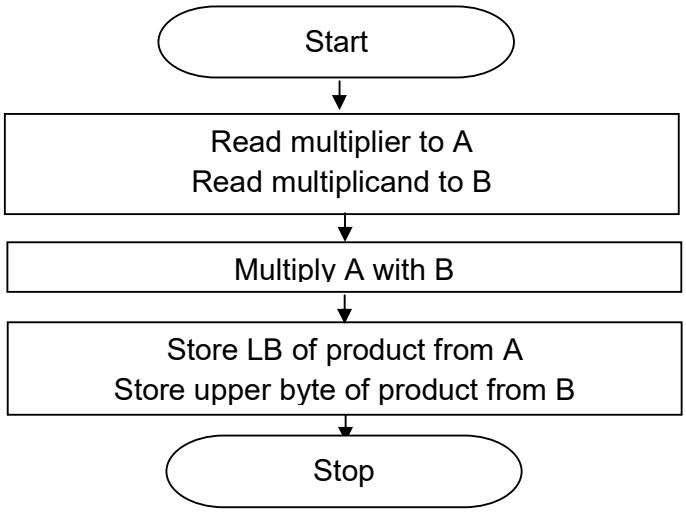


Fig: Multiplication of two 8-bit number stored in internal memory

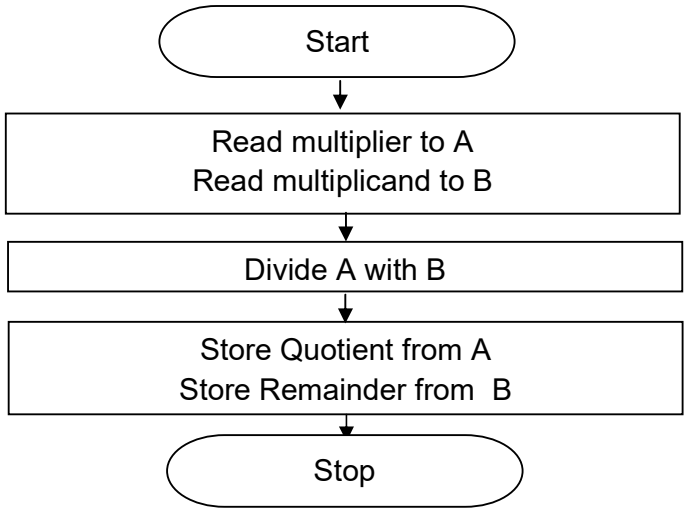


Fig: Division of two 8-bit number stored in internal memory

Program :

Addition, Subtraction, Multiplication and Division of two 8-bit numbers

Address	opCode/ Operand	Label	Mnemonics	Comment
Addition of two 8-bit number stored in internal RAM Result may be 8-bit or 16-bit				
0000	**75** 71 00		MOV 0X71, #0X00	; Initialise upper byte of Sum
0003	**E5** 60		MOV A,0X60	; Read Augend
0005	**25** 61		ADD A,0X61	; Add Addend from Memory
0007	**50** 02		JNC nxt	; Relative addressing mode
0009	**05**71		INC 0X71	; Update upper byte of Sum if cy=1
000B	**F5** 70	nxt:	MOV 0X70,A	; Store Lower Byte in Memory
Subtraction of two 8-bit number stored in internal RAM				

000D	**73**		CLR C	; Clear the carry bit
000E	**75** 73 00		MOV 0X73, #0X00	; Upper byte of result
0011	**E5** 60		MOV A,0X60	, Read Minuend
0013	**95** 61		SUBB A,0X61	; Subtract subtrahend and carry
0015	**50** 02		JNC skip	I; If no borrow skip updating
0017	**05** 73		INC 0X73	; Modify carry bit if a Borrow occurs
0019	**F5** 72	skip:	MOV 0X72,A	; Store difference
Multiplication of two 8-bit number stored in RAM				
001B	**E5** 60		MOV A,0X60	; Read multiplicand to Accumulator
001D	**85** 61 F0		MOV B,0X61	; Read multiplier to B
0020	**A4**		MUL AB	; Multiply
0021	**F5** 74		MOV 0X74, A	; Store Lower byte of product to memory
0023	**85** F0 75		MOV 0X75, B	; Store Upper byte of product to Memory
Division of two 8-bit number stored in internal RAM				
0026	**E5** 60		MOV A,0X60	; Read dividend to Accumulator
0028	**84**8561		MOV B,0X61	; Read divisor to B
002B	**F0**		DIV AB	; Divide
002C	**F5** 76		MOV 0X76, A	; Store Quotient
002E	**85**F0 77		MOV 0X77, B	; Store Remainder

