Department of Electronics and Communication Engineering

Sub Code/Name: **BEC5L6 -MICROPROCESSOR AND MICROCONTROLLER LAB**

Name : ……………………………………

Reg No : ……………………………………

Branch : ……………………………………

Year & Semester : ……………………………………

LIST OF EXPERIMENTS

INDEX

Expt.	Date	Name of the Experiment	Marks	Staff SIGN

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:

- <u>Sign flag</u>

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.

- <u>Zero flag</u>

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.

- <u>Auxillary carry flag</u>

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

- <u>Parity flag</u>

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.

- <u>Carry flag</u>

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.

Ex No:1(A)

Date:

8 BIT DATA ADDITION

AIM:

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

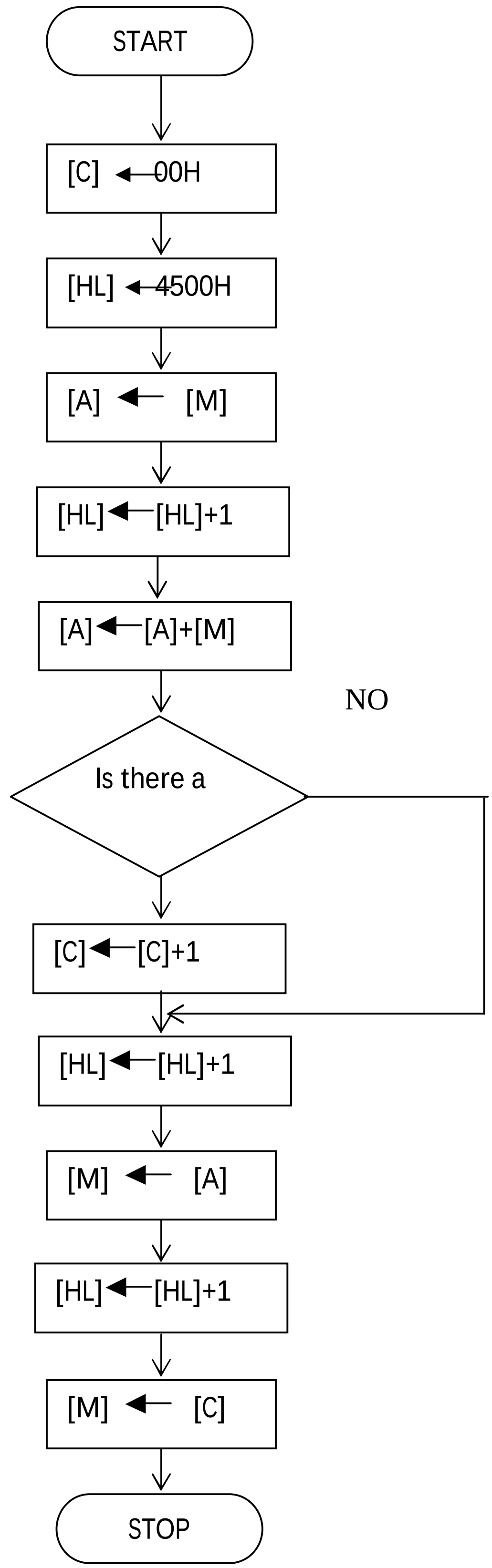
ALGORITHM:

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.

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

FLOW CHART:


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

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

RESULT:

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

Ex No:1(B)

Date:

8 BIT DATA SUBTRACTION

AIM:

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

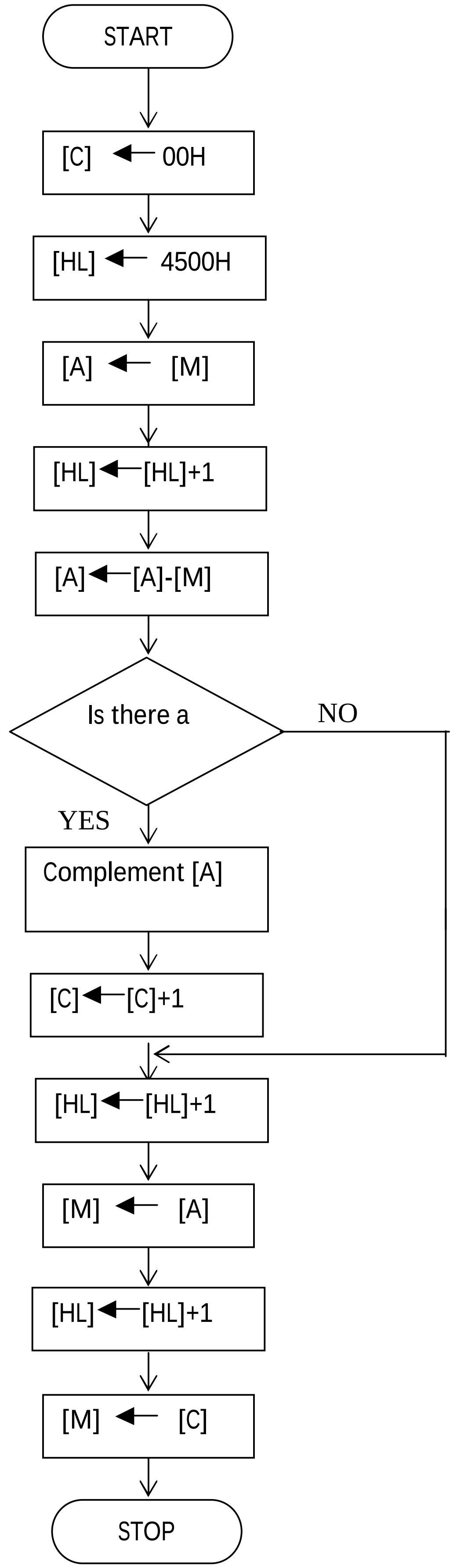
ALGORITHM:

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.

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 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

FLOW CHART:


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

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

RESULT:

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

Ex No:2(A)

Date:

16 BIT DATA ADDITION

AIM:

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

ALGORITHM:

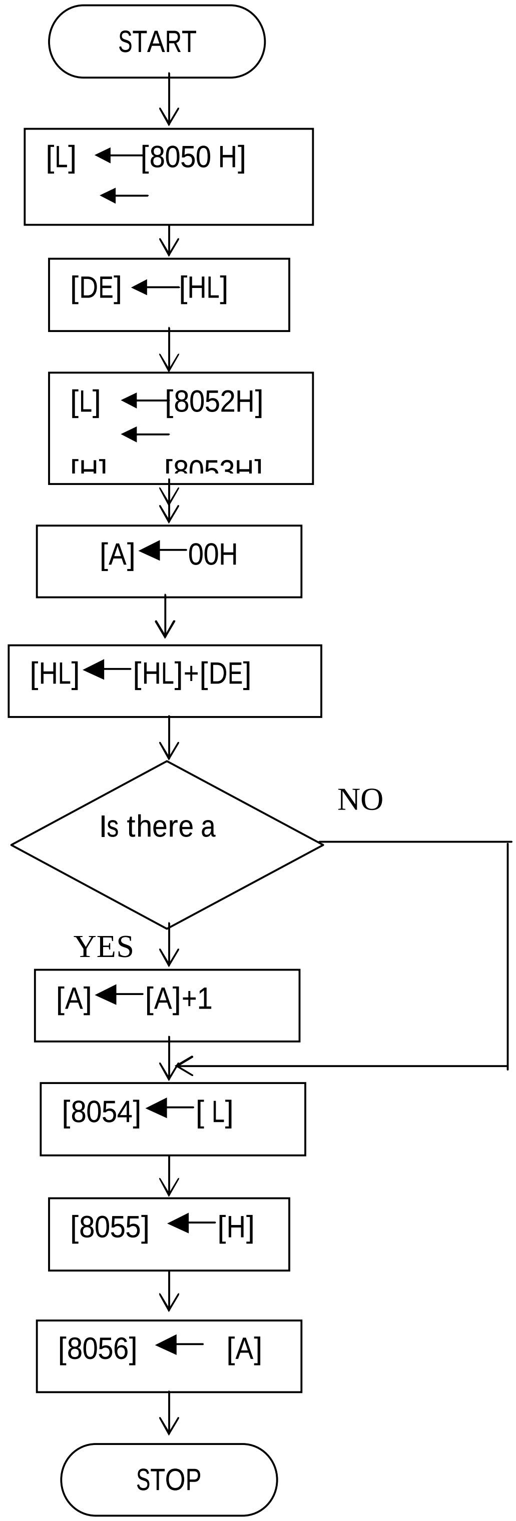
1. Initialize memory pointer to data location.
2. Get the first number from memory and store in Register pair.
3. Get the second number in memory and add it to the Register pair.
4. Store the sum & carry in separate memory locations.

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.

.

FLOW CHART:


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

OBSERVATION:

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

RESULT:

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

Ex No:2(B)

Date:

16 BIT DATA SUBTRACTION

AIM:

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

ALGORITHM:

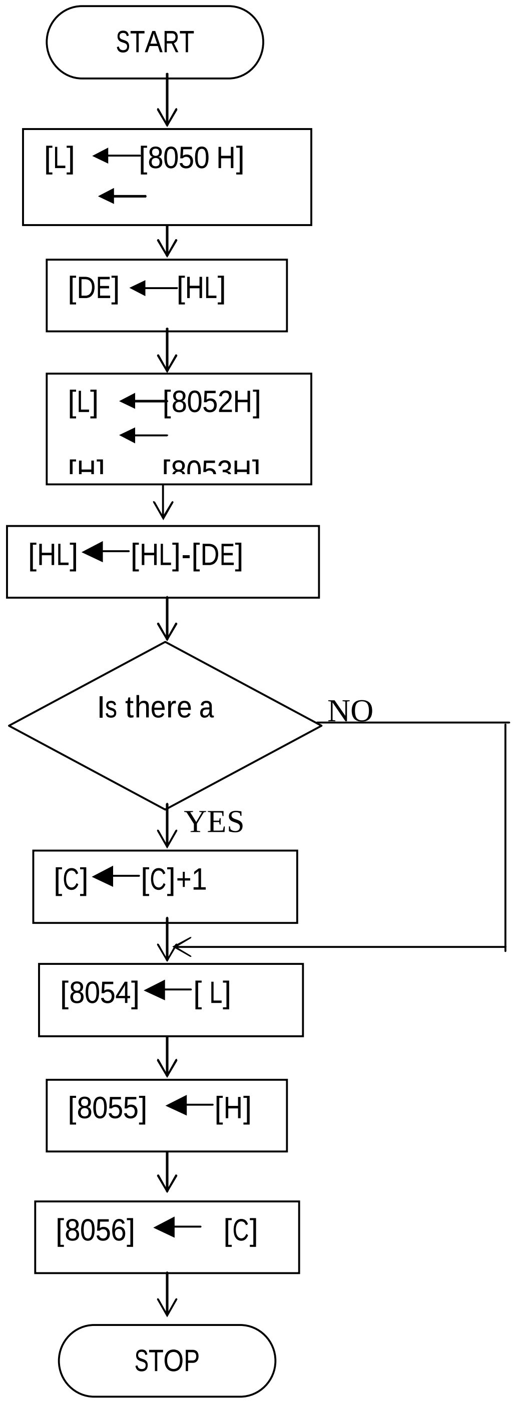
1. Initialize memory pointer to data location.
2. Get the subtrahend from memory and transfer it to register pair.
3. Get the minuend from memory and store it in another register pair.
4. Subtract subtrahend from minuend.
5. Store the difference and borrow in different memory locations.

