SYLLABUS

CS2259-MICROPROCESSORS LABORATORY

1. Programming with 8085
2. Programming with 8086-experiments including BIOS/DOS calls: Keyboard control, Display, File Manipulation.
3. Interfacing with 8085/8086-8255,8253
4. Interfacing with 8085/8086-8279,8251
5. 8051 Microcontroller based experiments for Control Applications
6. Mini- Project

GET
CSE
IT

1. Assemply Language programs in 8085

(A). 8 BIT DATA ADDITION

AIM:

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

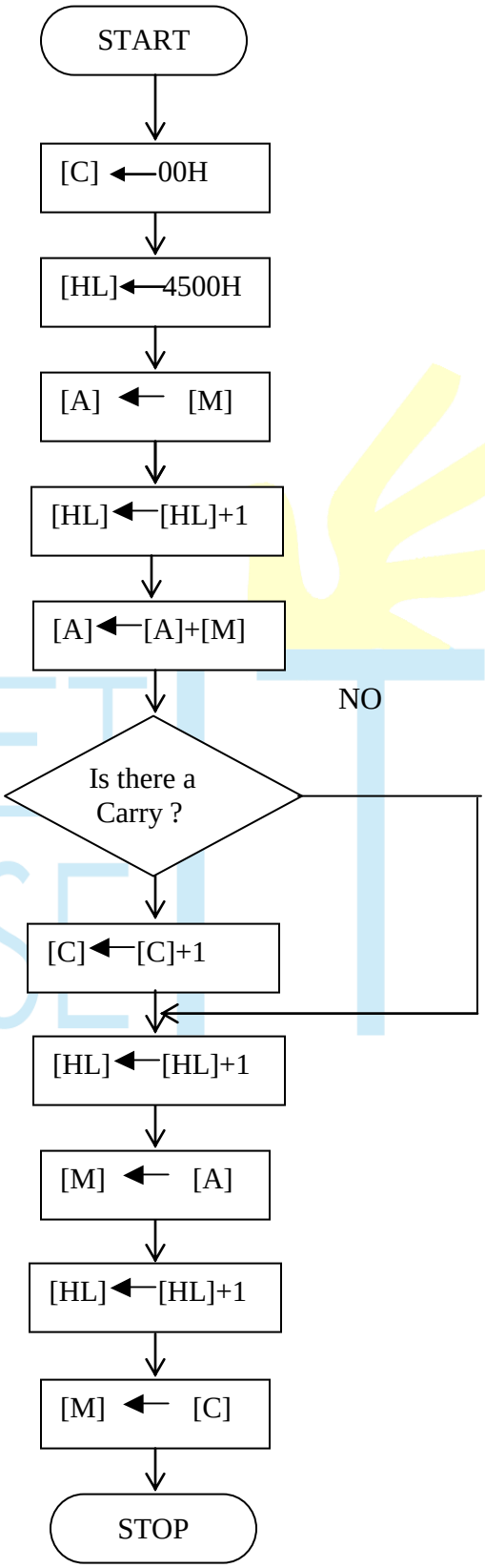
ALGORITHM:

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.