Testing of the program:

1. 8-bit numbers to be stored manually stored in desired memory location.
2. Assembled the program and run it.
3. Verify the results for different set of data.

Observations:

Addition of two 8-bit number

Sl.	Augend (0x60)	Addend (0x61)	Sum	
			UB (0x71)	LB (0x70)
1.	FF	83	1	82
2.				
3.				
4.				
5.				

Subtraction of two 8-bit number

Sl.	Minuend (0x60)	Subtrahend (0x61)	Result	
			Borrow (0x73)	Difference (0x72)
1.	FF	83	0	7C
2.				

3.				
4.				
5.				

Multiplication of two 8-bit number

Sl.	Multiplicand (0x60)	Multiplier (0x61)	Product (UB) (20x75)	Product (LB) (0x74)	Remarks
1.	FF	83	82	7D	
2.					
3.					
4.					
5.					

Division of two 8-bit number

Sl.	Dividend (0x60)	Divisor (0x61)	Quotient (0x76)	Remainder (0x77)	Remarks
1.	FF	83	1	7C	
2.					
3.					
4.					
5.					

Conclusion:

The program was assembled and executed with different types of data. The results were manually verified and found correct. A register may be used to hold the MSB byte of SUM and borrow information. In such case the desired register banks has to be selected.

Signature & Roll No. of the student

Appendix A: 8085 Instruction Set by Opcode

The information in this appendix is reproduced by kind permission of the Intel Corporation. The symbols and abbreviations used are listed below.

Symbol	Meaning
A	Accumulator
B, C, D E, H, L	One of the internal registers
F	Represents the flag register
M	The 16-bit memory address currently held by the register pair H and L
byte	An 8-bit data quantity
dble	A 16-bit (two byte) data quantity
addr	A 16-bit address
port	An 8-bit I/O port address
r, r1, r2	One of the registers A, B, C, D, E, H, L
rp	One of the following register pairs B represents the register pair B and C D represents the register pair D and E H represents the register pair H and L PSW represents the register pair A and F SP represents the 16-bit stack pointer
PC	The 16-bit program counter
CY	Carry flag
P	Parity flag
AC	Auxiliary carry flag
Z	Zero flag
S	Sign flag

Data Transfer Group

These instructions transfer data between registers and memory.
Flags - none affected by instructions in this group.

Move

MOV A,A	7F	MOV B,A	47	MOV C,A	4F
MOV A,B	78	MOV B,B	40	MOV C,B	48
MOV A,C	79	MOV B,C	41	MOV C,C	49
MOV A,D	7A	MOV B,D	42	MOV C,D	4A
MOV A,E	7B	MOV B,E	43	MOV C,E	4B
MOV A,H	7C	MOV B,H	44	MOV C,H	4C
MOV A,L	7D	MOV B,L	45	MOV C,L	4D
MOV A,M	7E	MOV B,M	46	MOV C,M	4E

MOV D,A	57	MOV E,A	5F	MOV H,A	67
MOV D,B	50	MOV E,B	58	MOV H,B	60
MOV D,C	51	MOV E,C	59	MOV H,C	61
MOV D,D	52	MOV E,D	5A	MOV H,D	62
MOV D,E	53	MOV E,E	5B	MOV H,E	63
MOV D,H	54	MOV E,H	5C	MOV H,H	64
MOV D,L	55	MOV E,L	5D	MOV H,L	65
MOV D,M	56	MOV E,M	5E	MOV H,M	66