. 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

8018					the memory location 8506H
8019					
801A			HLT		Stop the program execution.

FLOW CHART:



OBSERVATION:

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

RESULT:

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

Ex No:3(A)

Date:

ASCENDING ORDER

AIM:

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

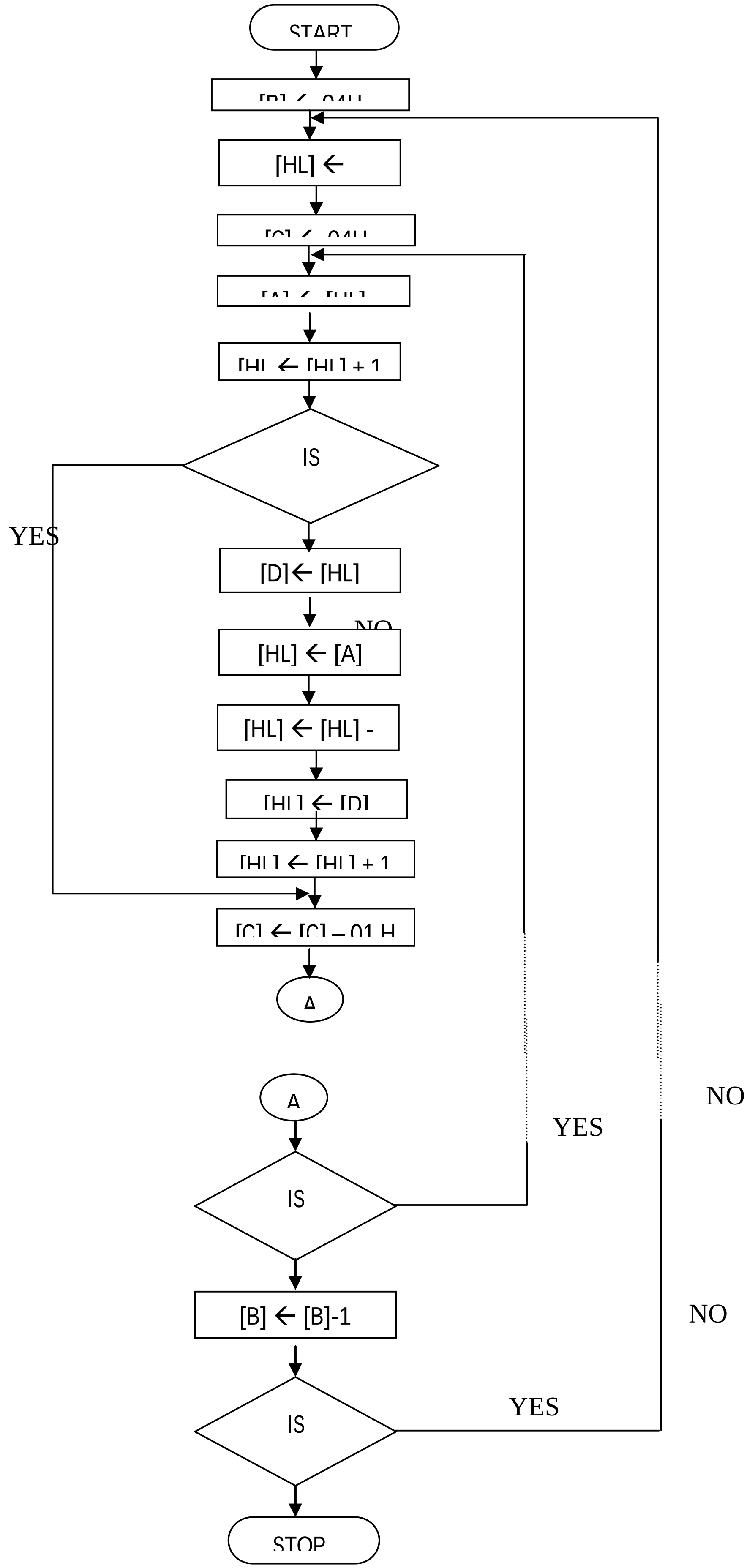
ALGORITHM:

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

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

FLOWCHART:



START
[HL] ←
IS
YES
[D]← [HL]
NO
[HL] ← [A]
[HL] ← [HL] -
[HL] ← [D]
[HL] ← [HL] + 1
[C] ← [C] – 01 H
A
A
IS
YES
NO
[B] ← [B]-1
NO
IS
YES
STOP

OBSERVATION:

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

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

Ex No:3(A)

Date:

DESCENDING ORDER

AIM:

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

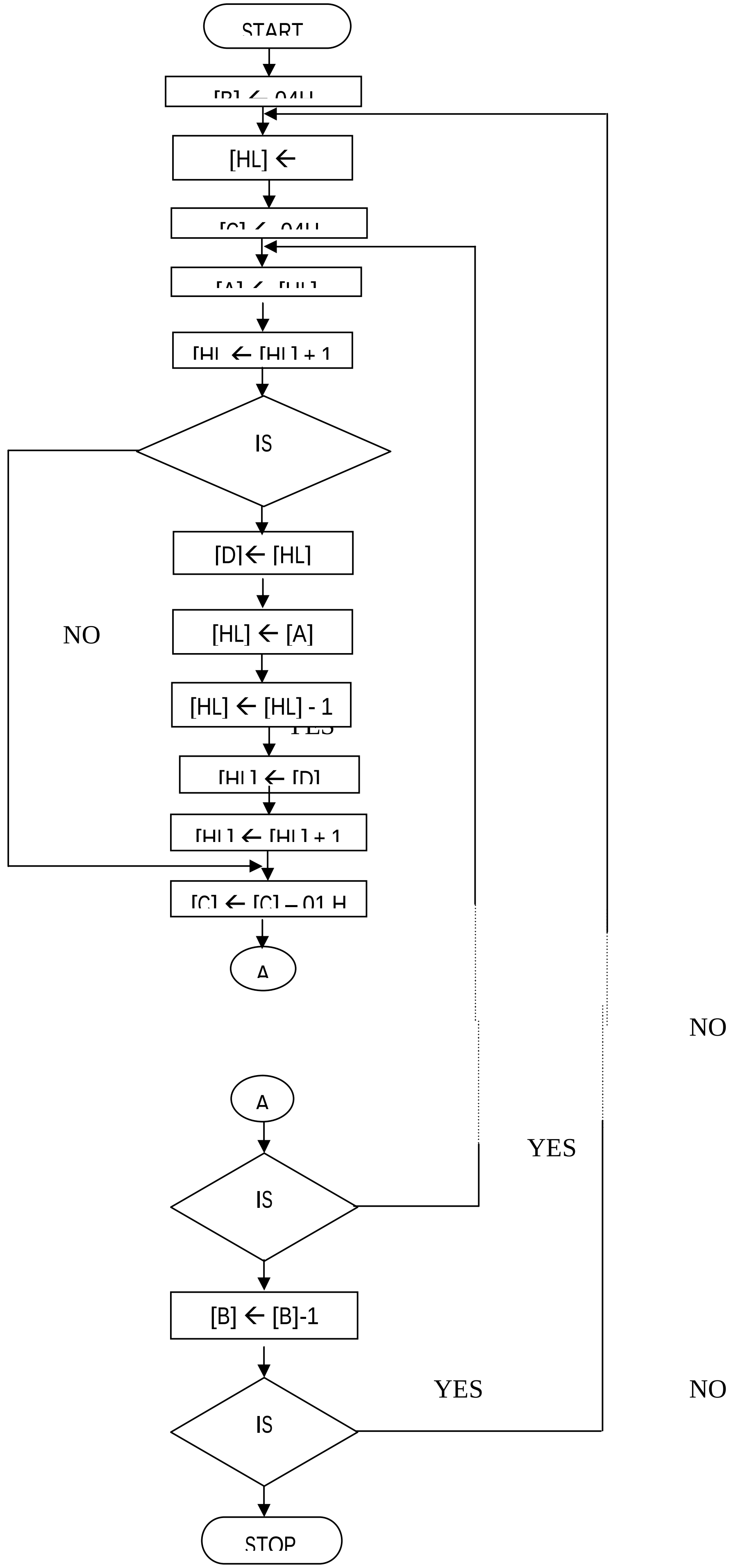
ALGORITHM:

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

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

FLOWCHART:



OBSERVATION:

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

RESULT:

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

Ex No:4(A)

Date:

6(A). LARGEST ELEMENT IN AN ARRAY

AIM:

To find the largest element in an array.

ALGORITHM:

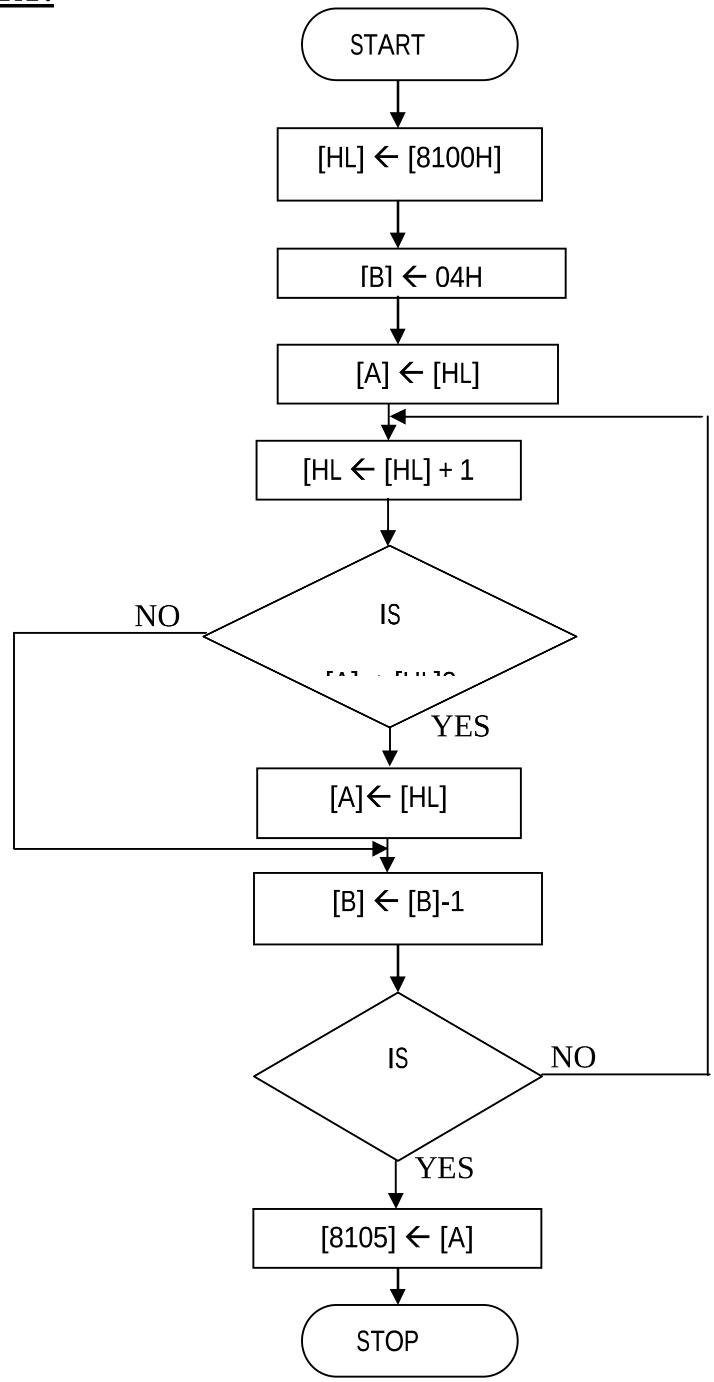
1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer 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.

