Laboratory Journal

of

MICROPROCESSOR

For completion of term work of 5^{th} *semester*

curriculum program

Bachelor of Technology

In

ELECTRONICS AND TELECOMMUNICATION

ENGINEERING

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION

ENGINEERING

Dr. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY

Lonere-402 103, Tal. Mangaon, Dist. Raigad (MS)

INDIA

List of Experiments

1. INTRODUCTION TO 8085

INTEL 8085 is one of the most popular 8-bit microprocessor capable of addressing 64 KB of memory and its architecture is simple. The device has 40 pins, requires +5 V power supply and can operate with 3MHz single phase clock.

ALU (Arithmetic Logic Unit):

The 8085A has a simple 8-bit ALU and it works in coordination with the accumulator, temporary registers, 5 flags and arithmetic and logic circuits. ALU has the capability of performing several mathematical and logical operations. The temporary registers are used to hold the data during an arithmetic and logic operation. The result is stored in the accumulator and the flags are set or reset according to the result of the operation. The flags are affected by the arithmetic and logic operation. They are as follows:

- Sign flag

  After the execution of the arithmetic - logic operation if the bit D7 of the result is 1, the sign flag is set. This flag is used with signed numbers. If it is 1, it is a negative number and if it is 0, it is a positive number.

- Zero flag

  The zero flag is set if the ALU operation results in zero. This flag is modified by the result in the accumulator as well as in other registers.

- Auxillary carry flag

  In an arithmetic operation when a carry is generated by digit D3 and passed on to D4, the auxillary flag is set.

- Parity flag

  After arithmetic – logic operation, if the result has an even number of 1's the flag is set. If it has odd number of 1's it is reset.

- Carry flag

  If an arithmetic operation results in a carry, the carry flag is set. The carry flag also serves as a borrow flag for subtraction.

Timing and control unit

This unit synchronizes all the microprocessor operation with a clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals RD (read) and WR (write) indicate the availability of data on the data bus.

Instruction register and decoder

The instruction register and decoder are part of the ALU. When an instruction is fetched from memory it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow.

Register array

The 8085 has six general purpose registers to store 8-bit data during program execution. These registers are identified as B, C, D, E, H and L. they can be combined as BC, DE and HL to perform 16-bit operation.

Accumulator

Accumulator is an 8-bit register that is part of the ALU. This register is used to store 8-bit data and to perform arithmetic and logic operation. The result of an operation is stored in the accumulator.

Program counter

The program counter is a 16-bit register used to point to the memory address of the next instruction to be executed.

Stack pointer

It is a 16-bit register which points to the memory location in R/W memory, called the Stack.

Communication lines

8085 microprocessor performs data transfer operations using three communication lines called buses. They are address bus, data bus and control bus.

- Address bus – it is a group of 16-bit lines generally identified as $A_0 – A_{15.}$ The address bus is unidirectional i.e., the bits flow in one direction from microprocessor to the peripheral devices. It is capable of addressing 2^{16} memory locations.
- Data bus – it is a group of 8 lines used for data flow and it is bidirectional. The data ranges from 00 – FF.
- Control bus – it consist of various single lines that carry synchronizing signals. The microprocessor uses such signals for timing purpose.

2(A). 8 BIT DATA ADDITION

AIM:

To add two 8 bit numbers stored at consecutive memory locations.

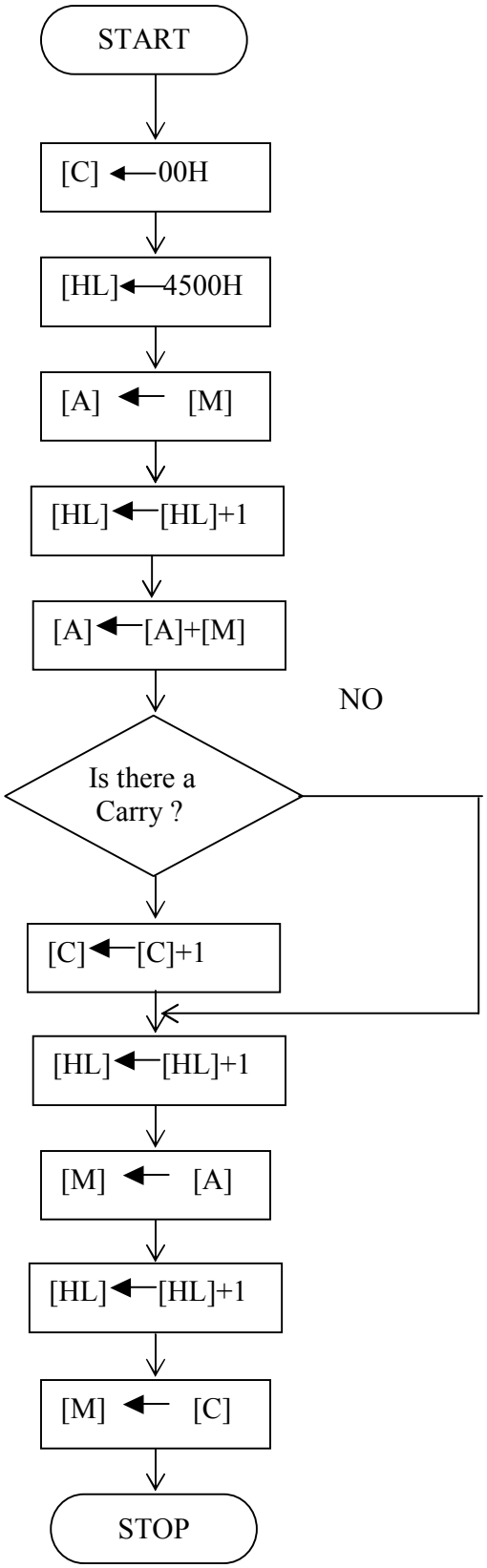
ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and add it to the accumulator.
4. Store the answer at another memory location.

RESULT:

Thus the 8 bit numbers stored at 4500 &4501 are added and the result stored at 4502 & 4503.

FLOW CHART:


START
[C] ←00H
[HL]←4500H
[A] ← [M]
[HL]←[HL]+1
[A]←[A]+[M]
NO
Is there a Carry ?
[C]←[C]+1
[HL]←[HL]+1
[M] ← [A]
[HL]←[HL]+1
[M] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	C, 00	Clear C reg.
4101					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	A, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next memory Location.
4107			ADD	M	Add first number to acc. Content.
4108			JNC	L1	Jump to location if result does not yield carry.
4109					
410A					
410B			INR	C	Increment C reg.
410C		L1	INX	H	Increment HL reg. to point next memory Location.
410D			MOV	M, A	Transfer the result from acc. to memory.
410E			INX	H	Increment HL reg. to point next memory Location.
410F			MOV	M, C	Move carry to memory
4110			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

2(B). 8 BIT DATA SUBTRACTION

AIM:

To Subtract two 8 bit numbers stored at consecutive memory locations.

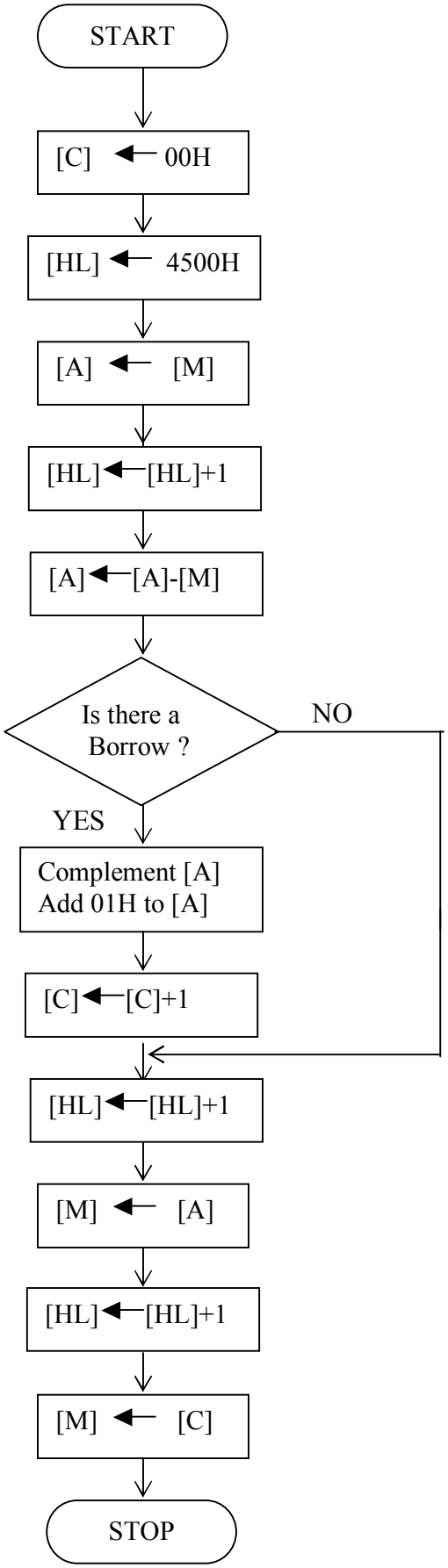
ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and subtract from the accumulator.
4. If the result yields a borrow, the content of the acc. is complemented and 01H is added to it (2's complement). A register is cleared and the content of that reg. is incremented in case there is a borrow. If there is no borrow the content of the acc. is directly taken as the result.
5. Store the answer at next memory location.

RESULT:

Thus the 8 bit numbers stored at 4500 &4501 are subtracted and the result stored at 4502 & 4503.

FLOW CHART:


START
[C] ← 00H
[HL] ← 4500H
[A] ← [M]
[HL]←[HL]+1
[A]←[A]-[M]
Is there a Borrow ?
NO
YES
Complement [A]
Add 01H to [A]
[C]←[C]+1
[HL]←[HL]+1
[M] ← [A]
[HL]←[HL]+1
[M] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	C, 00	Clear C reg.
4102					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	A, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next mem. Location.
4107			SUB	M	Subtract first number from acc. Content.
4108			JNC	L1	Jump to location if result does not yield borrow.
4109					
410A					
410B			INR	C	Increment C reg.
410C			CMA		Complement the Acc. content
410D			ADI	01H	Add 01H to content of acc.
410E					
410F		L1	INX	H	Increment HL reg. to point next mem. Location.
4110			MOV	M, A	Transfer the result from acc. to memory.
4111			INX	H	Increment HL reg. to point next mem. Location.
4112			MOV	M, C	Move carry to mem.
4113			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

3(A). 8 BIT DATA MULTIPLICATION

AIM:

To multiply two 8 bit numbers stored at consecutive memory locations and store the result in memory.