Move Immediate

MOV L,A	6F	MOV M,A	77	MVI A,byte	3E
MOV L,B	68	MOV M,B	70	MVI B,byte	06
MOV L,C	69	MOV M,C	71	MVI C,byte	0E
MOV L,D	6A	MOV M,D	72	MVI D,byte	16
MOV L,E	6B	MOV M,E	73	MVI E,byte	1E
MOV L,H	6C	MOV M,H	74	MVI H,byte	26
MOV L,L	6D	MOV M,L	75	MVI L,byte	2E
MOV L,M	6E			MVI M,byte	36

Load Immediate (Reg. pair)

LXI B,dble	01
LXI D,dble	11
LXI H,dble	21
LXI SP,dble	31

Load/Store A direct

LDAX B	0A
LDAX D	1A
STAX B	02
STAX D	12

Load/Store A direct

LDA addr	3A
STA addr	32

Load/Store HL direct

LHLD addr	2A
SHLD addr	22

Exchange HL with DE

XCHG	EB

Data Manipulation Group – Arithmetic

Instructions in this group perform arithmetic operations on data in the registers and the memory.

Add*

ADD	A	87	ADC	A	8F
	B	80		B	88
	C	81		C	89
	D	82		D	8A
	E	83		E	8B
	H	84		H	8C
	L	85		L	8D
	M	86		M	8E

Subtract*

SUB	A	97	SBB	A	9F
	B	90		B	98
	C	91		C	99
	D	92		D	9A
	E	93		E	9B
	H	94		H	9C
	L	95		L	9D
	M	96		M	9E

Add/Subtract Immediate*

ADI byte	C6
ACI byte	CE
SUI byte	D6
SBI byte	DE

Double Length Add***

DAD	B	09
	D	19
	H	29
	SP	39

Increment/Decrement**

INR	A	3C	DCR	A	3D
	B	04		B	05
	C	0C		C	0D
	D	14		D	15
	E	1C		E	1D
	H	24		H	25
	L	2C		L	2D
	M	34		M	35

Increment/Decrement Register Pair****

INX	B	03	DCX	B	0B
	D	13		D	1B
	H	23		H	2B
	SP	33		SP	3B

Decimal Adjust A*

DAA	27

Complement A****

CMA	2F

Complement/Set CY***

CMC	3F
STC	37

Arithmetic Immediate*

ADI byte	C6
ACI byte	CE
SUI byte	D6
SBI byte	DE

Notes

* All flags may be affected.
** All flags except CARRY may be affected.
*** Only CARRY FLAG affected.
**** No flags affected.

Data Manipulation Group – Logical

Instructions in this group perform logical operations on data in the registers and the memory.

AND*		**OR***		**Exclusive-OR***	
ANA A	A7	ORA A	B7	XRA A	AF
ANA B	A0	ORA B	B0	XRA B	A8
ANA C	A1	ORA C	B1	XRA C	A9
ANA D	A2	ORA D	B2	XRA D	AA
ANA E	A3	ORA E	B3	XRA E	AB
ANA H	A4	ORA H	B4	XRA H	AC
ANA L	A5	ORA L	B5	XRA L	AD
ANA M	A6	ORA M	B6	XRA M	AE

Compare*			Rotate***		Logical Immediate*	
CMP	A	BF	RLC	07	ANI byte	E6
	B	B8	RRC	0F	XRI byte	EE
	C	B9	RAL	17	ORI byte	F6
	D	BA	RAR	1F	CPI byte	FE
	E	BB				
	H	BC				
	L	BD				
	M	BE				

Notes
* All flags may be affected.
*** Only the CARRY flag may be affected.

Transfer of Control Group or Branch Group

This group of instructions alters the sequence of program flow by testing the condition flags.