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

8012					location.
8013					
8014			HLT		Stop the program

FLOW CHART:



OBSERVATION:

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

RESULT:

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

Ex No:4(B)

Date:

SMALLEST ELEMENT IN AN ARRAY

AIM:

To find the smallest element in an array.

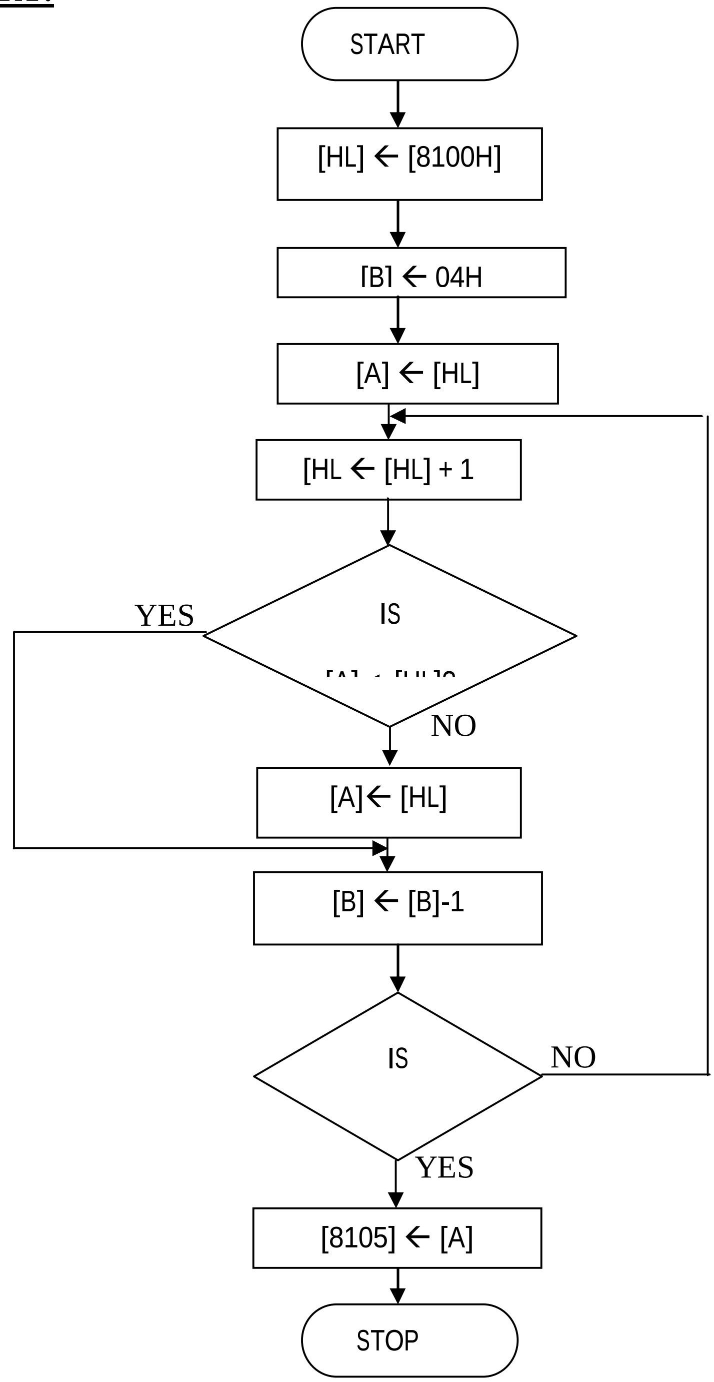
ALGORITHM:

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.

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

FLOW CHART:


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

OBSERVATION:

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

RESULT:

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

Ex No:5(A)

Date:

CODE CONVERSION –DECIMAL TO HEX

AIM:

To convert a given decimal number to hexadecimal.

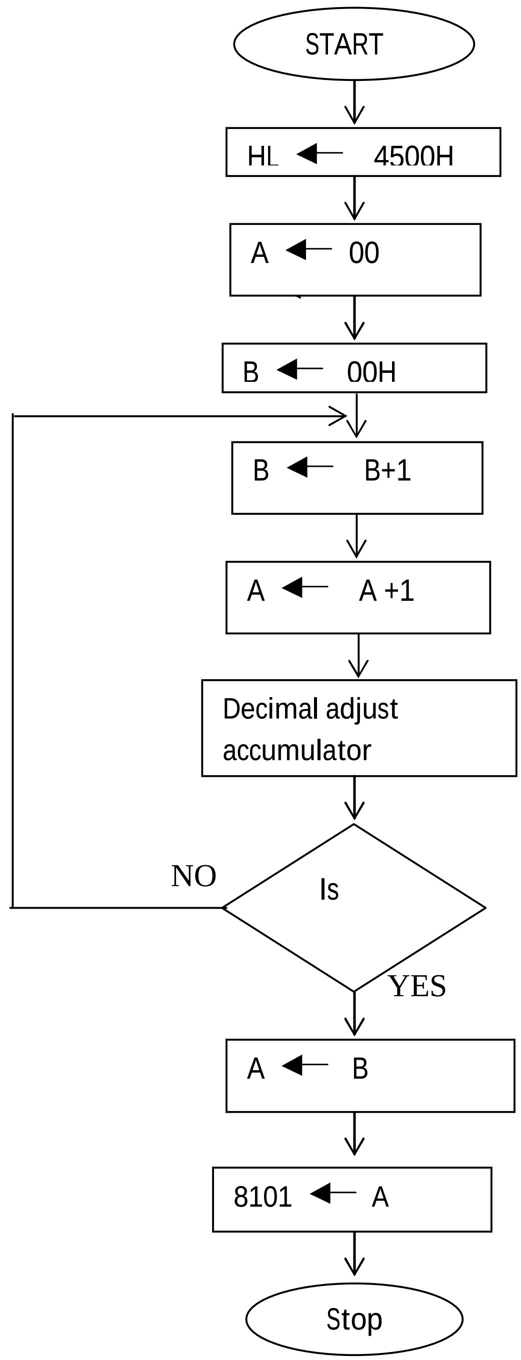
ALGORITHM:

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.

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

FLOWCHART:


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

RESULT:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8101	

RESULT:

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

Ex No:5(B)

Date:

CODE CONVERSION –HEXADECIMAL TO DECIMAL

AIM:

To convert a given hexadecimal number to decimal.

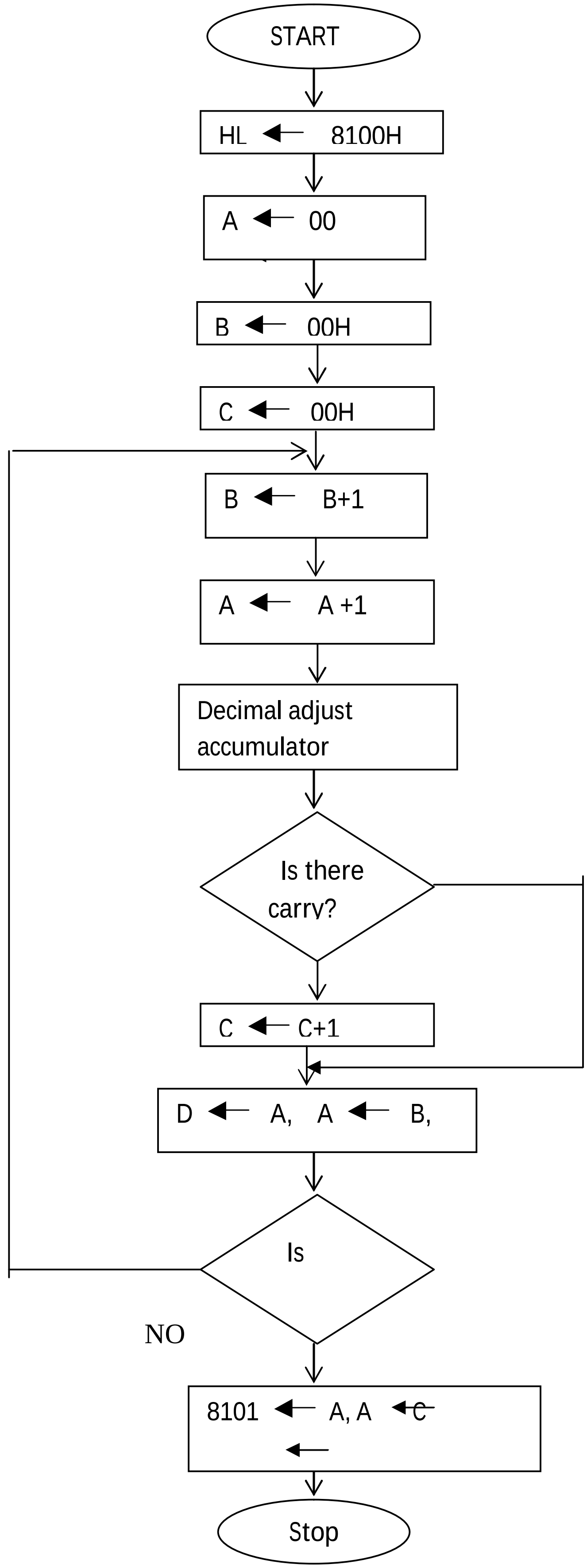
ALGORITHM:

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.