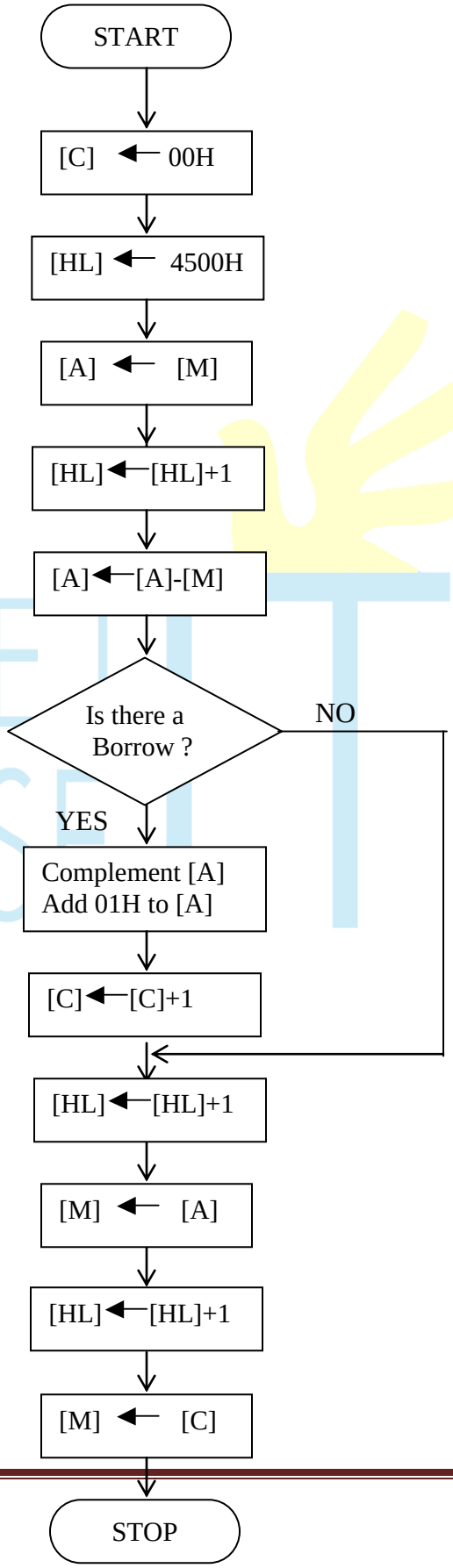
ALGORITHM:

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:

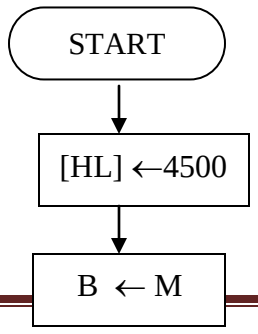
LOGIC: Multiplication can be done by repeated addition.

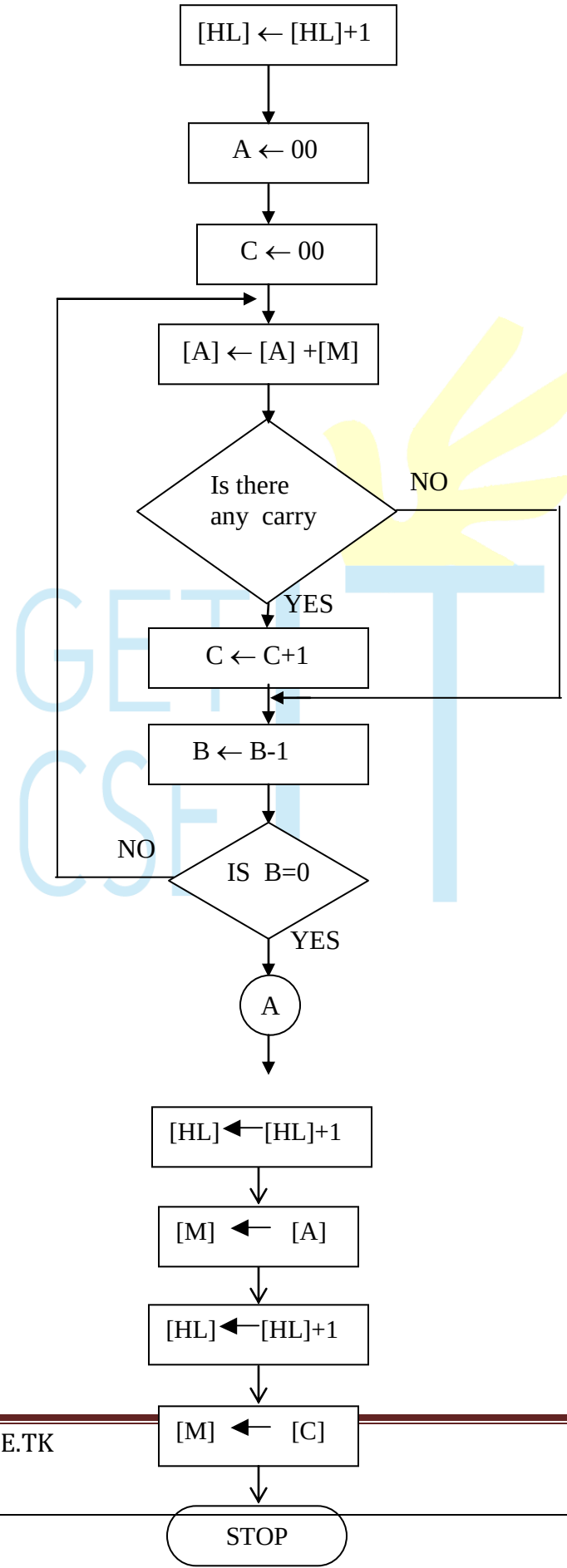
1. Initialize memory pointer to data location.
2. Move multiplicand to a register.
3. Move the multiplier to another register.
4. Clear the accumulator.
5. Add multiplicand to accumulator
6. Decrement multiplier
7. Repeat step 5 till multiplier comes to zero.
8. The result, which is in the accumulator,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