ALGORITHM:

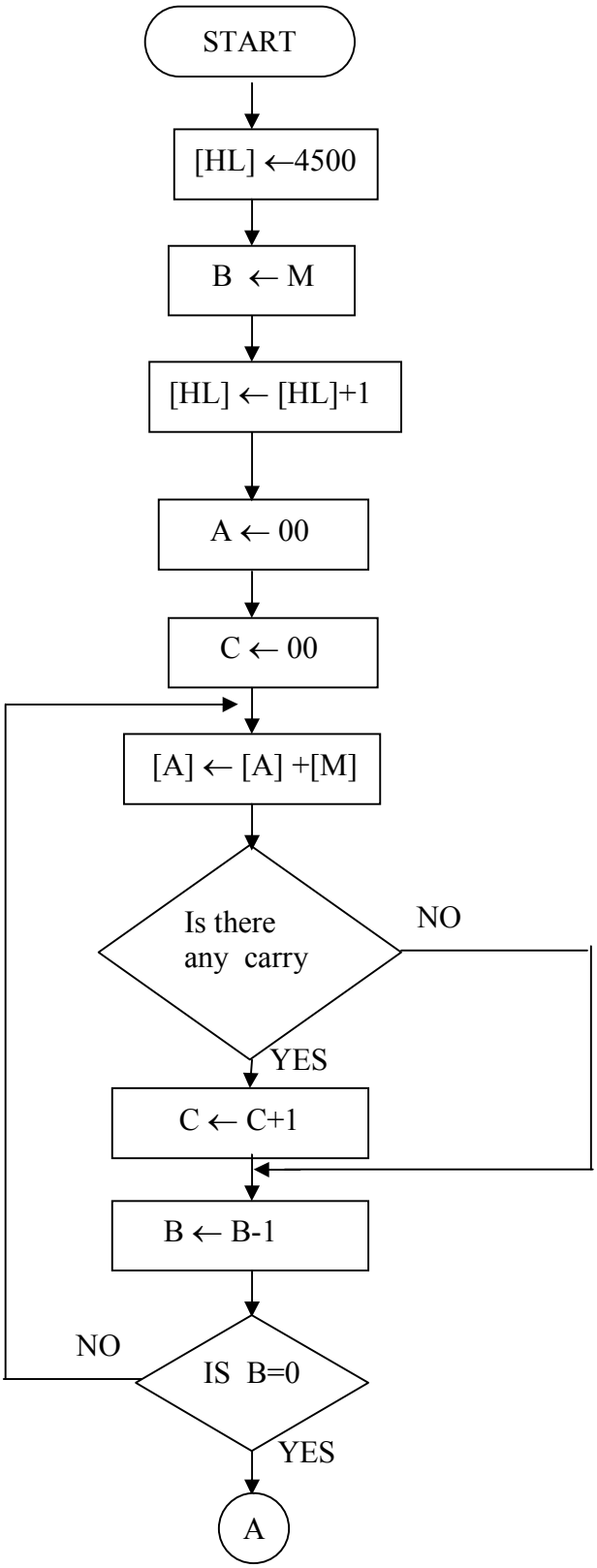
LOGIC: Multiplication can be done by repeated addition.

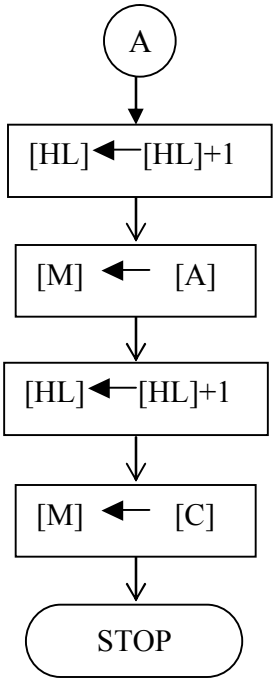
1. Initialize memory pointer to data location.
2. Move multiplicand to a register.
3. Move the multiplier to another register.
4. Clear the accumulator.
5. Add multiplicand to accumulator
6. Decrement multiplier
7. Repeat step 5 till multiplier comes to zero.
8. The result, which is in the accumulator, is stored in a memory location.

RESULT:

Thus the 8-bit multiplication was done in 8085μp using repeated addition method.

FLOW CHART:


START
[HL] ←4500
B ← M
[HL] ← [HL]+1
A ← 00
C ← 00
[A] ← [A] +[M]
Is there any carry
NO
YES
C ← C+1
B ← B-1
IS B=0
NO
YES
A


A
[HL]←[HL]+1
[M] ← [A]
[HL]←[HL]+1
[M] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	LXI	H, 4500	Initialize HL reg. to 4500
4101					
4102					
4103			MOV	B, M	Transfer first data to reg. B
4104			INX	H	Increment HL reg. to point next mem. Location.
4105			MVI	A, 00H	Clear the acc.
4106					
4107			MVI	C, 00H	Clear C reg for carry
4108					
4109		L1	ADD	M	Add multiplicand multiplier times.
410A			JNC	NEXT	Jump to NEXT if there is no carry
410B					
410C					
410D			INR	C	Increment C reg
410E		NEXT	DCR	B	Decrement B reg
410F			JNZ	L1	Jump to L1 if B is not zero.
4110					
4111					
4112			INX	H	Increment HL reg. to point next mem. Location.
4113			MOV	M, A	Transfer the result from acc. to memory.
4114			INX	H	Increment HL reg. to point next mem. Location.
4115			MOV	M, C	Transfer the result from C reg. to memory.
4116			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

3(B). 8 BIT DIVISION

AIM:

To divide two 8-bit numbers and store the result in memory.

ALGORITHM:

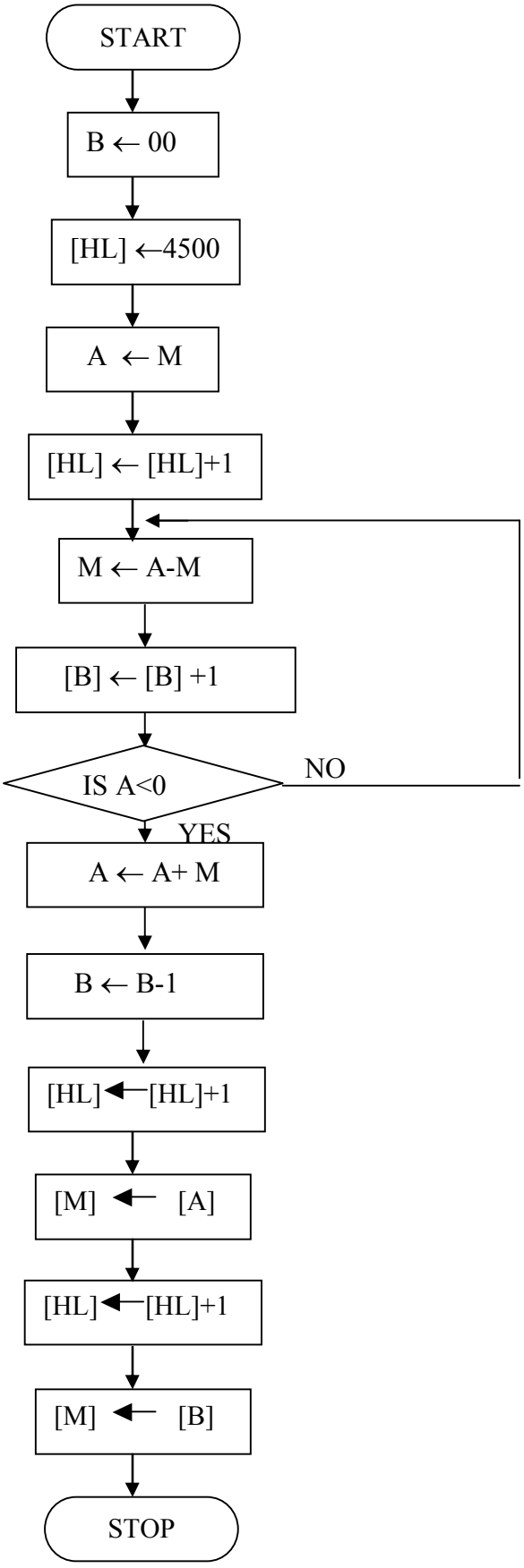
LOGIC: Division is done using the method Repeated subtraction.

1. Load Divisor and Dividend
2. Subtract divisor from dividend
3. Count the number of times of subtraction which equals the quotient
4. Stop subtraction when the dividend is less than the divisor .The dividend now becomes the remainder. Otherwise go to step 2.
5. stop the program execution.

RESULT:

Thus an ALP was written for 8-bit division using repeated subtraction method and executed using 8085μ p kits

FLOWCHART:


START
B ← 00
[HL] ←4500
A ← M
[HL] ← [HL]+1
M ← A-M
[B] ← [B] +1
IS A<0
NO
YES
A ← A+ M
B ← B-1
[HL]←[HL]+1
[M] ← [A]
[HL]←[HL]+1
[M] ← [B]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100			MVI	B,00	Clear B reg for quotient
4101					
4102			LXI	H,4500	Initialize HL reg. to 4500H
4103					
4104					
4105			MOV	A,M	Transfer dividend to acc.
4106			INX	H	Increment HL reg. to point next mem. Location.
4107		LOOP	SUB	M	Subtract divisor from dividend
4108			INR	B	Increment B reg
4109			JNC	LOOP	Jump to LOOP if result does not yield borrow
410A					
410B					
410C			ADD	M	Add divisor to acc.
410D			DCR	B	Decrement B reg
410E			INX	H	Increment HL reg. to point next mem. Location.
410F			MOV	M,A	Transfer the remainder from acc. to memory.
4110			INX	H	Increment HL reg. to point next mem. Location.
4111			MOV	M,B	Transfer the quotient from B reg. to memory.
4112			HLT		Stop the program

OBSERVATION:

S.NO	INPUT		OUTPUT	
	ADDRESS	DATA	ADDRESS	DATA
1	4500		4502	
	4501		4503	
2	4500		4502	
	4501		4503	

4(A). 16 BIT DATA ADDITION

AIM:

To add two 16-bit numbers stored at consecutive memory locations.

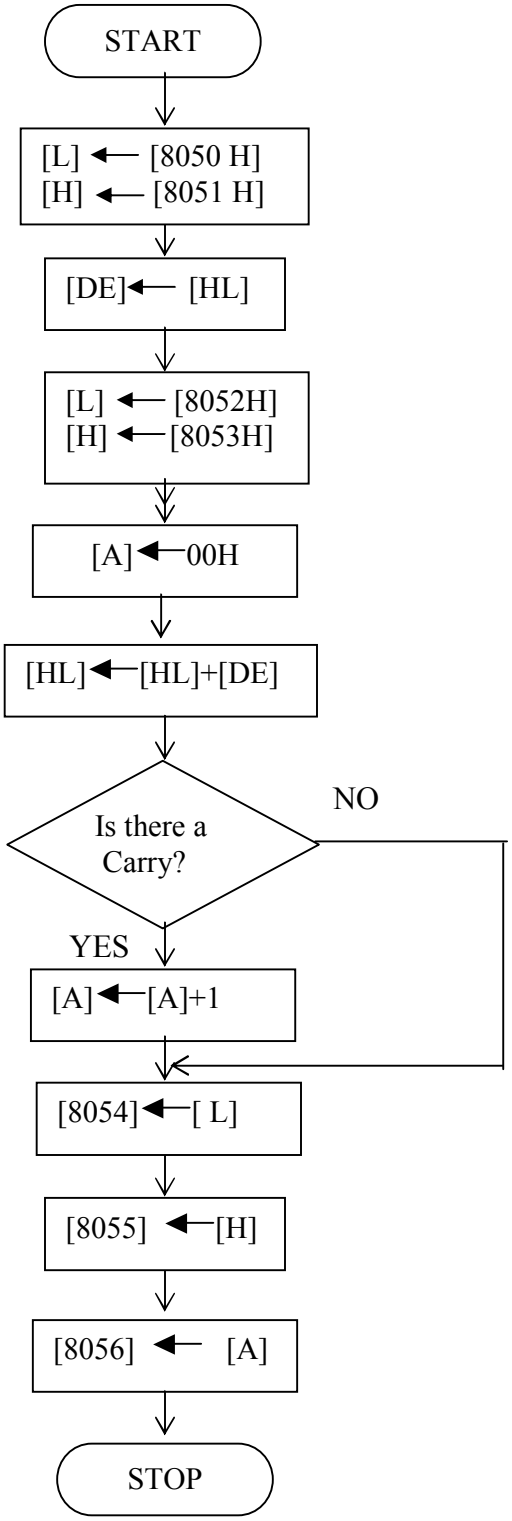
ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory and store in Register pair.
3. Get the second number in memory and add it to the Register pair.
4. Store the sum & carry in separate memory locations.

RESULT:

Thus an ALP program for 16-bit addition was written and executed in 8085μp using special instructions.