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

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
NO
8101 ← A, A ← C
Stop

RESULT:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
8100		8101	
		8102	

RESULT:

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

Ex No: 5(C)

Date:

BINARY ARITHMETIC-BCD ADDITION

AIM:

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

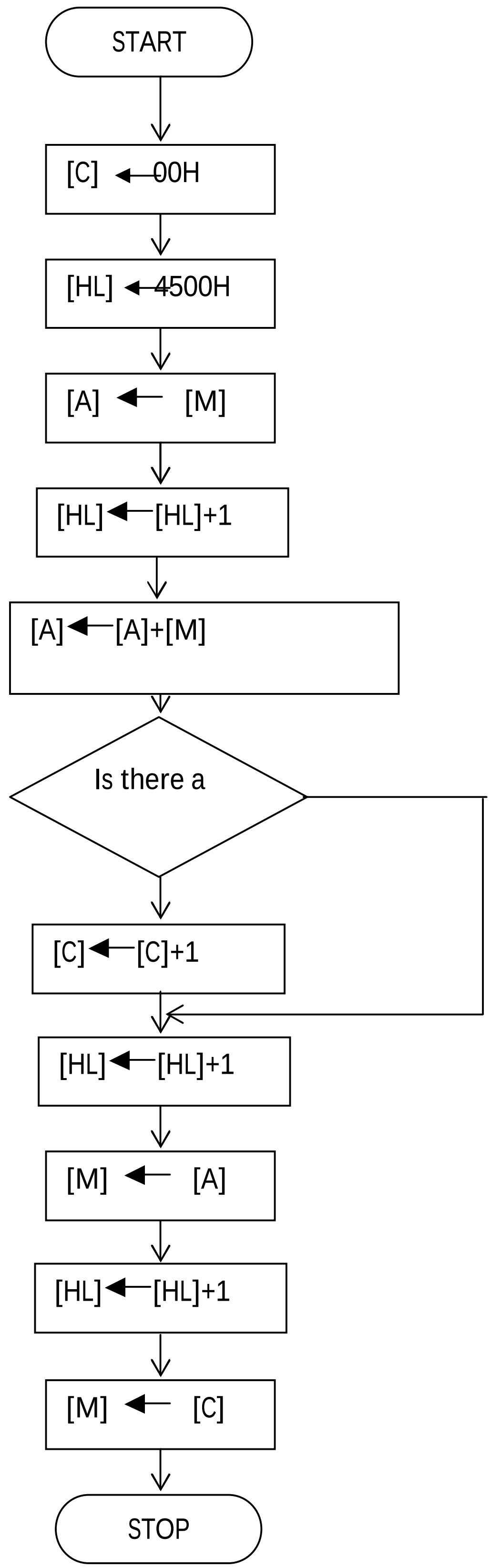
ALGORITHM:

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.

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			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

FLOW CHART:



START
[C] ← 00H
[HL] ← 4500H
[A] ← [M]
[HL] ← [HL]+1
[A] ← [A]+[M]
Is there a
[C] ← [C]+1
[HL] ← [HL]+1
[M] ← [A]
[HL] ← [HL]+1
[M] ← [C]
STOP

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

RESULT:

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

Ex No: 5(D)

Date:

BCD SUBTRACTION

AIM:

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

ALGORITHM:

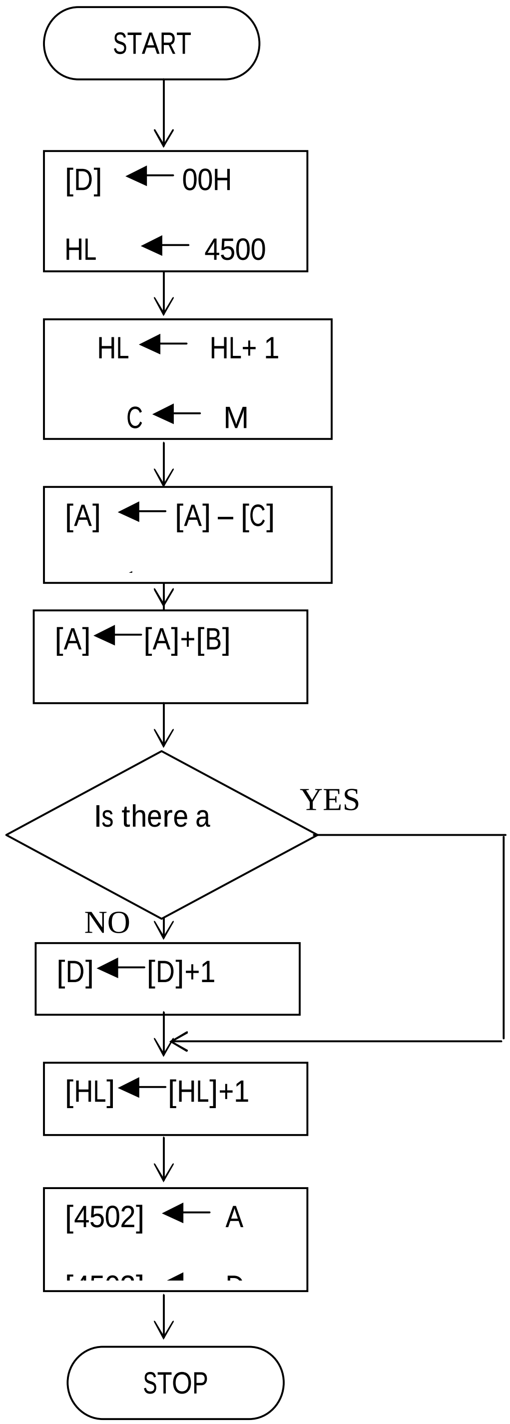
1. Load the minuend and subtrahend in two registers.
2. Initialize Borrow register to 0.
3. Take the 100's complement of the subtrahend.
4. Add the result with the minuend which yields the result.
5. Adjust the accumulator value to the proper BCD value using DAA instruction. If there is a carry ignore 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

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

FLOW CHART:



OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

RESULT:

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

Ex No: 6(A)

Date:

2 X 2 MATRIX MULTIPLICATION

AIM:

To perform the 2 x 2 matrix multiplication.

ALGORITHM:

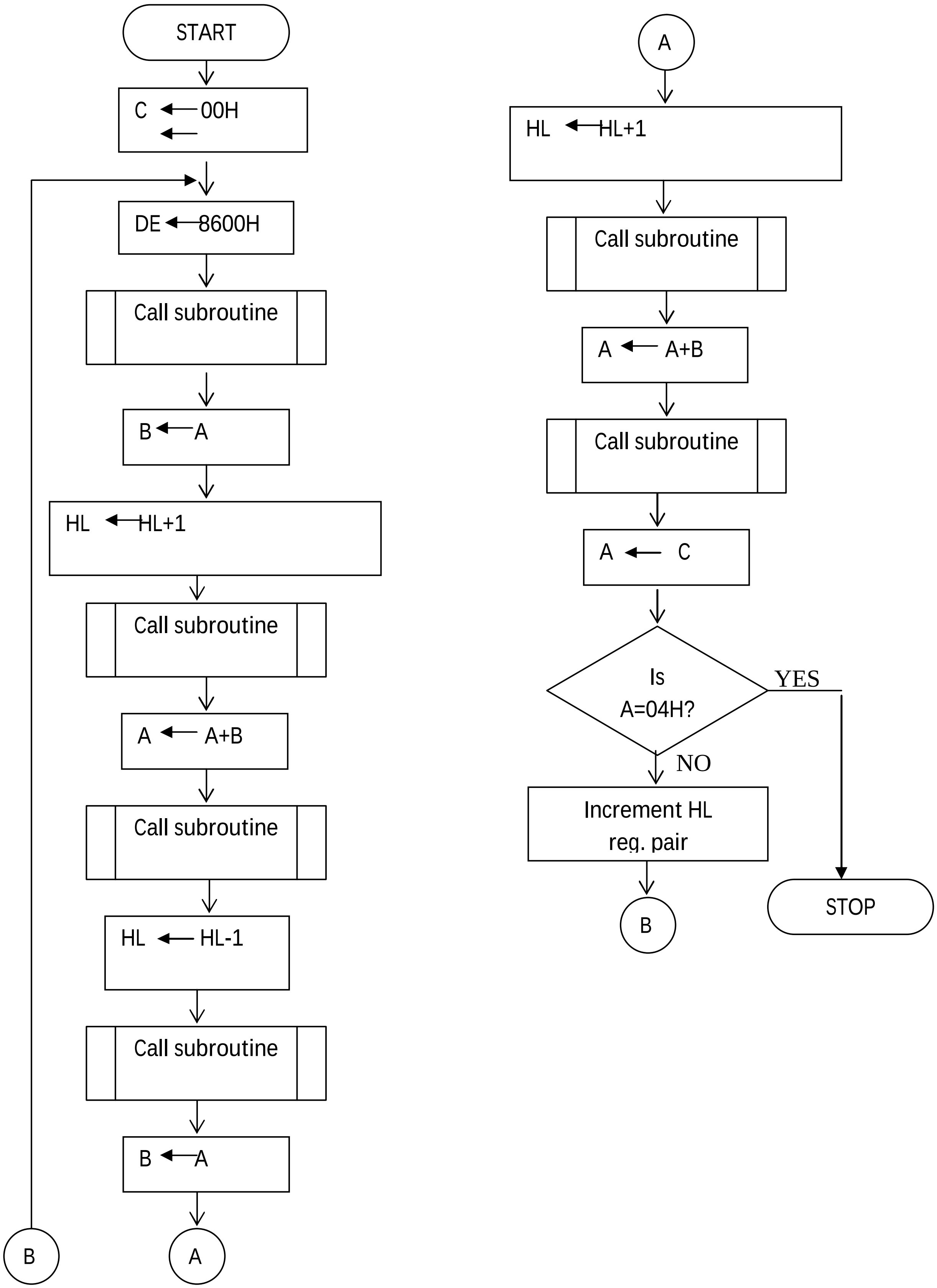
1. Load the 2 input matrices in the separate address and initialize the HL and the DE register pair with the starting address respectively.
2. Call a subroutine for performing the multiplication of one element of a matrix with the other element of the other matrix.
3. Call a subroutine to store the resultant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

PROGRAM:

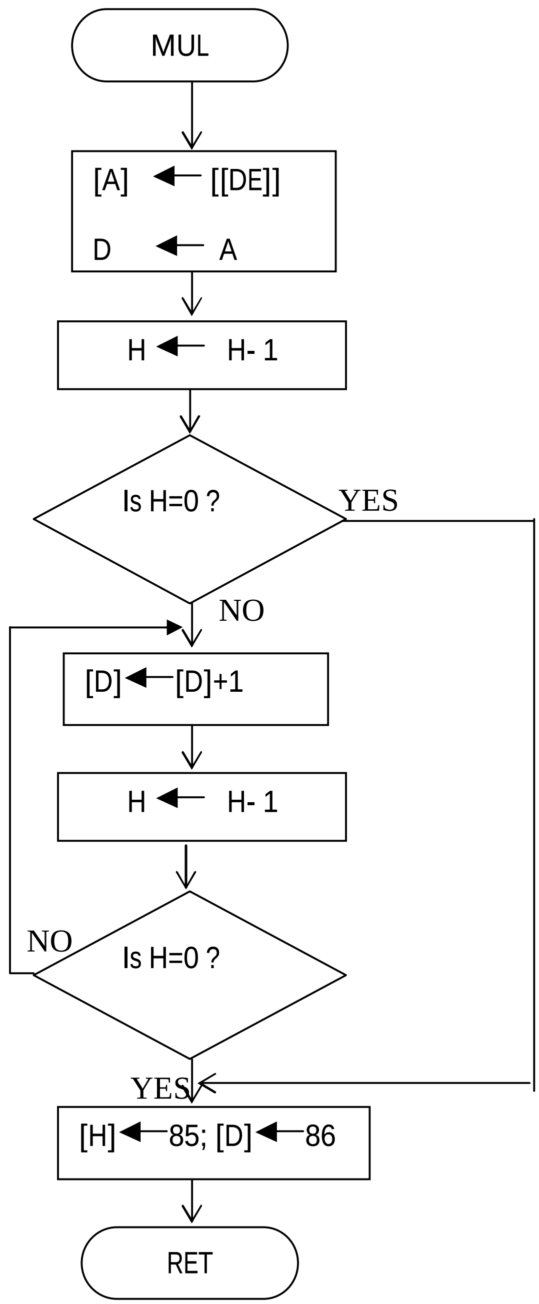
ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
8100			MVI	C, 00	Clear C reg.
8101					
8102			LXI	H, 8500	Initialize HL reg. to 4500
8103					
8104					
8105		LOOP2	LXI	D, 8600	Load DE register pair
8106					
8107					
8108			CALL	MUL	Call subroutine MUL
8109					
810A					
810B			MOV	B,A	Move A to B 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.

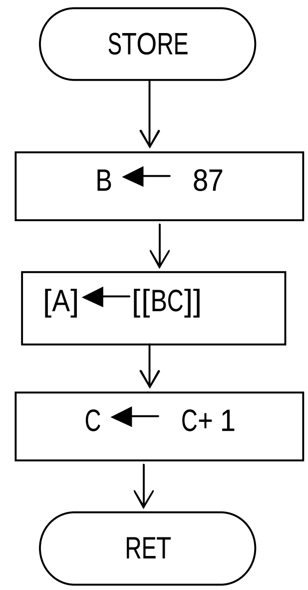
811C			INX	H	Increment HL register pair
811D			INX	D	Increment DE register pair
811E			INX	D	Increment DE register pair
811F			CALL	MUL	Call subroutine MUL
8120					
8121					
8122			ADD	B	Add A with B
8123			CALL	STORE	Call subroutine MUL
8124					
8125					
8126			MOV	A,C	Transfer C register content 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.

FLOW CHART:



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


MUL
[A] ← [[DE]]
D ← A
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

OBSERVATION:

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

RESULT:

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

Ex No: 7(A)

Date:

1.8086 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.

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

RESULT:

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

Ex No: 7(B)

Date:

2.8086 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.

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

RESULT:

A word is found and replaced from a string.

Ex No: 7(C)

Date:

3. 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.

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

RESULT:

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

Ex No: 7(D)

Date:

4.8086 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.

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

RESULT:

A group of data bytes are arranged in ascending order.

Ex No: 8(A)

Date:

1. INTERFACING 8255 WITH 8085

AIM:

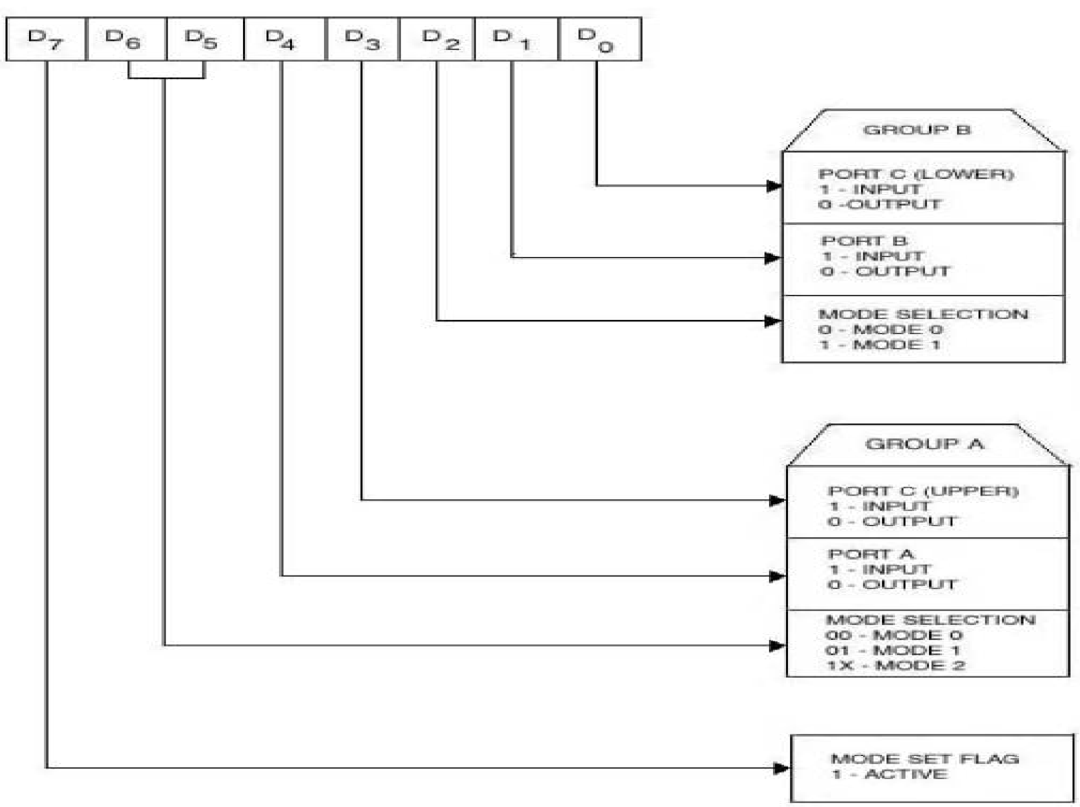
To interface programmable peripheral interface 8255 with 8085 and study its characteristics in mode0, mode1 and BSR mode.

APPARATUS REQUIRED:

8085 μp kit, 8255Interface board, DC regulated power supply, VXT parallel bus

I/O MODES:

Control Word:



MODE 0 – SIMPLE I/O MODE:

This mode provides simple I/O operations for each of the three ports and is suitable for synchronous data transfer. In this mode all the ports can be configured either as input or output port.

Let us initialize port A as input port and port B as output port

PROGRAM:

ADDRESS	OPCODES	LABEL	MNEMONICS	OPERAND	COMMENTS
4100		START:	MVI	A, 90	Initialize port A as Input and Port B as output.
4101					
4102			OUT	C6	Send Mode Control word
4103					
4104			IN	C0	Read from Port A
4105					
4106			OUT	C2	Display the data in port B
4107					
4108			STA	4200	Store the data read from Port A in 4200
4109					
410A					
410B			HLT		Stop the program.

MODE1 STROBED I/O MODE:

In this mode, port A and port B are used as data ports and port C is used as control signals for strobed I/O data transfer.

Let us initialize port A as input port in mode1

MAIN PROGRAM:

ADDRESS	OPCODES	LABEL	MNEMONICS	OPERAND	COMMENTS
4100		START:	MVI	A, B4	Initialize port A as Input port in mode 1.
4101					
4102			OUT	C6	Send Mode Control word
4103					
4104			MVI	A,09	Set the PC4 bit for INTE A
4105					

4106			OUT	C6	Display the data in port B
4107					
			EI		
4108			MVI	A,08	Enable RST5.5
4109					
410A			SIM		
			EI		
410B			HLT		Stop the program.

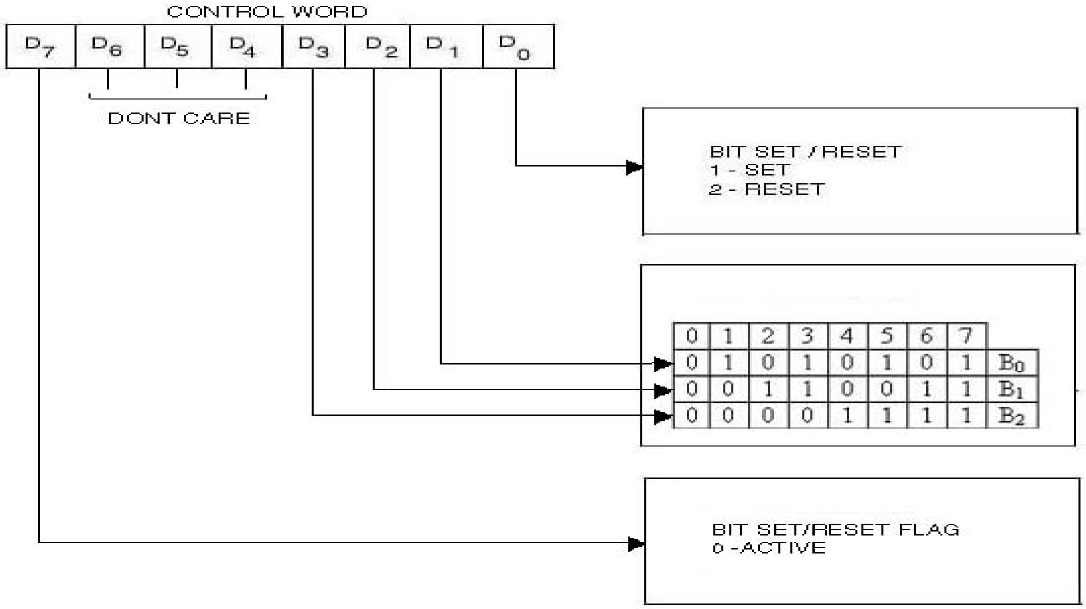
ISR (Interrupt Service Routine)

ADDRESS	OPCODES	LABEL	MNEMONICS	OPERAND	COMMENTS
4200		START:	IN	C0	Read from port A
4201					
4202			STA	4500	Store in 4500.
4203					
4204					
4205			HLT		Stop the program.

Sub program:

ADDRESS	OPCODES	LABEL	MNEMONICS	OPERAND	COMMENTS
405E			JMP	4200	Go to 4200
405F					
4060					

BSR MODE (Bit Set Reset mode)



Any lines of port c can be set or reset individually without affecting other lines using this mode. Let us set PC0 and PC3 bits using this mode.

PROGRAM:

ADDRESS	OPCODES	LABEL	MNEMONICS	OPERAND	COMMENTS
4100		START:	MVI	A, 01	Set PC0
4101					
4102			OUT	C6	Send Mode Control word
4103					
4104			MVI	A,07	Set PC3
4105					
4106			OUT	C6	Send Mode Control word
4107					
4109			HLT		Stop the program.