RESULT:

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

FLOW CHART:




[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
[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:

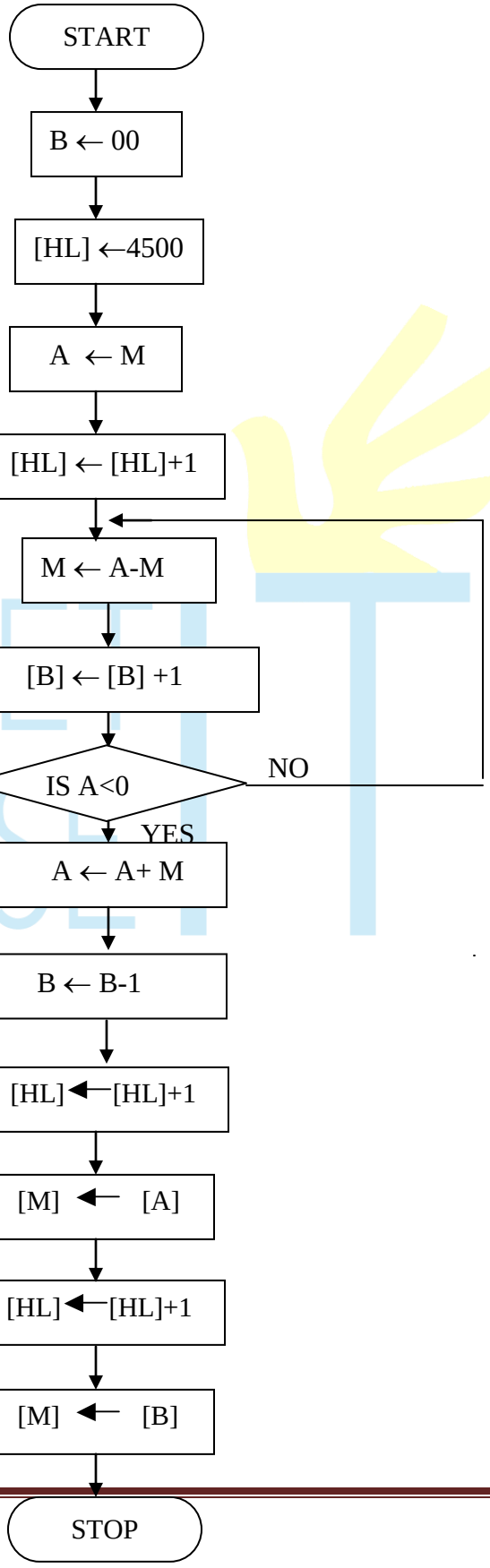
LOGIC: Division is done using the method Repeated subtraction.