FLOW CHART:


START
[L] ← [8050 H]
[H] ← [8051 H]
[DE] ← [HL]
[L] ← [8052H]
[H] ← [8053H]
[A] ← 00H
[HL] ← [HL]+[DE]
Is there a Carry?
NO
YES
[A] ← [A]+1
[8054] ← [L]
[8055] ← [H]
[8056] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
8000		START	LHLD	8050H	Load the augend in DE pair through HL pair.
8001					
8002					
8003			XCHG		
8004			LHLD	8052H	Load the addend in HL pair.
8005					
8006					
8007			MVI	A, 00H	Initialize reg. A for carry
8008					
8009			DAD	D	Add the contents of HL Pair with that of DE pair.
800A			JNC	LOOP	If there is no carry, go to the instruction labeled LOOP.
800B					
800C					
800D			INR	A	Otherwise increment reg. A
800E		LOOP	SHLD	8054H	Store the content of HL Pair in 8054H(LSB of sum)
800F					
8010					
8011			STA	8056H	Store the carry in 8056H through Acc. (MSB of sum).
8012					
8013					
8014			HLT		Stop the program.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8050H		8054H	
8051H		8055H	
8052H		8056H	
8053H			

4(B). 16 BIT DATA SUBTRACTION

AIM:

To subtract two 16-bit numbers stored at consecutive memory locations.

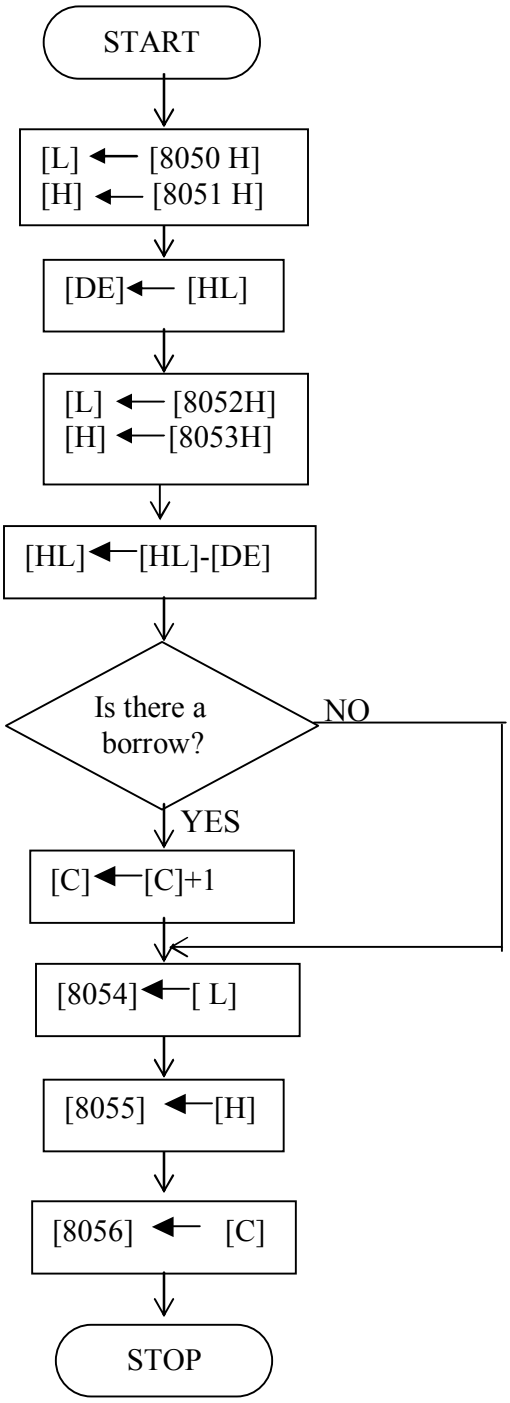
ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the subtrahend from memory and transfer it to register pair.
3. Get the minuend from memory and store it in another register pair.
4. Subtract subtrahend from minuend.
5. Store the difference and borrow in different memory locations.

RESULT:

Thus an ALP program for subtracting two 16-bit numbers was written and executed.

FLOW CHART:


START
[L] ← [8050 H]
[H] ← [8051 H]
[DE] ← [HL]
[L] ← [8052H]
[H] ← [8053H]
[HL] ← [HL]-[DE]
Is there a borrow?
NO
YES
[C] ← [C]+1
[8054] ← [L]
[8055] ← [H]
[8056] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000		START	MVI	C, 00	Initialize C reg.
8001					
8002			LHLD	8050H	Load the subtrahend in DE reg. Pair through HL reg. pair.
8003					
8004					
8005			XCHG		
8006			LHLD	8052H	Load the minuend in HL reg. Pair.
8007					
8008					
8009			MOV	A, L	Move the content of reg. L to Acc.
800A			SUB	E	Subtract the content of reg. E from that of acc.
800B			MOV	L, A	Move the content of Acc. to reg. L
800C			MOV	A, H	Move the content of reg. H to Acc.
800D			SBB	D	Subtract content of reg. D with that of Acc.
800E			MOV	H, A	Transfer content of acc. to reg. H
800F			SHLD	8054H	Store the content of HL pair in memory location 8504H.
8010					
8011					
8012			JNC	NEXT	If there is borrow, go to the instruction labeled NEXT.
8013					
8014					
8015			INR	C	Increment reg. C
8016		NEXT	MOV	A, C	Transfer the content of reg. C to Acc.
8017			STA	8056H	Store the content of acc. to the memory location 8506H
8018					
8019					
801A			HLT		Stop the program execution.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8050H		8054H	
8051H		8055H	
8052H		8056H	
8053H			

5(A). 16 BIT MULTIPLICATION

AIM:

To multiply two 16 bit numbers and store the result in memory.

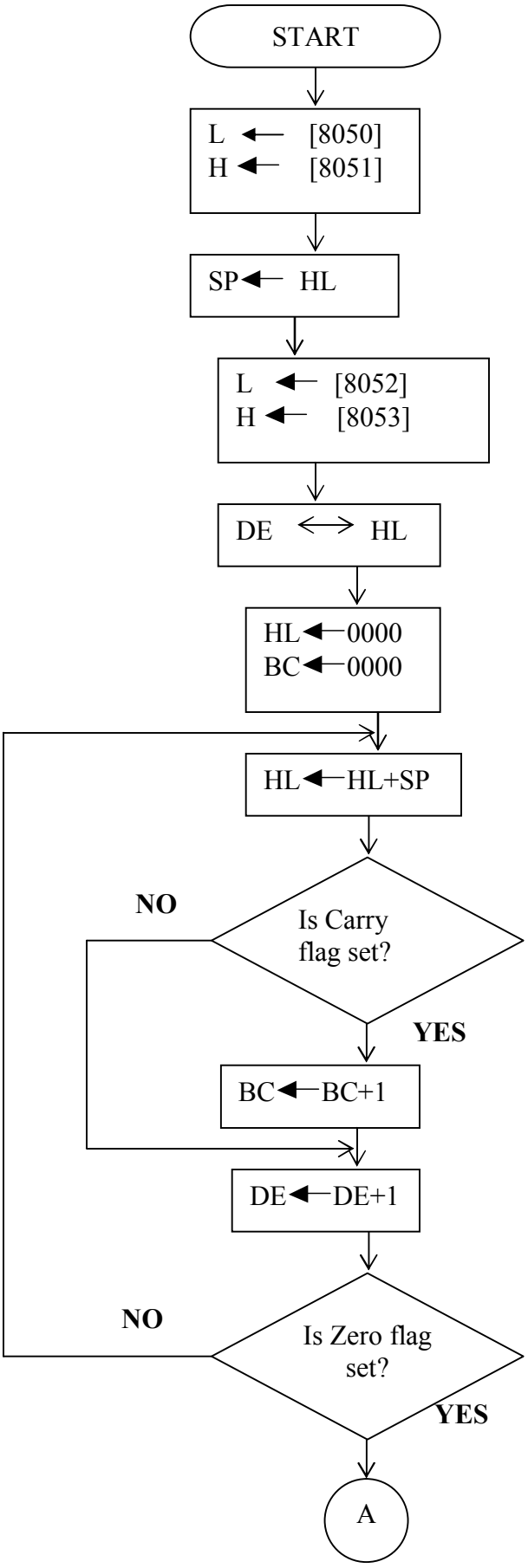
ALGORITHM:

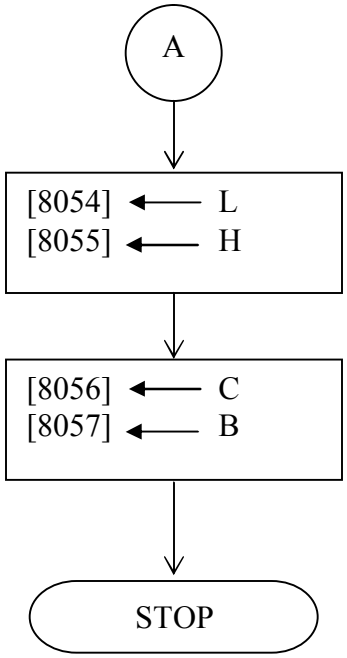
1. Get the multiplier and multiplicand.
2. Initialize a register to store partial product.
3. Add multiplicand, multiplier times.
4. Store the result in consecutive memory locations.

RESULT:

Thus the 16-bit multiplication was done in 8085μp using repeated addition method.

FLOWCHART:


START
L ← [8050]
H ← [8051]
SP ← HL
L ← [8052]
H ← [8053]
DE ⟷ HL
HL ← 0000
BC ← 0000
HL ← HL+SP
Is Carry flag set?
NO
YES
BC ← BC+1
DE ← DE+1
Is Zero flag set?
NO
YES
A


A
[8054] ← L
[8055] ← H
[8056] ← C
[8057] ← B
STOP

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000		START	LHLD	8050	Load the first No. in stack pointer through HL reg. pair
8001					
8002					
8003			SPHL		
8004			LHLD	8052	Load the second No. in HL reg. pair & Exchange with DE reg. pair.
8005					
8006					
8007			XCHG		
8008			LXI	H, 0000H	Clear HL & DE reg. pairs.
8009					
800A					
800B			LXI	B, 0000H	
800C					
800D					
800E		LOOP	DAD	SP	Add SP with HL pair.
800F			JNC	NEXT	If there is no carry, go to the instruction labeled NEXT
8010					
8011					
8012			INX	B	Increment BC reg. pair
8013		NEXT	DCX	D	Decrement DE reg. pair.
8014			MOV	A,E	Move the content of reg. E to Acc.
8015			ORA	D	OR Acc. with D reg.
8016			JNZ	LOOP	If there is no zero, go to instruction labeled LOOP
8017					
8018					
8019			SHLD	8054	Store the content of HL pair in memory locations 8054 & 8055.
801A					
801B					
801C			MOV	A, C	Move the content of reg. C to Acc.
801D			STA	8056	Store the content of Acc. in memory location 8056.
801E					
801F					
8020			MOV	A, B	Move the content of reg. B to Acc.
8021			STA	8057	Store the content of Acc. in memory location 8056.
8022					
8023					
8024			HLT		Stop program execution

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8050		8054	
8051		8055	
8052		8056	
8053		8057	

5(B). 16- BIT DIVISION

AIM:

To divide two 16-bit numbers and store the result in memory using 8085 mnemonics.