RESULT: Thus 8255 is interfaced and its characteristics in mode0,mode1 and BSR mode is studied.

Ex No: 8(B)

Date:

2. <u>INTERFACING 8253 TIMER WITH 8085</u>

Interfacing 8253 Programmable Interval Timer with 8085 μp

<u>AIM</u>:

To interface 8253 Interface board to 8085 μp and verify the operation of 8253in six different modes.

<u>APPARATUS REQUIRED</u>:

8085 μp kit, 8253 Interface board, DC regulated power supply, VXT parallel bus, CRO.

Mode 0 – Interrupt on terminal count:

The output will be initially low after mode set operations. After loading the counter, the output will be remaining low while counting and on terminal count; the output will become high, until reloaded again.

Let us set the channel 0 in mode 0. Connect the CLK 0 to the debounce circuit by changing the jumper J3 and then execute the following program.

Program:

Address	**Opcodes**	**Label**	**Mnemonic**	**Operands**	**Comments**
4100		START:	MVI	A, 30	Channel 0 in mode 0
4102			OUT	CE	Send Mode Control word
4104			MVI	A, 05	LSB of count
4106			OUT	C8	Write count to register
4108			MVI	A, 00	MSB of count
410A			OUT	C8	Write count to register
410C			HLT		

It is observed in CRO that the output of Channel 0 is initially LOW. After giving six clock pulses, the output goes HIGH.

Mode 1 – Programmable ONE-SHOT:

After loading the counter, the output will remain low following the rising edge of the gate input. The output will go high on the terminal count. It is retriggerable; hence the output will remain low for the full count, after any rising edge of the gate input.

Example:

The following program initializes channel 0 of 8253 in Mode 1 and also initiates triggering of Gate 0. OUT 0 goes low, as clock pulse after triggering the goes back to high level after 5 clock pulses. Execute the program, give clock pulses through the debounce logic and verify using CRO.

Address	Opcodes	Label	Mnemonic	Operands	Comments
4100		START:	MVI	A, 32	Channel 0 in mode 1
4102			OUT	CE	Send Mode Control word
4104			MVI	A, 05	LSB of count
4106			OUT	C8	Write count to register
4108			MVI	A, 00	MSB of count
410A			OUT	C8	Write count to register
410C			OUT	D0	Trigger Gate0
4100			HLT		

Mode 2 – Rate Generator:

It is a simple divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected but the subsequent period will reflect the new value.
Example:

Using Mode 2, Let us divide the clock present at Channel 1 by 10. Connect the CLK1 to PCLK.

Address	Opcodes	Label	Mnemonic	Operands	Comments
4100	3E 74	START:	MVI	A, 74	Channel 1 in mode 2
4102	D3 CE		OUT	CE	Send Mode Control word
4104	3E 0A		MVI	A, 0A	LSB of count
4106	D3 CA		OUT	CA	Write count to register
4108	3E 00		MVI	A, 00	MSB of count
410A	D3 CA		OUT	CA	Write count to register
410C	76		HLT		

In CRO observe simultaneously the input clock to channel 1 and the output at Out1.

Mode 3 Square wave generator:

It is similar to Mode 2 except that the output will remain high until one half of count and go low for the other half for even number count. If the count is odd, the output will be high for (count + 1)/2 counts. This mode is used of generating Baud rate for 8251A (USART).

Example:

We utilize Mode 0 to generate a square wave of frequency 150 KHz at channel 0.

Address	Opcodes	Label	Mnemonic	Operands	Comments
4100	3E 36	START:	MVI	A, 36	Channel 0 in mode 3
4102	D3 CE		OUT	CE	Send Mode Control word
4104	3E 0A		MVI	A, 0A	LSB of count
4106	D3 C8		OUT	C8	Write count to register
4108	3E 00		MVI	A, 00	MSB of count
410A	D3 C8		OUT	C8	Write count to register
410C	76		HLT		

Set the jumper, so that the clock 0 of 8253 is given a square wave of frequency 1.5 MHz. This program divides this PCLK by 10 and thus the output at channel 0 is 150 KHz.

Vary the frequency by varying the count. Here the maximum count is FFFF H. So, the square wave will remain high for 7FFF H counts and remain low for 7FFF H counts. Thus with the input clock frequency of 1.5 MHz, which corresponds to a period of 0.067 microseconds, the resulting square wave has an ON time of 0.02184 microseconds and an OFF time of 0.02184 microseconds.

To increase the time period of square wave, set the jumpers such that CLK2 of 8253 is connected to OUT 0. Using the above-mentioned program, output a square wave of frequency 150 KHz at channel 0. Now this is the clock to channel 2.

Mode 4: Software Triggered Strobe:

The output is high after mode is set and also during counting. On terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

The following program initializes channel 2 of 8253 in mode 4.

Example:

Connect OUT 0 to CLK 2 (jumper J1). Execute the program and observe the output OUT 2. Counter 2 will generate a pulse after 1 second.

Address	**Opcodes**	**Label**	**Mnemonic**	**Operands**	**Comments**
4100		START:	MVI	A, 36	Channel 0 in mode 0
4102			OUT	CE	Send Mode Control word
4104			MVI	A, 0A	LSB of count
4106			OUT	C8	Write count to register
4108			MVI	A, 00	MSB of count
410A			OUT	C8	Write count to register
410C			MVI	A, B8	Channel 2 in Mode 4
410E			OUT	CE	Send Mode control Word
4110			MVI	A, 98	LSB of Count
4112			OUT	CC	Write Count to register
4114			MVI	A, 3A	MSB of Count
4116			OUT	CC	Write Count to register

4118			HLT		

Mode 5 Hardware triggered strobe:

Counter starts counting after rising edge of trigger input and output goes low for one clock period when terminal count is reached. The counter is retrigger able.

Example:

The program that follows initializes channel 0 in mode 5 and also triggers Gate 0. Connect CLK 0 to debounce circuit.

Execute the program. After giving Six clock pulses, you can see using CRO, the initially HIGH output goes LOW. The output (OUT 0 pin) goes high on the next clock pulse.

Address	**Opcodes**	**Label**	**Mnemonic**	**Operands**	**Comments**
4100		START:	MVI	A, 1A	Channel 0 in mode 5
4102			OUT	CE	Send Mode Control word
4104			MVI	A, 05	LSB of count
4106			OUT	C8	Write count to register
4108			MVI	A, 00	MSB of count
410A			OUT	D0	Trigger Gate 0
410C			HLT		

Result:

Thus the 8253 has been interfaced to 8085 μp and six different modes of 8253 have been studied.

Ex No: 9(A)

Date:

7.INTERFACING 8279 WITH 8085

AIM:

To interface 8279 Interface board to 8085 µ p and verify the operation of 8279.

APPARATUS REQUIRED :

8085 µ p kit, 8253 Interface board, DC regulated power supply.

INTERFACING DIAGRAM

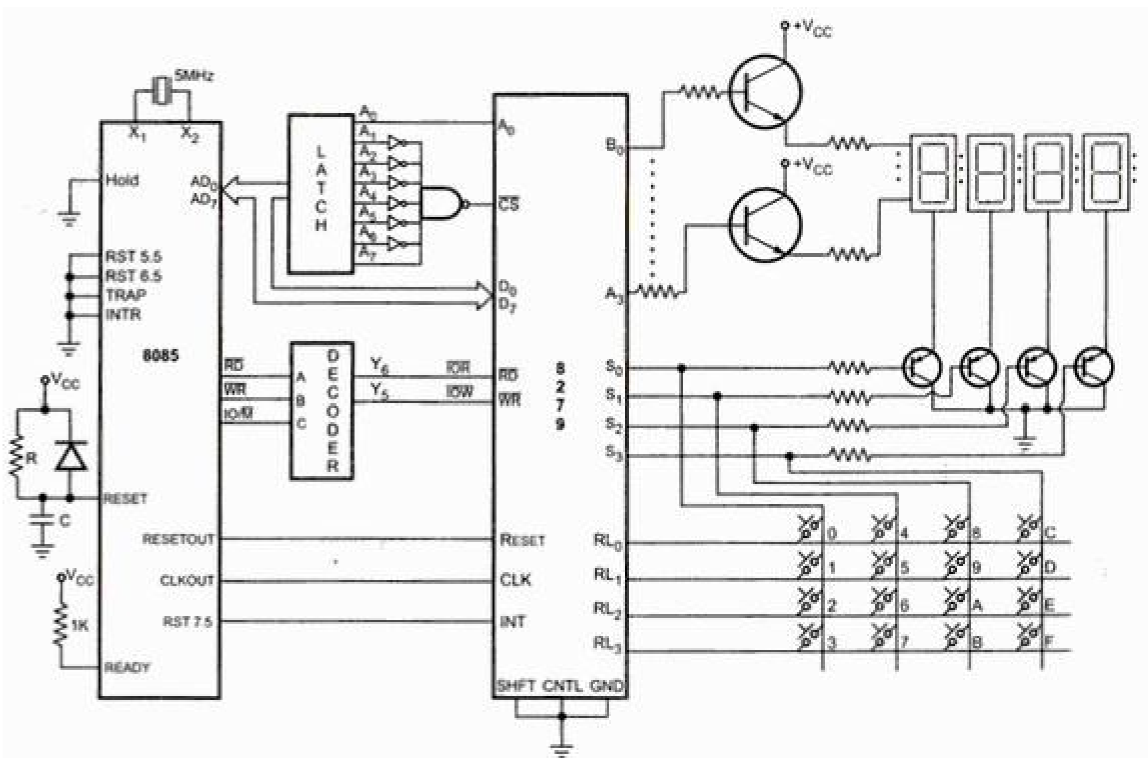


Fig. — Keyboard and display interfacing in decoded mode using 8279

The four steps needed to write the software are:

Step 1: Find keyboard/display command word.

			D	D	K	K	K	
0	0	0	0	0	0	0	1	= 01H

Step 2: Find program clock command word

			P	P	P	P	P	
0	0	1	1	1	0	0	1	= 39H

Step 3: Find Read FIFO RAM command word.