Jump		Call		Return	
JMP addr	C3	CALL addr	CD	RET	C9
JNZ addr	C2	CNZ addr	C4	RNZ	C0
JZ addr	CA	CZ addr	CC	RZ	C8
JNC addr	D2	CNC addr	D4	RNC	D0
JC addr	DA	CC addr	DC	RC	D8
JPO addr	E2	CPO addr	E4	RPO	E0
JPE addr	EA	CPE addr	EC	RPE	E8
JP addr	F2	CP addr	F4	RP	F0
JM addr	FA	CM addr	FC	RM	F8

Jump Indirect

PCHL	E9

Input/Output Group

This group of instructions performs I/O instructions between the accumulator and a specified port.

IN port	DB
OUT port	D3

Stack and Machine Control Group

This group of instructions maintains the stack and internal control flags.

Stack operations

PUSH	B	C5		POP	B C1
	D	D5			D D1
	H	E5			H E1
	PSW	F5			PSW F1
XTHL	E3		SPHL	F9	

Interrupt Control

EI	FB
DI	F3
RIM	20
SIM	30

Processor Control

NOP	00
HLT	76

Restart

RST	0	C7
	1	CF
	2	D7
	3	DF
	4	E7
	5	EF
	6	F7
	7	FF

Appendix B: 8085 Instruction Set by Clock Cycles

Mnemonic	*Clock cycles*
MOVE, LOAD AND STORE	
MOV r1,r2	4
MOV M,r	7
MOV r,M	7
MVI r	7
MVI M	10
LXI B	10
LXI D	10
LXI H	10
LXI SP	10
STAX B	7
STAX D	7
LDAX B	7
LDAX D	7
STA	13
LDA	13
SHLD	16
LHLD	16
XCHG	4
STACK OPERATIONS	
PUSH B	12
PUSH D	12
PUSH H	12
PUSH PSW	12
POP B	10
POP D	10
POP H	10
POP PSW	10
XTHL	16
SPHL	6

Mnemonic	*Clock cycles*
RETURN	
RET	10
RC	6/12
RNC	6/12
RZ	6/12
RNZ	6/12
RP	6/12
RM	6/12
RPE	6/12
RPO	6/12
RESTART	
RST	12
INPUT/OUTPUT	
IN	10
OUT	10
INCREMENT AND DECREMENT	
INR r	4
DCR r	4
INR M	10
DCR M	10
INX B	6
INX D	6
INX H	6
INX SP	6
DCX B	6
DCX D	6
DCX H	6
DCX SP	6

JUMP	
JMP	10
JC	7/10
JNC	7/10
JZ	7/10
JNZ	7/10
JP	7/10
JM	7/10
JPE	7/10
JPO	7/10
PCHL	6

CALL	
CALL	18
CC	9/18
CNC	9/18
CZ	9/18
CNZ	9/18
CP	9/18
CM	9/18
CPE	9/18
CPO	9/18

ROTATE	
RLC	4
RRC	4
RAL	4
RAR	4

SPECIALS	
CMA	4
STC	4
CMC	4
DAA	4

CONTROL	
EI	4
DI	4
NOP	4
HLT	5

ADD	
ADD r	4
ADC r	4
ADD M	7
ADC M	7
ADI	7
ACI	7
DAD B	10
DAD D	10
DAD H	10
DAD SP	10

SUBTRACT	
SUB r	4
SBB r	4
SUB M	7
SBB M	7
SUI	7
SBI	7

LOGICAL	
ANA r	4
XRA r	4
ORA r	4
CMP r	4
ANA M	7
XRA M	7
ORA M	7
CMP M	7
ANI	7
XRI	7
ORI	7
CPI	7

INTERRUPT MASK	
RIM	4
SIM	4

Note: Two possible cycle times, for example, 6/12, indicates that the number of instruction cycles involved is dependent on the condition flags.

MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings(1)

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

(1)Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

Rn — Register R7–R0 of the currently selected Register Bank.

direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].

@Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.

#data — 8-bit constant included in instruction.

#data 16 — 16-bit constant included in instruction.

addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.

bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

Table 11. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A, #data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A, #data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr, #data
44	2	ORL	A, #data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr, #data
54	2	ANL	A, #data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr, #data
64	2	XRL	A, #data
65	2	XRL	A,data addr

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A