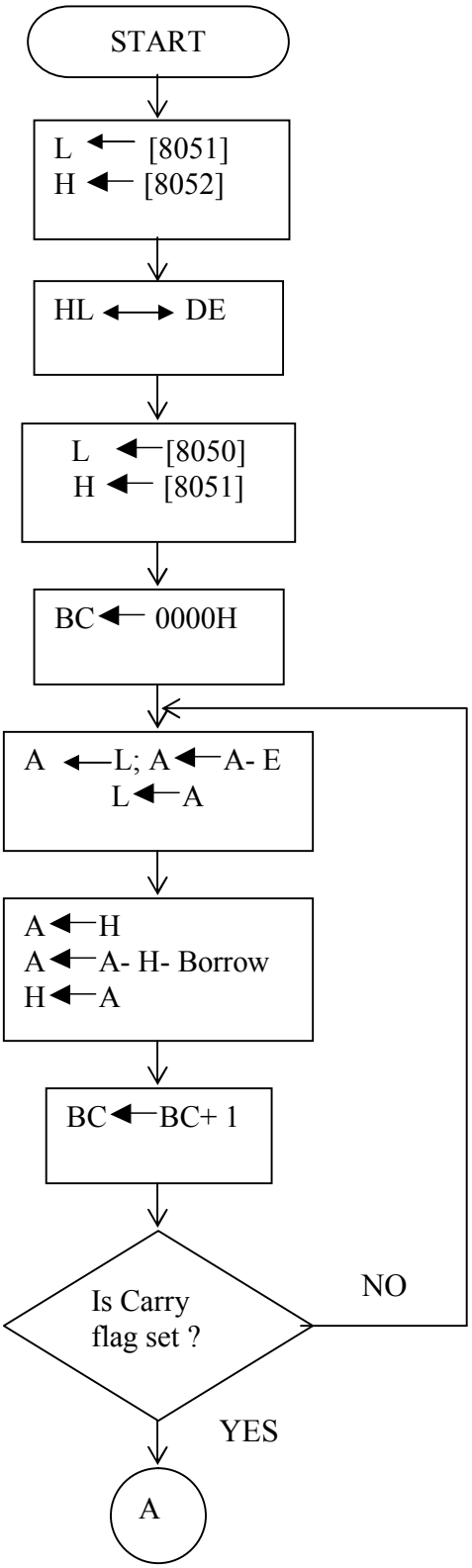
ALGORITHM:

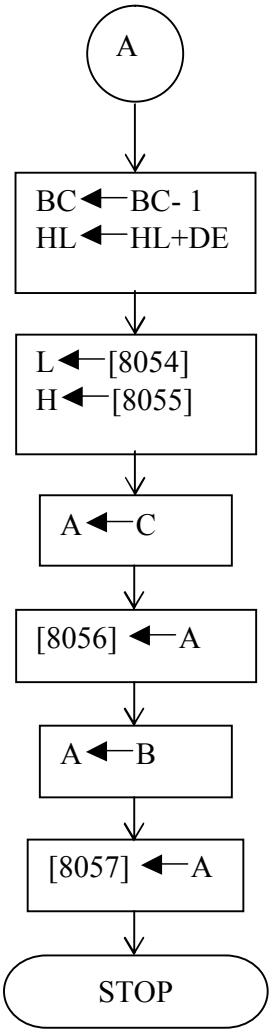
1. Get the dividend and divisor.
2. Initialize the register for quotient.
3. Repeatedly subtract divisor from dividend till dividend becomes less than divisor.
4. Count the number of subtraction which equals the quotient.
5. Store the result in memory.

RESULT:

Thus the 16-bit Division was done in 8085μp using repeated subtraction method.

FLOWCHART:


START
L ← [8051]
H ← [8052]
HL ↔ DE
L ← [8050]
H ← [8051]
BC ← 0000H
A ← L; A ← A- E
L ← A
A ← H
A ← A- H- Borrow
H ← A
BC ← BC+ 1
Is Carry flag set ?
NO
YES
A


A
BC ← BC- 1
HL ← HL+DE
L ← [8054]
H ← [8055]
A ← C
[8056] ← A
A ← B
[8057] ← A
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000		START	LHLD	8052	Load the first No. in stack pointer through HL reg. pair
8001					
8002					
8003			XCHG		
8004			LHLD	8050	Load the second No. in HL reg. pair & Exchange with DE reg. pair.
8005					
8006					
8007			LXI	B, 0000H	Clear BC reg. pair.
8008					
8009					
800A		LOOP	MOV	A, L	Move the content of reg. L to Acc.
800B			SUB	E	Subtract reg. E from that of Acc.
800C			MOV	L, A	Move the content of Acc to L.
800D			MOV	A, H	Move the content of reg. H Acc.
800E			SBB	D	Subtract reg. D from that of Acc.
800F			MOV	H, A	Move the content of Acc to H.
8010			INX	B	Increment reg. Pair BC
8011			JNC	LOOP	If there is no carry, go to the location labeled LOOP.
8012					
8013					
8014			DCX	B	Decrement BC reg. pair.
8015			DAD	D	Add content of HL and DE reg. pairs.
8016			SHLD	8054	Store the content of HL pair in 8054 & 8055.
8017					
8018					
8019			MOV	A, C	Move the content of reg. C to Acc.
801A			STA	8056	Store the content of Acc. in memory 8056
801B					
801C					
801D			MOV	A, B	Move the content of reg. B to Acc.
801E			STA	8057	Store the content of Acc. in memory 8057.
801F					
8020					
8021			HLT		Stop the program execution.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8050		8054	
8051		8055	
8052		8056	
8053		8057	

6(A). LARGEST ELEMENT IN AN ARRAY

AIM:

To find the largest element in an array.

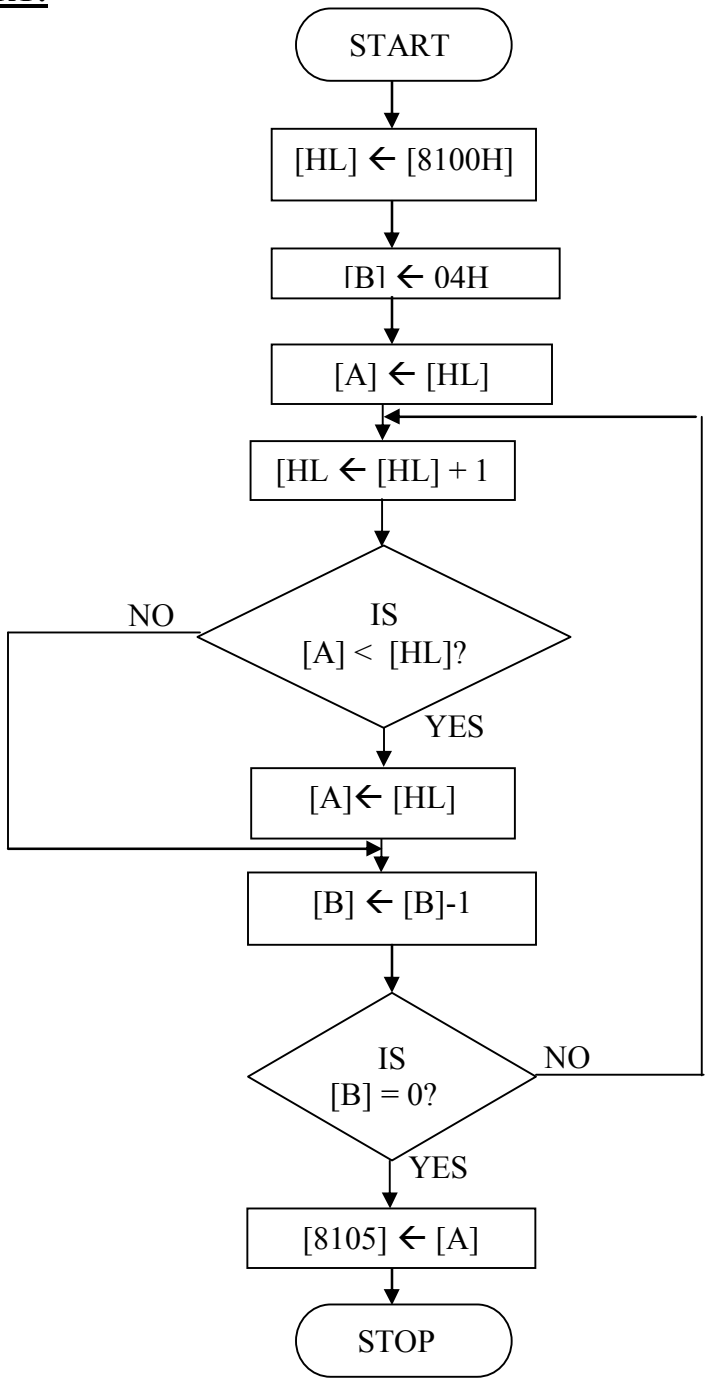
ALGORITHM:

1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero
10. Store the result (accumulator content) in the specified memory location.

RESULT:

Thus the largest number in the given array is found out.

FLOW CHART:


START
[HL] ← [8100H]
[B] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
NO
YES
[A]← [HL]
[B] ← [B]-1
IS
[B] = 0?
NO
YES
[8105] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8001			LXI	H,8100	Initialize HL reg. to 8100H
8002					
8003					
8004			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
8005					
8006			MOV	A,M	Transfer first data to acc.
8007		LOOP1	INX	H	Increment HL reg. to point next memory location
8008			CMP	M	Compare M & A
8009			JNC	LOOP	If A is greater than M then go to loop
800A					
800B					
800C			MOV	A,M	Transfer data from M to A reg
800D		LOOP	DCR	B	Decrement B reg
800E			JNZ	LOOP1	If B is not Zero go to loop1
800F					
8010					
8011			STA	8105	Store the result in a memory location.
8012					
8013					
8014			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8105	
8101			
8102			
8103			
8104			

6(B). SMALLEST ELEMENT IN AN ARRAY

AIM:

To find the smallest element in an array.

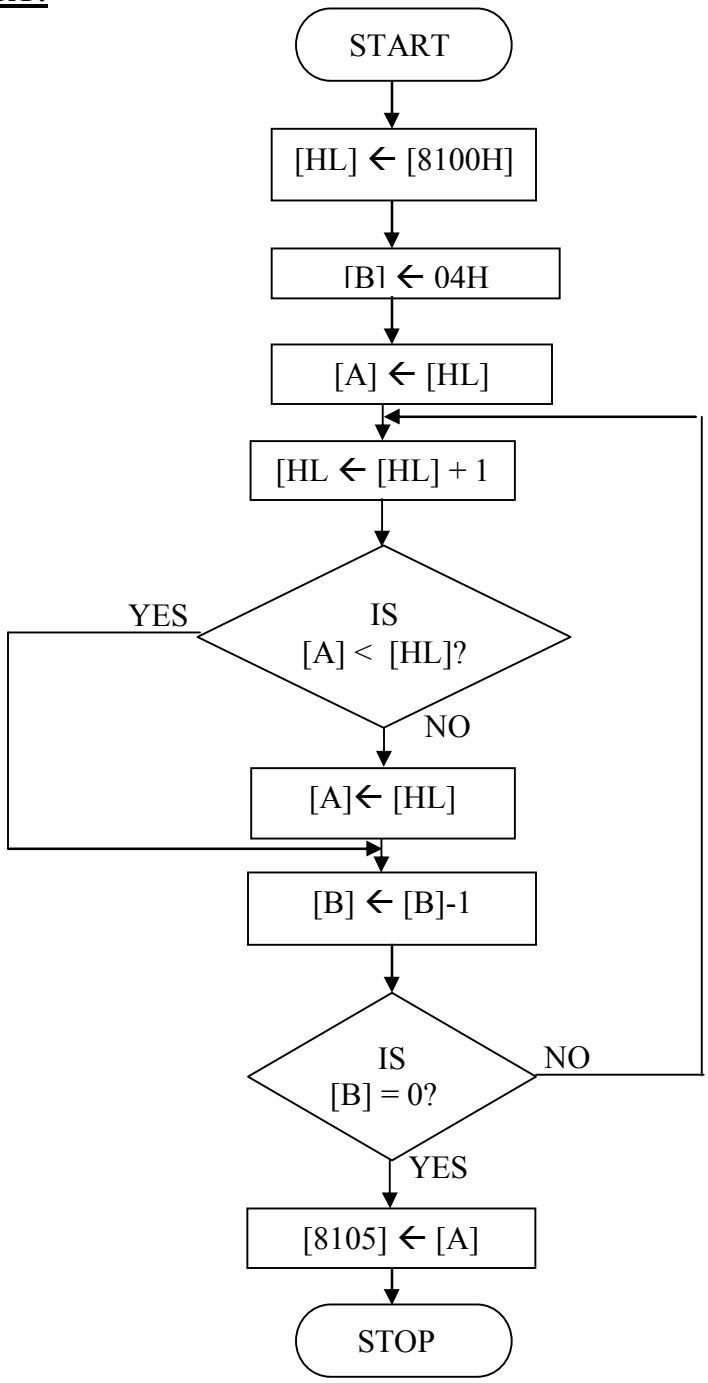
ALGORITHM:

1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero
10. Store the result (accumulator content) in the specified memory location.