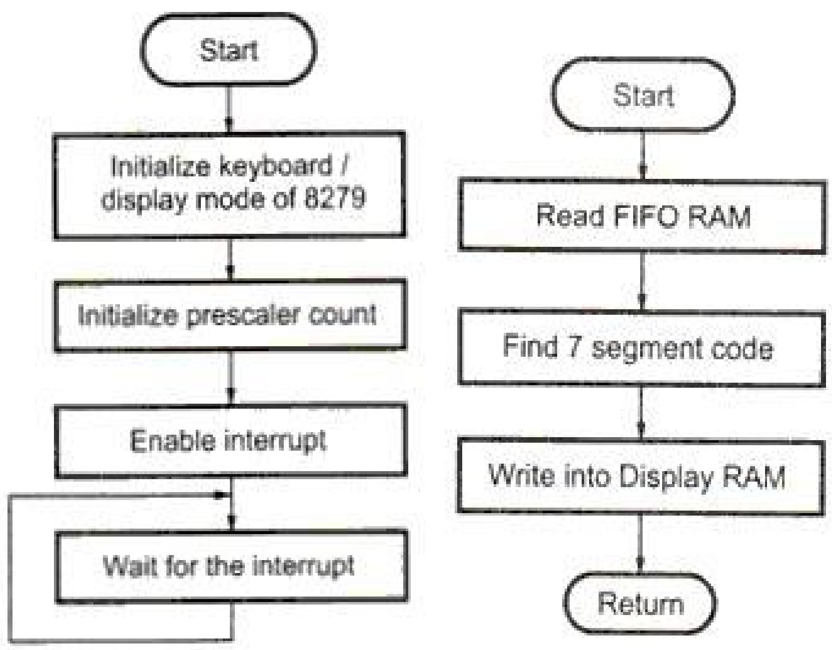
			A	AI	A	A	A	
0	1	0	0	X	0	0	0	= 40H

Step 4: Find Write FIFO RAM command word.

			AI	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	= 80H

Source Program and Interrupt Service Routine

FLOWCHART:



SOURCE PROGRAM:

```
MVI A, 00H     : Initialize keyboard/display in encoded
OUT 81H        : scan keyboard 2 key lockout mode
MVI A, 34H
OUT 81H        : Initialize prescaler count
MVI A, 0BH

SIM

 EI

HERE: JMP
HERE    : Wai
interrupt
```

INTERRUPT SERVICE ROUTINE

```
MVI A, 40H     : Initialize 8279 in read FIFO RAM mode
OUT 81H
IN 80H         : Get keycode
MVI H, 62H     : Initialize memory pointer to point
MOV L, A       : 7-Segment code
MVI A, 80H     : Initialize 8279 in write display RAM mode
OUT 81H
MOV A, M       : Get the 7 segment code
OUT 80H        : Write 7-segment code in display RAM
EI             : Enable interrupt
RET            : Return to main program
```

RESULT:

Thus the 8279 has been interfaced to 8085 µ p and the operation of 8279 is verified.

Ex No:9(B)

Date:

INTERFACING 8251 WITH 8085

AIM:

To interface 8251A Interface board to 8085 μ p and verify the operation of 8251A.

APPARATUS REQUIRED :

8085 μ p kit, 8251A Interface board, DC regulated power supply, RS232 cable.

INTERFACING DIAGRAM

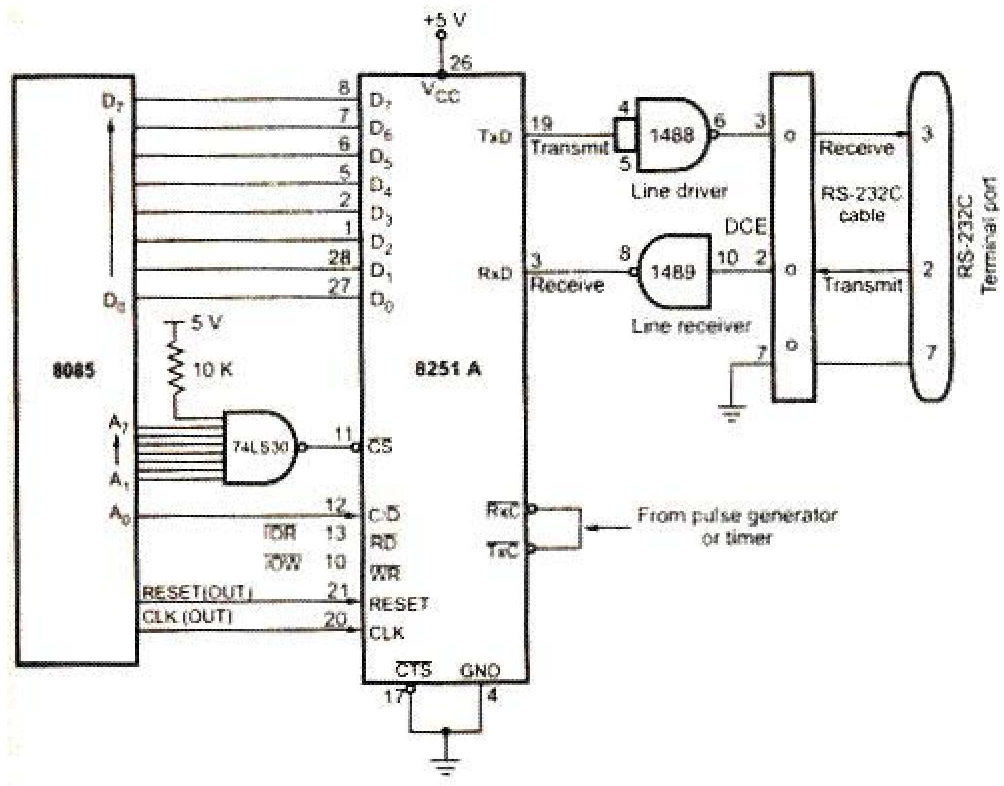
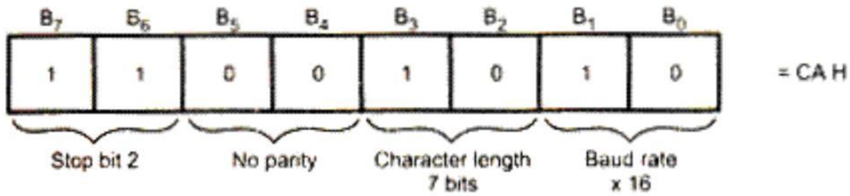


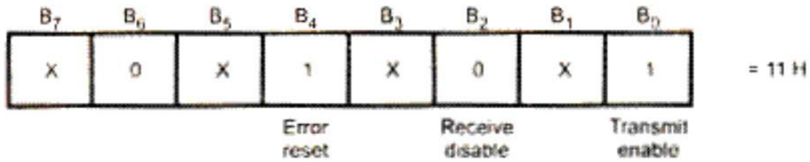
Fig.- Schematic of interfacing an RS-232C terminal with an 8085 system using the 8251A

TRANSMITTING THE DATA:

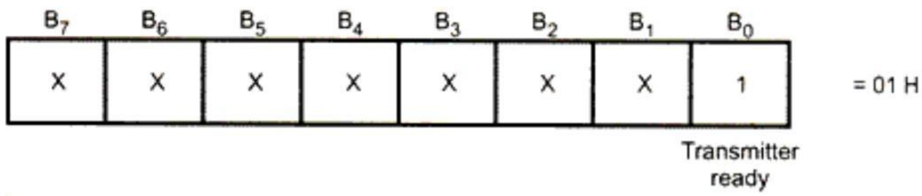
Mode word necessary for the given specification is as follows :



Command word necessary for the given specification is as follows

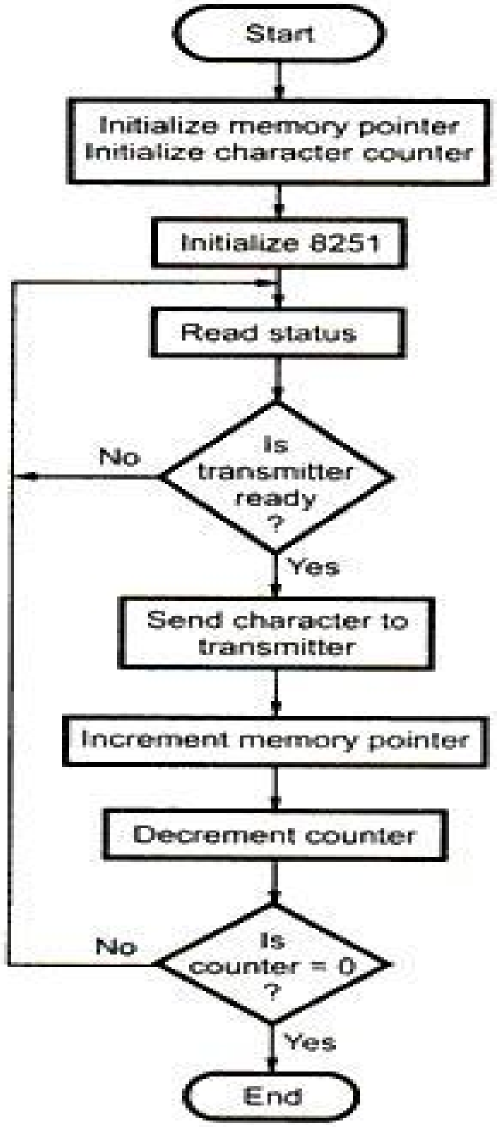


Status word necessary for the given specification is as follows



If bit 0 of the **Status** word is logic '1' then transmitter is ready to accept the **character**

FLOWCHART

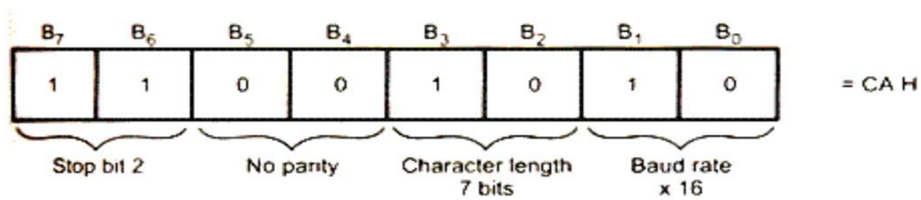


SOURCE PROGRAM:

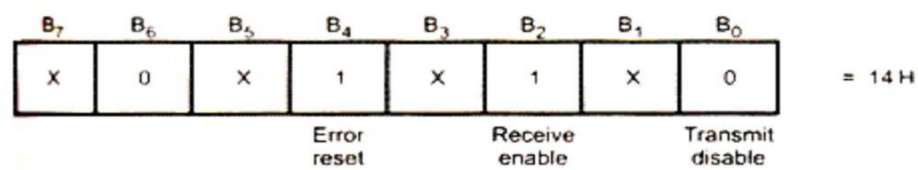
```
LXI H, 2200H      : Initialize memory pointer to pointer the message
MVI C, 32H        : Initialize counter to send 50 characters
MVI A, 00H
OUT FFH
OUT FFH           : Dummy mode word
OUT FFH
MVI A, 40H        : Reset command word
OUT FFH           : Reset 8251A
MVI A, CAH        : Mode word initialization
OUT FFH
```