RESULT:

Thus the smallest number in the given array is found out.

FLOW CHART:


START
[HL] ← [8100H]
[B] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
YES
NO
[A]← [HL]
[B] ← [B]-1
IS
[B] = 0?
NO
YES
[8105] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8001			LXI	H,8100	Initialize HL reg. to 8100H
8002					
8003					
8004			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
8005					
8006			MOV	A,M	Transfer first data to acc.
8007		LOOP1	INX	H	Increment HL reg. to point next memory location
8008			CMP	M	Compare M & A
8009			JC	LOOP	If A is lesser than M then go to loop
800A					
800B					
800C			MOV	A,M	Transfer data from M to A reg
800D		LOOP	DCR	B	Decrement B reg
800E			JNZ	LOOP1	If B is not Zero go to loop1
800F					
8010					
8011			STA	8105	Store the result in a memory location.
8012					
8013					
8014			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8105	
8101			
8102			
8103			
8104			

7(A).ASCENDING ORDER

AIM:

To sort the given number in the ascending order using 8085 microprocessor.

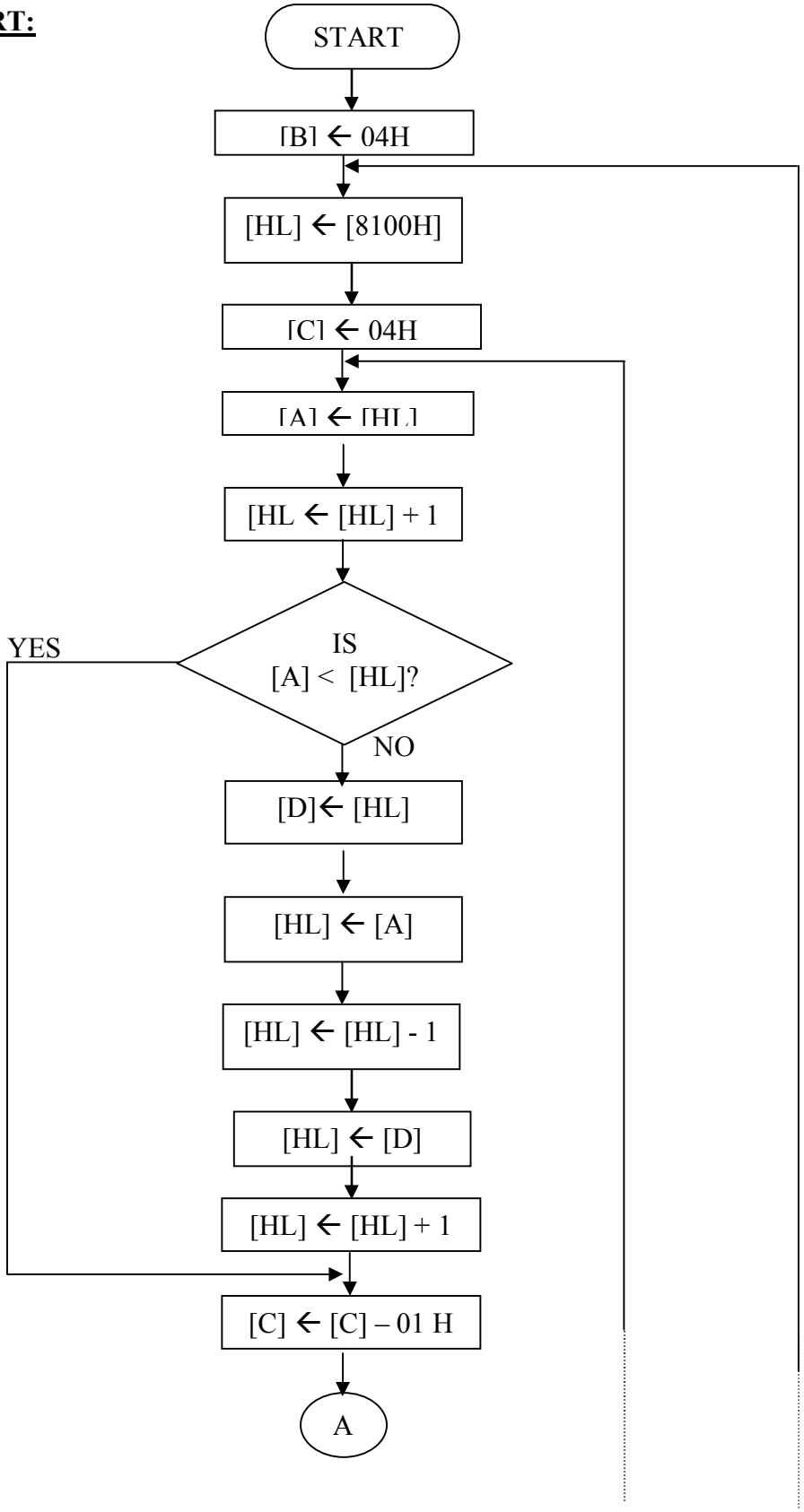
ALGORITHM:

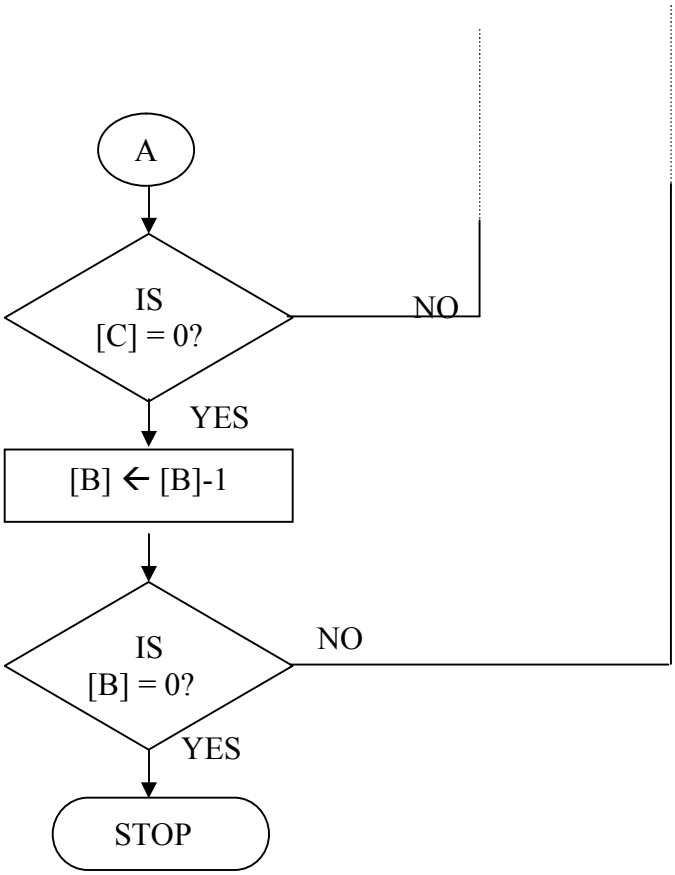
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is larger than second then I interchange the number.
3. If the first number is smaller, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

RESULT:

Thus the ascending order program is executed and thus the numbers are arranged in ascending order.

FLOWCHART:


START
[B] ← 04H
[HL] ← [8100H]
[C] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
YES
NO
[D]← [HL]
[HL] ← [A]
[HL] ← [HL] - 1
[HL] ← [D]
[HL] ← [HL] + 1
[C] ← [C] – 01 H
A


A
IS
[C] = 0?
NO
YES
[B] ← [B]-1
IS
[B] = 0?
NO
YES
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000			MVI	B,04	Initialize B reg with number of comparisons (n-1)
8001					
8002		LOOP 3	LXI	H,8100	Initialize HL reg. to 8100H
8003					
8004					
8005			MVI	C,04	Initialize C reg with no. of comparisons(n-1)
8006					
8007		LOOP2	MOV	A,M	Transfer first data to acc.
8008			INX	H	Increment HL reg. to point next memory location
8009			CMP	M	Compare M & A
800A			JC	LOOP1	If A is less than M then go to loop1
800B					
800C					
800D			MOV	D,M	Transfer data from M to D reg
800E			MOV	M,A	Transfer data from acc to M
800F			DCX	H	Decrement HL pair
8010			MOV	M,D	Transfer data from D to M
8011			INX	H	Increment HL pair
8012		LOOP1	DCR	C	Decrement C reg
8013			JNZ	LOOP2	If C is not zero go to loop2
8014					
8015					
8016			DCR	B	Decrement B reg
8017			JNZ	LOOP3	If B is not Zero go to loop3
8018					
8019					
801A			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
8100		8100	
8101		8101	
8102		8102	
8103		8103	
8104		8104	

7(B). DESCENDING ORDER

AIM:

To sort the given number in the descending order using 8085 microprocessor.

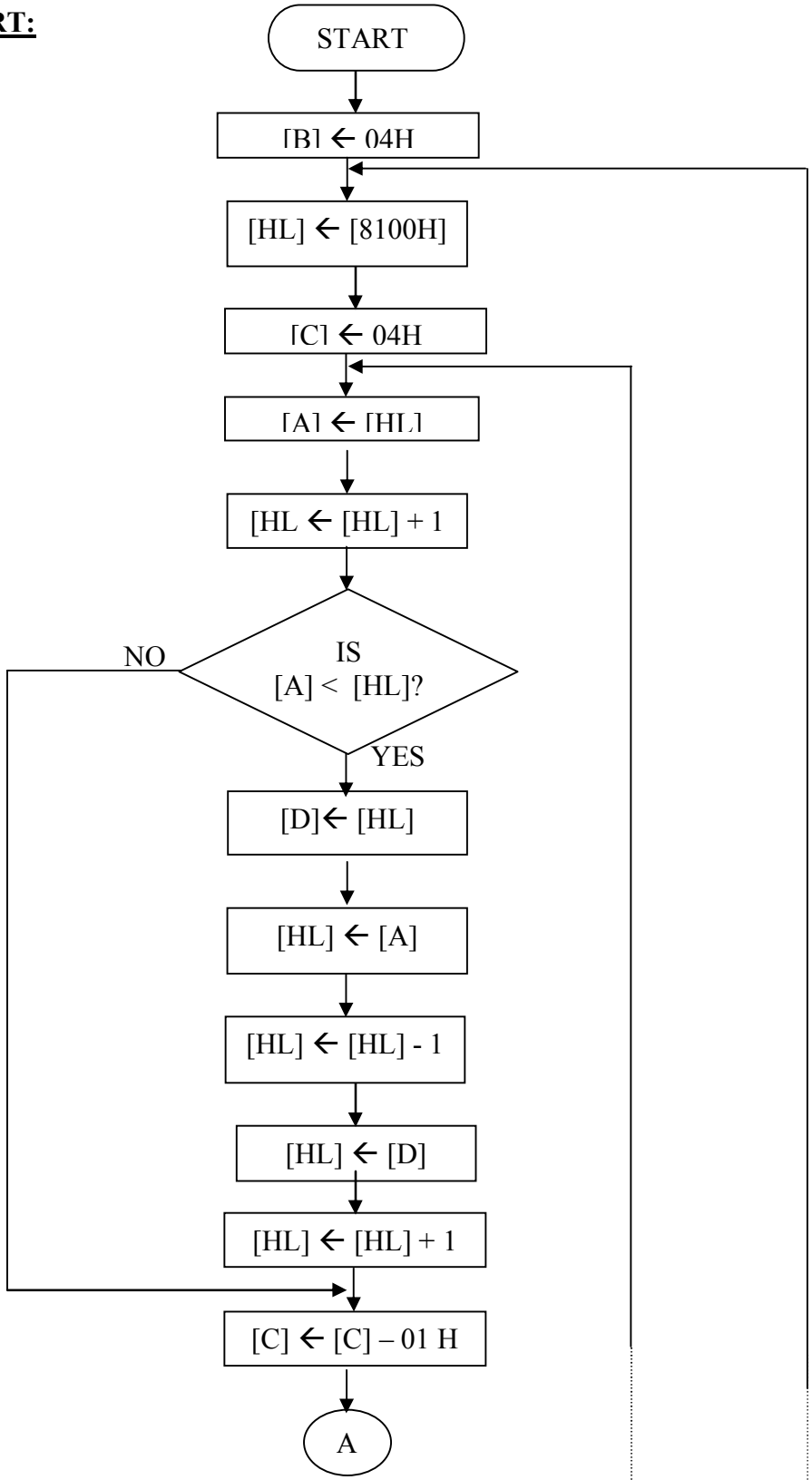
ALGORITHM:

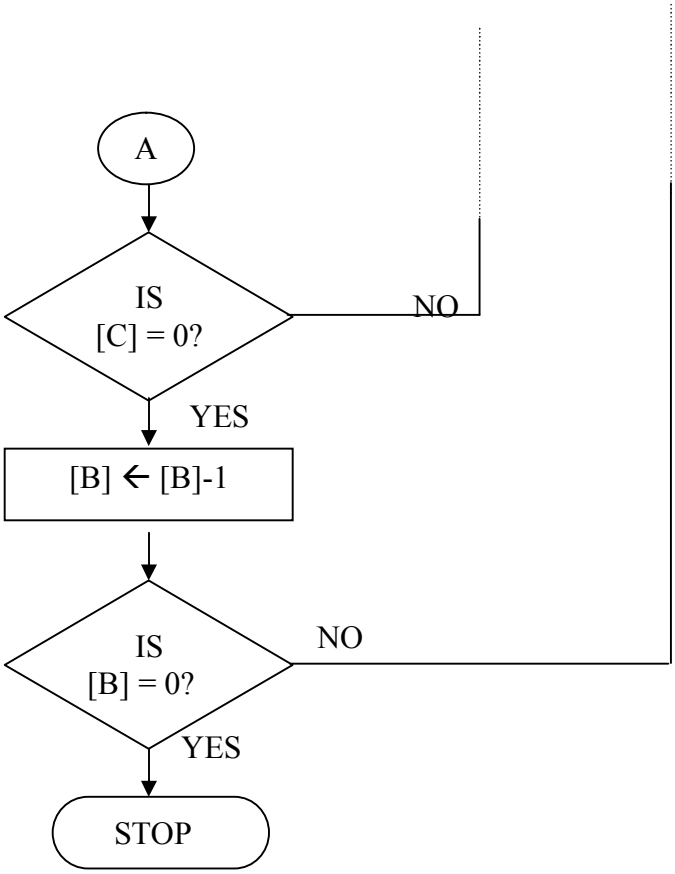
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is smaller than second then I interchange the number.
3. If the first number is larger, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

RESULT:

Thus the descending order program is executed and thus the numbers are arranged in descending order.

FLOWCHART:


START
[B] ← 04H
[HL] ← [8100H]
[C] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
NO
YES
[D]← [HL]
[HL] ← [A]
[HL] ← [HL] - 1
[HL] ← [D]
[HL] ← [HL] + 1
[C] ← [C] – 01 H
A


A
IS
[C] = 0?
NO
YES
[B] ← [B]-1
IS
[B] = 0?
NO
YES
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000			MVI	B,04	Initialize B reg with number of comparisons (n-1)
8001					
8002		LOOP 3	LXI	H,8100	Initialize HL reg. to 8100H
8003					
8004					
8005			MVI	C,04	Initialize C reg with no. of comparisons(n-1)
8006					
8007		LOOP2	MOV	A,M	Transfer first data to acc.
8008			INX	H	Increment HL reg. to point next memory location
8009			CMP	M	Compare M & A
800A			JNC	LOOP1	If A is greater than M then go to loop1
800B					
800C					
800D			MOV	D,M	Transfer data from M to D reg
800E			MOV	M,A	Transfer data from acc to M
800F			DCX	H	Decrement HL pair
8010			MOV	M,D	Transfer data from D to M
8011			INX	H	Increment HL pair
8012		LOOP1	DCR	C	Decrement C reg
8013			JNZ	LOOP2	If C is not zero go to loop2
8014					
8015					
8016			DCR	B	Decrement B reg
8017			JNZ	LOOP3	If B is not Zero go to loop3
8018					
8019					
801A			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
8100		8100	
8101		8101	
8102		8102	
8103		8103	
8104		8104	

8(A). CODE CONVERSION –DECIMAL TO HEX

AIM:

To convert a given decimal number to hexadecimal.

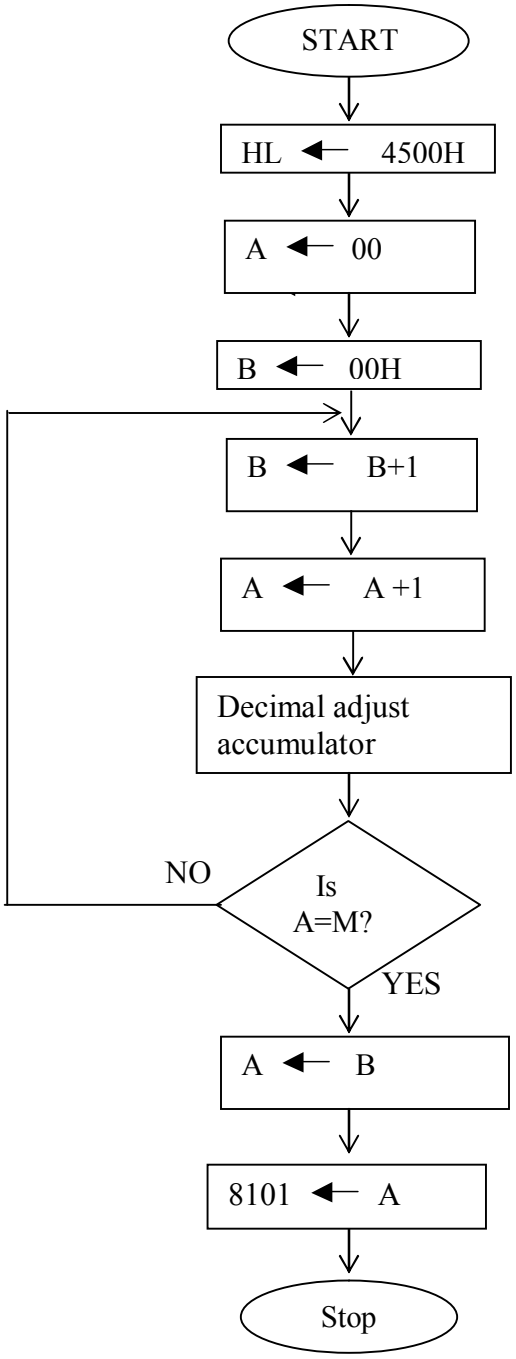
ALGORITHM:

1. Initialize the memory location to the data pointer.
2. Increment B register.
3. Increment accumulator by 1 and adjust it to decimal every time.
4. Compare the given decimal number with accumulator value.
5. When both matches, the equivalent hexadecimal value is in B register.
6. Store the resultant in memory location.

RESULT:

Thus an ALP program for conversion of decimal to hexadecimal was written and executed.

FLOWCHART:


START
HL ← 4500H
A ← 00
B ← 00H
B ← B+1
A ← A +1
Decimal adjust accumulator
Is A=M?
NO
YES
A ← B
8101 ← A
Stop

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000			LXI	H,8100	Initialize HL reg. to 8100H
8001					
8002					
8003			MVI	A,00	Initialize A register.
8004					
8005			MVI	B,00	Initialize B register..
8006					
8007		LOOP	INR	B	Increment B reg.
8008			ADI	01	Increment A reg
8009					
800A			DAA		Decimal Adjust Accumulator
800B			CMP	M	Compare M & A
800C			JNZ	LOOP	If acc and given number are not equal, then go to LOOP
800D					
800E					
800F			MOV	A,B	Transfer B reg to acc.
8010			STA	8101	Store the result in a memory location.
8011					
8012					
8013			HLT		Stop the program

RESULT:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8101	

8(B). CODE CONVERSION –HEXADECIMAL TO DECIMAL

AIM:

To convert a given hexadecimal number to decimal.

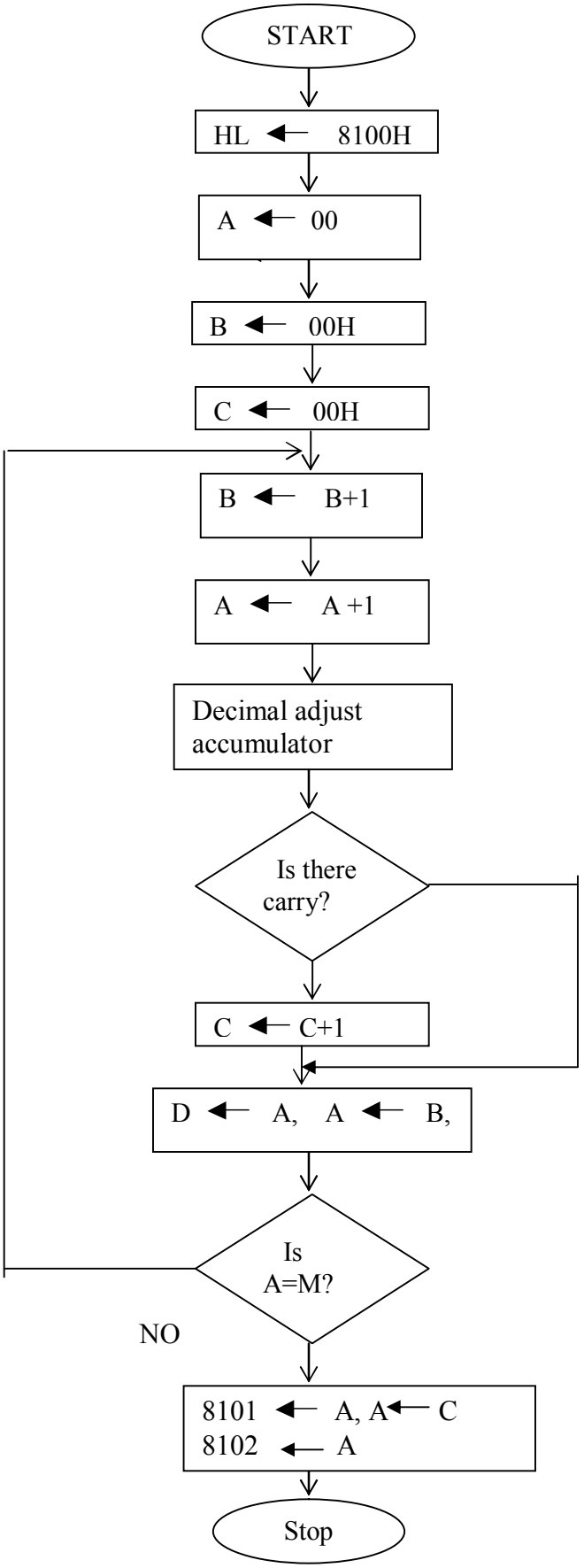
ALGORITHM:

1. Initialize the memory location to the data pointer.
2. Increment B register.
3. Increment accumulator by 1 and adjust it to decimal every time.
4. Compare the given hexadecimal number with B register value.
5. When both match, the equivalent decimal value is in A register.
6. Store the resultant in memory location.

RESULT:

Thus an ALP program for conversion of hexadecimal to decimal was written and executed.

FLOWCHART:


START
HL ← 8100H
A ← 00
B ← 00H
C ← 00H
B ← B+1
A ← A +1
Decimal adjust accumulator
Is there carry?
C ← C+1
D ← A, A ← B,
Is A=M?
NO
8101 ← A, A ← C
8102 ← A
Stop

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000			LXI	H,8100	Initialize HL reg. to 8100H
8001					
8002					
8003			MVI	A,00	Initialize A register.
8004					
8005			MVI	B,00	Initialize B register.
8006					
8007			MVI	C,00	Initialize C register for carry.
8008					
8009		LOOP	INR	B	Increment B reg.
800A			ADI	01	Increment A reg
800B					
800C			DAA		Decimal Adjust Accumulator
800D			JNC	NEXT	If there is no carry go to NEXT.
800E					
800F					
8010			INR	C	Increment c register.
8011		NEXT	MOV	D,A	Transfer A to D
8012			MOV	A,B	Transfer B to A
8013			CMP	M	Compare M & A
8014			MOV	A,D	Transfer D to A
8015			JNZ	LOOP	If acc and given number are not equal, then go to LOOP
8016					
8017					
8018			STA	8101	Store the result in a memory location.
8019					
801A					
801B			MOV	A,C	Transfer C to A
801C			STA	8102	Store the carry in another memory location.
801D					
801E					
801F			HLT		Stop the program

RESULT:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8101	
		8102	

9(A) BCD ADDITION

AIM:

To add two 8 bit BCD numbers stored at consecutive memory locations.

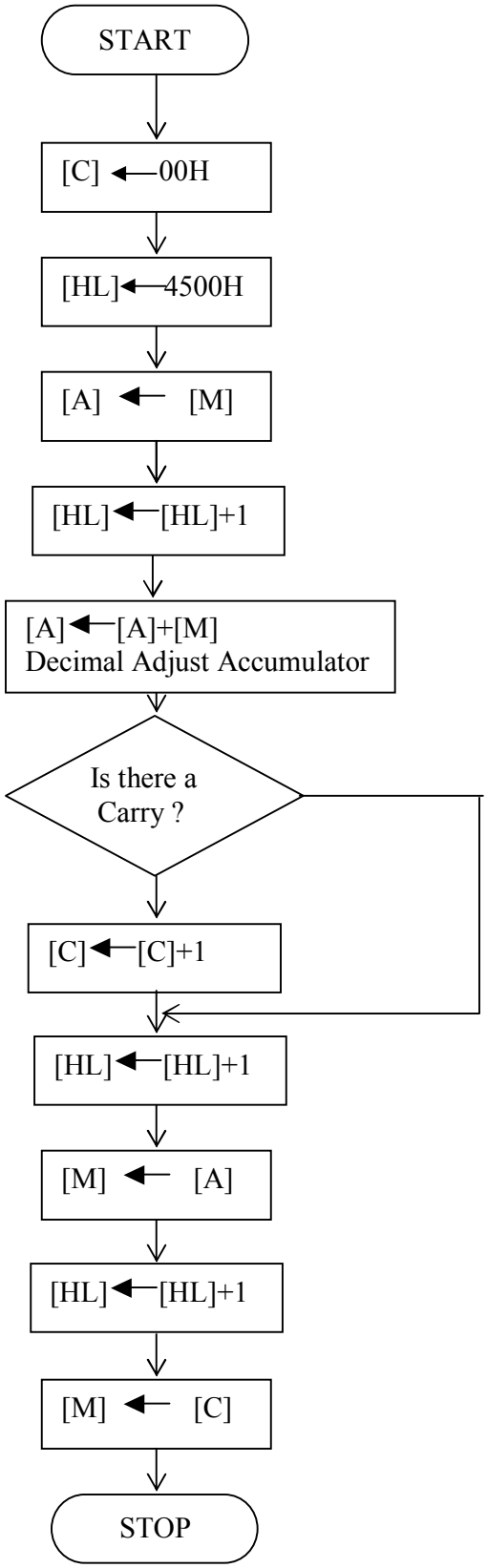
ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and add it to the accumulator
4. Adjust the accumulator value to the proper BCD value using DAA instruction.
5. Store the answer at another memory location.

RESULT:

Thus the 8 bit BCD numbers stored at 4500 &4501 are added and the result stored at 4502 & 4503.

FLOW CHART:


START
[C] ←00H
[HL]←4500H
[A] ← [M]
[HL]←[HL]+1
[A]←[A]+[M]
Decimal Adjust Accumulator
Is there a Carry ?
[C]←[C]+1
[HL]←[HL]+1
[M] ← [A]
[HL]←[HL]+1
[M] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	C, 00	Clear C reg.
4103					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	A, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next memory Location.
4107			ADD	M	Add first number to acc. Content.
4108			DAA		Decimal adjust accumulator
4109			JNC	L1	Jump to location if result does not yield carry.
410A					
410B					
410C			INR	C	Increment C reg.
410D		L1	INX	H	Increment HL reg. to point next memory Location.
410E			MOV	M, A	Transfer the result from acc. to memory.
410F			INX	H	Increment HL reg. to point next memory Location.
4110			MOV	M, C	Move carry to memory
4111			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

9(B). BCD SUBTRACTION

AIM:

To Subtract two 8 bit BCD numbers stored at consecutive memory locations.

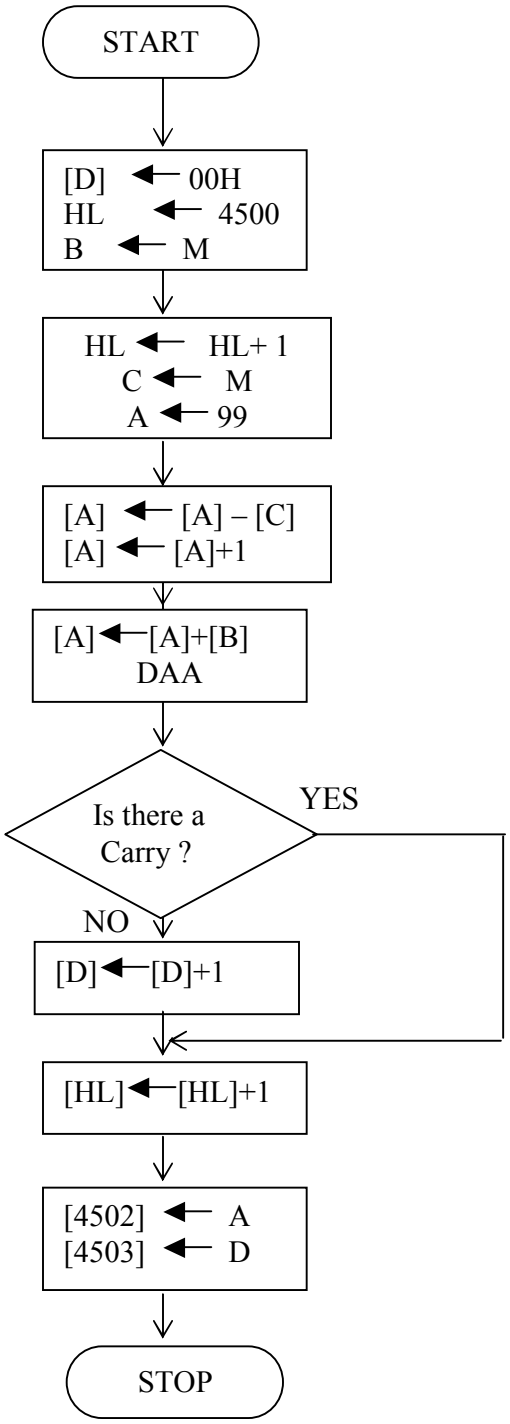
ALGORITHM:

1. Load the minuend and subtrahend in two registers.
2. Initialize Borrow register to 0.
3. Take the 100's complement of the subtrahend.
4. Add the result with the minuend which yields the result.
5. Adjust the accumulator value to the proper BCD value using DAA instruction. If there is a carry ignore it.
6. If there is no carry, increment the carry register by 1
7. Store the content of the accumulator (result)and borrow register in the specified memory location

RESULT:

Thus the 8 bit BCD numbers stored at 4500 &4501 are subtracted and the result stored at 4502 & 4503.

FLOW CHART:


START
[D] ← 00H
HL ← 4500
B ← M
HL ← HL+ 1
C ← M
A ← 99
[A] ← [A] – [C]
[A] ← [A]+1
[A] ← [A]+[B]
DAA
Is there a Carry ?
YES
NO
[D] ← [D]+1
[HL] ← [HL]+1
[4502] ← A
[4503] ← D
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4100		START	MVI	D, 00	Clear D reg.
4101					
4102			LXI	H, 4500	Initialize HL reg. to 4500
4103					
4104					
4105			MOV	B, M	Transfer first data to accumulator
4106			INX	H	Increment HL reg. to point next mem. Location.
4107			MOV	C, M	Move second no. to B reg.
4108			MVI	A, 99	Move 99 to the Accumulator
4109					
410A			SUB	C	Subtract [C] from acc. Content.
410B			INR	A	Increment A register
410C			ADD	B	Add [B] with [A]
410D			DAA		Adjust Accumulator value for Decimal digits
410E			JC	LOOP	Jump on carry to loop
410F					
4110					
4111			INR	D	Increment D reg.
4112		LOOP	INX	H	Increment HL register pair
4113			MOV	M , A	Move the Acc.content to the memory location
4114			INX	H	Increment HL reg. to point next mem. Location.
4115			MOV	M, D	Transfer D register content to memory.
4116			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

10. 2 X 2 MATRIX MULTIPLICATION

AIM:

To perform the 2 x 2 matrix multiplication.

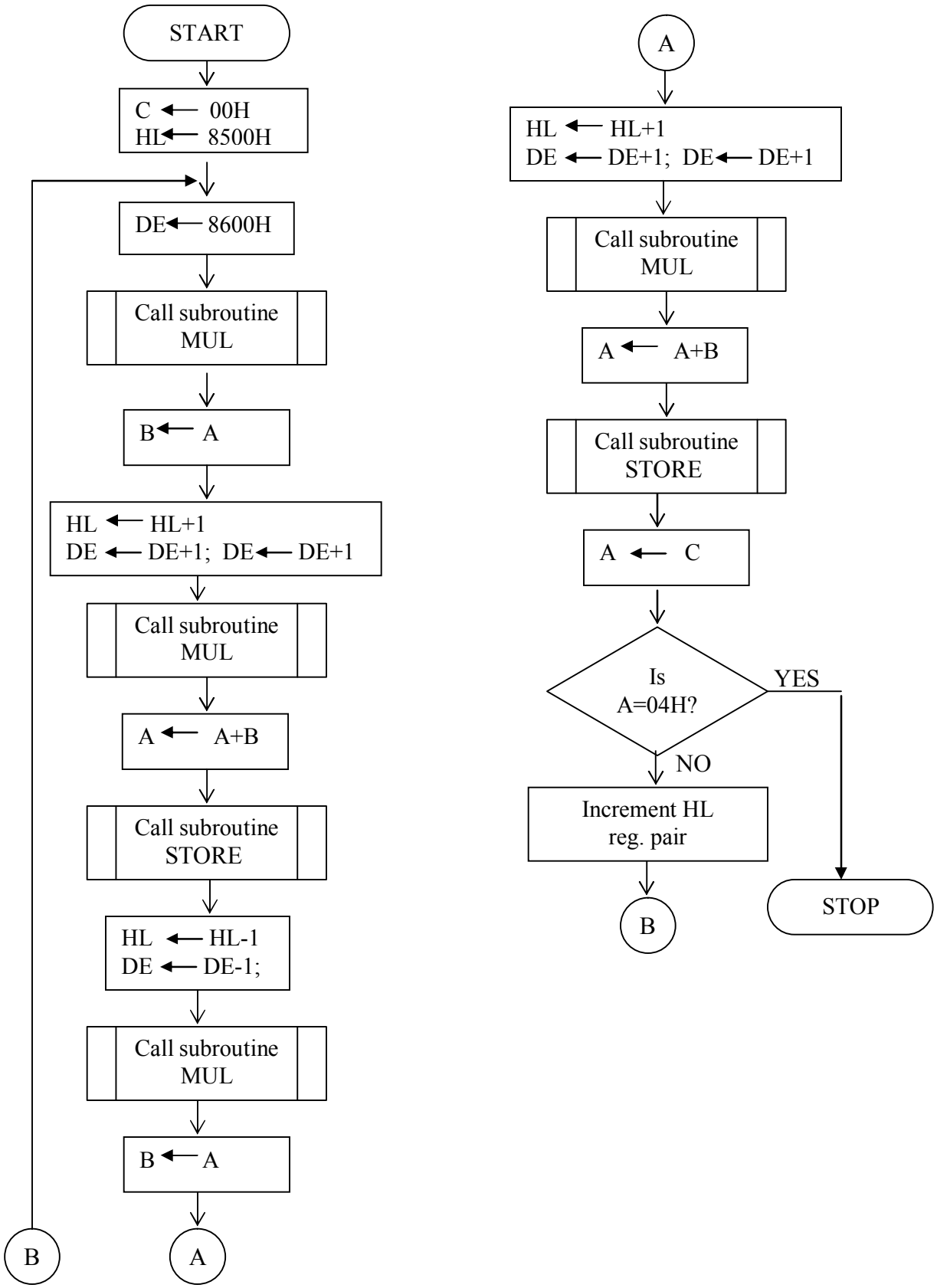
ALGORITHM:

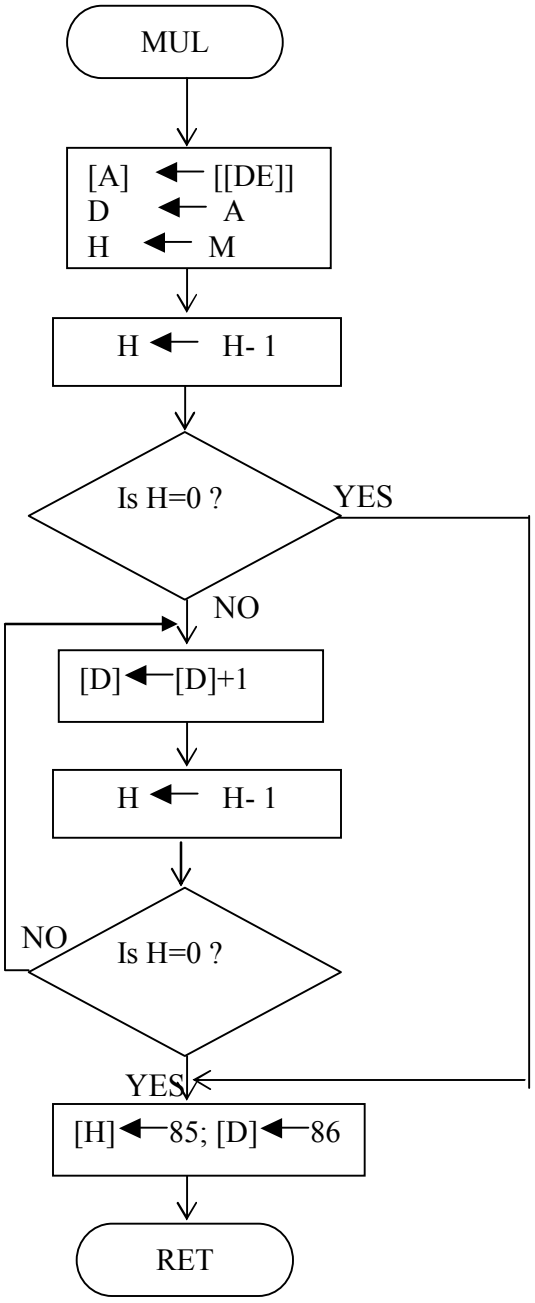
1. Load the 2 input matrices in the separate address and initialize the HL and the DE register pair with the starting address respectively.
2. Call a subroutine for performing the multiplication of one element of a matrix with the other element of the other matrix.
3. Call a subroutine to store the resultant values in a separate matrix.

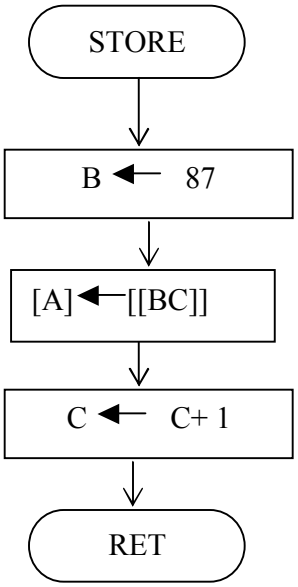
RESULT:

Thus the 2 x 2 matrix multiplication is performed and the result is stored at 4700,4701 , 4702 & 4703.

FLOW CHART:


START
C ← 00H
HL ← 8500H
DE ← 8600H
Call subroutine MUL
B ← A
HL ← HL+1
DE ← DE+1; DE ← DE+1
Call subroutine MUL
A ← A+B
Call subroutine STORE
HL ← HL-1
DE ← DE-1;
Call subroutine MUL
B ← A
B
A
A
HL ← HL+1
DE ← DE+1; DE ← DE+1
Call subroutine MUL
A ← A+B
Call subroutine STORE
A ← C
Is A=04H?
YES
NO
Increment HL reg. pair
B
STOP


MUL
[A] ← [[DE]]
D ← A
H ← M
H ← H- 1
Is H=0 ?
YES
NO
[D] ← [D]+1
H ← H- 1
Is H=0 ?
NO
YES
[H] ← 85; [D] ← 86
RET


STORE
B ← 87
[A] ← [[BC]]
C ← C+ 1
RET

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
8100			MVI	C, 00	Clear C reg.
8101					
8102			LXI	H, 8500	Initialize HL reg. to 4500
8103					
8104					
8105		LOOP2	LXI	D, 8600	Load DE register pair
8106					
8107					
8108			CALL	MUL	Call subroutine MUL
8109					
810A					
810B			MOV	B,A	Move A to B reg.
810C			INX	H	Increment HL register pair .
810D			INX	D	Increment DE register pair
810E			INX	D	Increment DE register pair
810F			CALL	MUL	Call subroutine MUL
8110					
8111					
8112			ADD	B	Add [B] with [A]
8113			CALL	STORE	Call subroutine STORE
8114					
8115					
8116			DCX	H	Decrement HL register pair
8117			DCX	D	Decrement DE register pair
8118			CALL	MUL	Call subroutine MUL
8119					
811A					
811B			MOV	B,A	Transfer A reg content to B reg.
811C			INX	H	Increment HL register pair
811D			INX	D	Increment DE register pair
811E			INX	D	Increment DE register pair
811F			CALL	MUL	Call subroutine MUL
8120					
8121					
8122			ADD	B	Add A with B
8123			CALL	STORE	Call subroutine MUL
8124					
8125					
8126			MOV	A,C	Transfer C register content to Acc.

8127			CPI	04	Compare with 04 to check whether all elements are multiplied.
8128					
8129			JZ	LOOP1	If completed, go to loop1
812A					
812B					
812C			INX	H	Increment HL register Pair.
812D			JMP	LOOP2	Jump to LOOP2.
812E					
812F					
8130		LOOP1	HLT		Stop the program.
8131		MUL	LDAX	D	Load acc from the memory location pointed by DE pair.
8132			MOV	D,A	Transfer acc content to D register.
8133			MOV	H,M	Transfer from memory to H register.
8134			DCR	H	Decrement H register.
8135			JZ	LOOP3	If H is zero go to LOOP3.
8136					
8137					
8138		LOOP4	ADD	D	Add Acc with D reg
8139			DCR	H	Decrement H register.
813A			JNZ	LOOP4	If H is not zero go to LOOP4.
813B					
813C					
813D		LOOP3	MVI	H,85	Transfer 85 TO H register.
813E					
813F			MVI	D,86	Transfer 86 to D register.
8140					
8141			RET		Return to main program.
8142		STORE	MVI	B,87	Transfer 87 to B register.
8143					
8144			STAX	B	Load A from memory location pointed by BC pair.
8145			INR	C	Increment C register.
8146			RET		Return to main program.

OBSERVATION:

INPUT				OUTPUT	
4500		4600		4700	
4501		4601		4701	
4502		4602		4702	
4503		4603		4703	

11.8086 STRING MANIPULATION – SEARCH A WORD

AIM:

To search a word from a string.

ALGORITHM:

1. Load the source and destination index register with starting and the ending address respectively.
2. Initialize the counter with the total number of words to be copied.
3. Clear the direction flag for auto incrementing mode of transfer.
4. Use the string manipulation instruction SCASW with the prefix REP to search a word from string.
5. If a match is found (z=1), display 01 in destination address. Otherwise, display 00 in destination address.

RESULT:

A word is searched and the count of number of appearances is displayed.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DW 53H, 15H, 19H, 02H
DEST EQU 3000H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        MOV AX, 15H
        MOV SI, OFFSET LIST
        MOV DI, DEST
        MOV CX, COUNT
        MOV AX, 00
        CLD
REP     SCASW
        JZ LOOP
        MOV AX, 01
LOOP    MOV [DI], AX
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

INPUT:

LIST: 53H, 15H, 19H, 02H

OUTPUT:

3000 01

12.8086 STRING MANIPULATION –FIND AND REPLACE A WORD

AIM:

To find and replace a word from a string.

ALGORITHM:

1. Load the source and destination index register with starting and the ending address respectively.
2. Initialize the counter with the total number of words to be copied.
3. Clear the direction flag for auto incrementing mode of transfer.
4. Use the string manipulation instruction SCASW with the prefix REP to search a word from string.
5. If a match is found (z=1), replace the old word with the current word in destination address. Otherwise, stop.

RESULT:

A word is found and replaced from a string.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DW 53H, 15H, 19H, 02H
REPLACE EQU 30H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        MOV AX, 15H
        MOV SI, OFFSET LIST
        MOV CX, COUNT
        MOV AX, 00
        CLD
REP     SCASW
        JNZ LOOP
        MOV DI, LABEL LIST
        MOV [DI], REPLACE
LOOP    MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

INPUT:

LIST: 53H, 15H, 19H, 02H

OUTPUT:

LIST: 53H, 30H, 19H, 02H

13. 8086 STRING MANIPULATION – COPY A STRING

AIM:

To copy a string of data words from one location to the other.

ALGORITHM:

6. Load the source and destination index register with starting and the ending address respectively.
7. Initialize the counter with the total number of words to be copied.
8. Clear the direction flag for auto incrementing mode of transfer.
9. Use the string manipulation instruction MOVSW with the prefix REP to copy a string from source to destination.

RESULT:

A string of data words is copied from one location to other.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
SOURCE EQU 2000H
DEST EQU 3000H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV ES, AX
            MOV SI, SOURCE
            MOV DI, DEST
            MOV CX, COUNT
            CLD
REP   MOVSW
            MOV AH, 4CH
            INT 21H
CODE ENDS
END START
```

INPUT:		**OUTPUT:**	
2000	48	3000	48
2001	84	3001	84
2002	67	3002	67
2003	90	3003	90
2004	21	3004	21

14.8086 STRING MANIPULATION – SORTING

AIM:

To sort a group of data bytes.

ALGORITHM:

- Place all the elements of an array named list (in the consecutive memory locations).
- Initialize two counters DX & CX with the total number of elements in the array.
- Do the following steps until the counter B reaches 0.
    - Load the first element in the accumulator
    - Do the following steps until the counter C reaches 0.
        1. Compare the accumulator content with the next element present in the next memory location. If the accumulator content is smaller go to next step; otherwise, swap the content of accumulator with the content of memory location.
        2. Increment the memory pointer to point to the next element.
        3. Decrement the counter C by 1.
- Stop the execution.

RESULT:

A group of data bytes are arranged in ascending order.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DW 53H, 25H, 19H, 02H
COUNT EQU 04H
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, COUNT-1
LOOP2:  MOV CX, DX
        MOV SI, OFFSET LIST
AGAIN:  MOV AX, [SI]
        CMP AX, [SI+2]
        JC LOOP1
        XCHG [SI +2], AX
        XCHG [SI], AX
LOOP1:  ADD SI, 02
        LOOP AGAIN
        DEC DX
        JNZ LOOP2
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

INPUT:

LIST: 53H, 25H, 19H, 02H

OUTPUT:

LIST: 02H, 19H, 25H, 53H