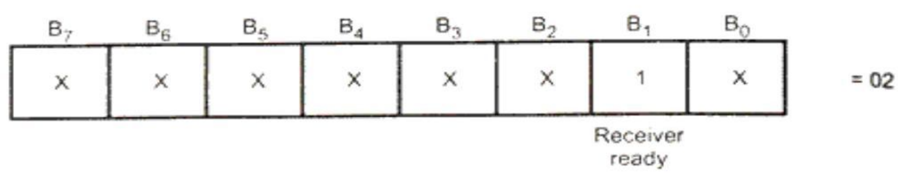
```
      MVI A, 11H        : Command word initialization
      OUT FFH
CHECK: IN FFH
      ANI 0lH           : Check TxRDY
      JZ CHECK          : Is TxRDY I? if not, check again
      MOV A, M          : Get the character in accumulator
      OUT FEH           : Send character to the transmitter
      INX H             : Increment memory pointer
      DCR C             : Decrement counter
      JNZ CHECK         : if not zero, send next character
      HLT               : Stop program execution
```

RECEIVING THE DATA



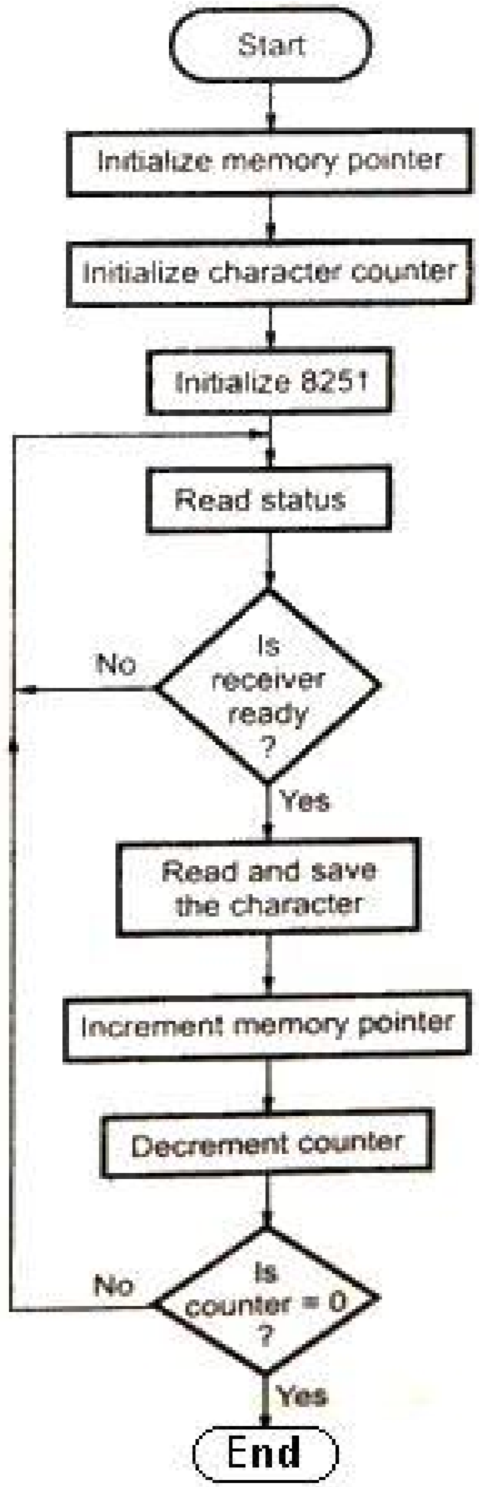
Command word necessary for the given specification is as follows





If bit 1 of the status word is logic '1' then receiver is ready to give the character

FLOWCHART



Start
Initialize memory pointer
Initialize character counter
Initialize 8251
Read status
Is receiver ready ?
No
Yes
Read and save the character
Increment memory pointer
Decrement counter
Is counter = 0 ?
No
Yes
End

SOURCE PROGRAM:

```
      LXI H, 2300 H        : Initialize memory pointer
      MVI C, FFH           : Initialize counter to accept 25 characters
      MVI A, 00H
      OUT FFH
      OUT FFH              : Dummy mode word
      OUT FFH
      MVI A, 40H           : Reset command word
      OUT FFH              : Reset 8251 A
      MVI A, CAH           : Mode word initialization
      OUT FFH
      MVI A, 14 H          : Command word initialization
      OUT FFH
CHECK: IN FFH
      ANI 02 H             : Check RxRDY
      JZ CHECK             : Is RxRDY ? If not, check again
      IN FEH               : Get the character
      MOV M, A             : save the character
      INX H                : Increment memory pointer
      DCR C                : Decrement memory pointer
      OUT FEH              : Send character to the transmitter
      JNZ CHECK            : If not zero, accept next character
      HLT                  : Stop program execution
```

Note: Reading of status word is necessary for checking the status of RxD line of 8085 that whether receiver is ready to give data or not.

RESULT:

Thus the 8251A has been interfaced to 8085 μ p and the operation of 8251A is verified.

Ex No:10
Date:

8051 MICROCONTROLLER BASED EXPERIMENTS – SIMPLE ASSEMBLY LANGUAGE PROGRAMS
(8051 BASIC PROGRAMMING)

Aim:

To program 8051 using its Arithmetic and Logical and Bit Manipulation instructions.

a) Arithmetic operations

Address	Label	Mnemonics	Machine Code	Comments
		MOV DPTR, #8500		
		MOVX A, @DPTR		
		MOV B, A		
		MOV R0, A		
		INC DPTR		
		MOVX A, @DPTR		
		MOV R1, A		
		ADD A, B		
		INC DPTR		
		MOVX @DPTR, A		
		MOV R2, A		
		MOV A, R1		
		SUBB A, B		
		INC DPTR		
		MOVX @DPTR, A		
		MOV R3, A		
		MOV B, R2		
		MUL AB		

```
INC DPTR
MOVX  @DPTR, A
MOV A, R2
MOV B, R3
DIV AB
INC DPTR
MOVX  @DPTR, A
LCALL 00BB
```

Input: M8500 - a

M8501 - b

Output: M8502 : sum (a+b)

M8503: difference (a-b)

M8504: Product ((a+b)×(a-b))

M8505: Quotient ((a+b)/(a-b))

b) 32 bit subtraction

Address	Label	Mnemonics	Machine Code	Comments

		CLR C
		
		MOV A, 43
		
		SUBB A, 53
		
		MOV 63, A
		
		MOV A, 42
		
		SUBB A, 52
		
		MOV 62, A
		
		MOV A, 41
		
		SUBB A, 51
		
		MOV 61, A
		
		MOV A, 40
		
		SUBB A, 50
		
		MOV 60, A
		
		LCALL 00BB		

Input: I40 to 43 – data 1

I50 to 53 – data 2

Output: I60 to 63 – difference

C) Fibonacci series

Address	Label	Mnemonics	Machine Code	Comments
		MOV R0, 60		
		MOV R1, #01		
		MOV R2, #01		
		MOV A, #00		
		MOV DPTR, # 9000		
		CJNE R0, #00, BEGIN		
		LJMP EXIT		
	BEGIN:	MOVX @DPTR, A		
		INC DPTR		
	RPT:	MOV R2, A		
		ADD A, R1		
		MOV 01, 02		
		MOVX @DPTR, A		

```
            INC DPTR
            DJNZ R0, RPT
EXIT:       LCALL 00BB
```

INPUT: I60 – COUNT

OUTPUT: M9000 – 00

M9001 – 01

M9002 – 01

M9003 – 02 & so on…

Ex No:11

Date:

8051 MICROCONTROLLER BASED EXPERIMENTS – SIMPLE CONTROL APPLICATIONS

STEPPER MOTOR INTERFACING WITH 8051

AIM:

To interface a stepper motor with 8051 microcontroller and operate it.

APPARATUS REQUIRED:

i) 8051 microcontroller kit.

ii) Stepper motor

THEORY:

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotary motion occurs in a step- wise manner from one equilibrium position to the next. Stepper Motors are used very wisely in position control systems like printers, disk drives, process control machine tools, etc.

The basic two-phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding. The excitation of any one winding generates a North Pole. A South Pole gets induced at the diametrically opposite side. The rotor magnetic system has two end faces. It is a permanent magnet with one face as South Pole and the other as North Pole.

The Stepper Motor windings A1, A2, B1, B2 are cyclically excited with a DC current to run the motor in clockwise direction. By reversing the phase sequence as A1, B2, A2, B1, anticlockwise stepping can be obtained.

2-PHASE SWITCHING SCHEME:

In this scheme, any two adjacent stator windings are energized. The switching scheme is shown in the table given below. This scheme produces more torque.

ANTICLOCKWISE						**CLOCKWISE**					
STEP	**A1**	**A2**	**B1**	**B2**	**DATA**	**STEP**	**A1**	**A2**	**B1**	**B2**	**DATA**
1	1	0	0	1	9h	1	1	0	1	0	Ah
2	0	1	0	1	5h	2	0	1	1	0	6h
3	0	1	1	0	6h	3	0	1	0	1	5h
4	1	0	1	0	Ah	4	1	0	0	1	9h

ADDRESS DECODING LOGIC:

The 74138 chip is used for generating the address decoding logic to generate the device select pulses, CS1 & CS2 for selecting the IC 74175.The 74175 latches the data bus to the stepper motor driving circuitry.

PROGRAM :

Address	**OPCODES**	*Label*			Comments
			ORG	4100h	
4100		START:	MOV	DPTR, #TABLE	Load the start address of switching scheme data TABLE into Data Pointer (DPTR)
4103			MOV	R0, #04	Load the count in R0
4105		LOOP:	MOVX	A, @DPTR	Load the number in TABLE into A
4106			PUSH	DPH	Push DPTR value to Stack
4108			PUSH	DPL	
410A			MOV	DPTR, #0FFC0h	Load the Motor port address into DPTR
410D			MOVX	@DPTR, A	Send the value in A to stepper Motor port address
410E			MOV	R4, #0FFh	Delay loop to cause a specific amount of time delay before next data item is sent to the Motor
4110		DELAY :	MOV	R5, #0FFh	
4112		DELAY 1:	DJNZ	R5, DELAY1	
4114			DJNZ	R4, DELAY	
4116			POP	DPL	POP back DPTR value from Stack
4118			POP	DPH	
411A			INC	DPTR	Increment DPTR to point to next item in the table
411B			DJNZ	R0, LOOP	Decrement R0, if not zero repeat the loop
411D			SJMP	START	Short jump to Start of the program to make the motor rotate continuously
411F		TABLE:	DB	09 05 06 0Ah	Values as per two-phase switching scheme

Stepper Motor requires logic signals of relatively high power. Therefore, the interface circuitry that generates the driving pulses use silicon darlington pair transistors. The inputs for the interface circuit are TTL pulses generated under software control using the Microcontroller Kit. The TTL levels of pulse sequence from the data bus is translated to high voltage output pulses using a buffer 7407 with open collector.

PROCEDURE:

Enter the above program starting from location 4100.and execute the same. The stepper motor rotates. Varying the count at R4 and R5 can vary the speed. Entering the data in the look-up TABLE in the reverse order can vary direction of rotation.

RESULT:

Thus a stepper motor was interfaced with 8051 and run in forward and reverse directions at various speeds.