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	

GET
CSE
IT

4(A). 16 BIT DATA ADDITION

AIM:

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

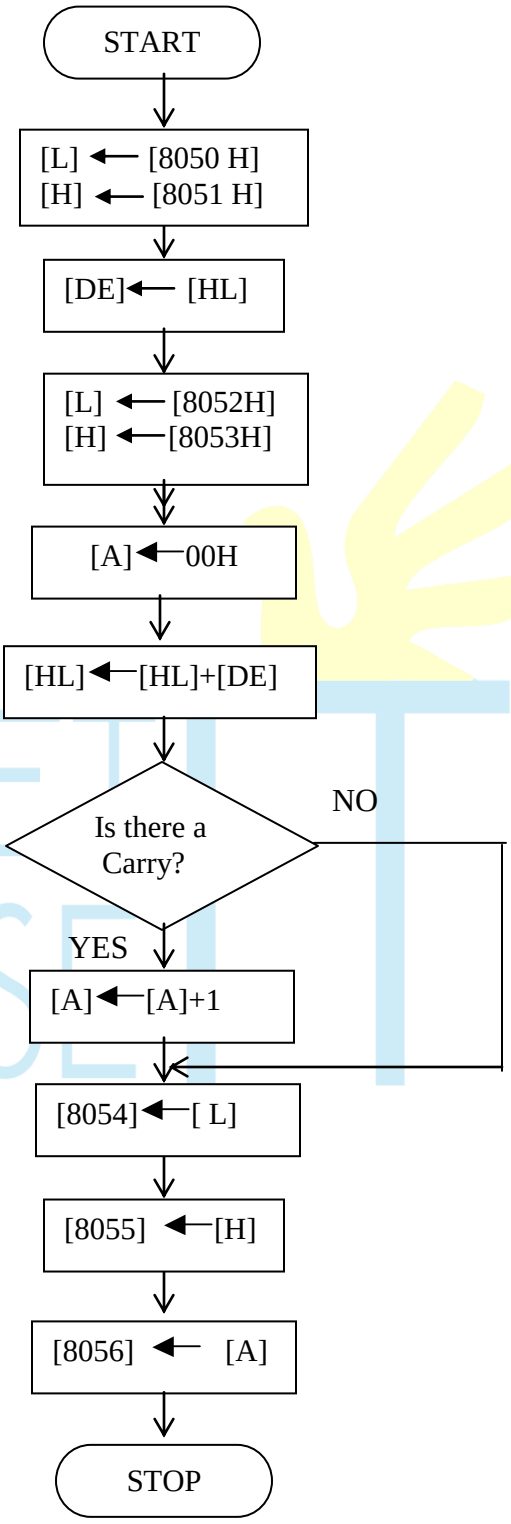
ALGORITHM:

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.

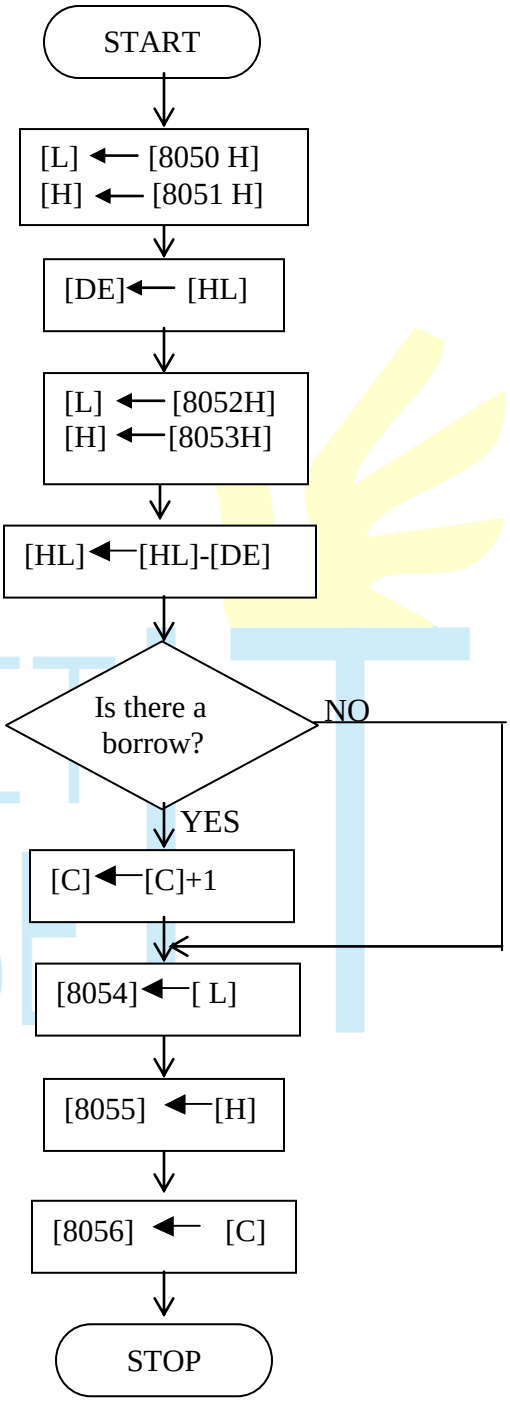
ALGORITHM:

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.

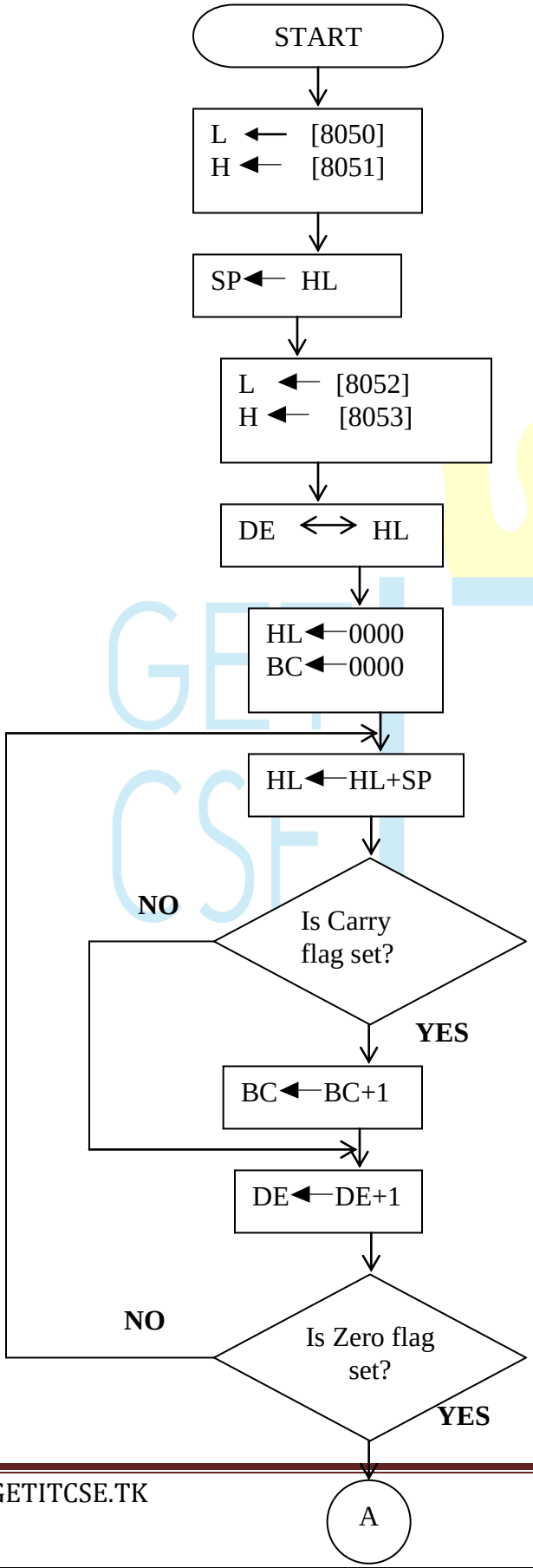
ALGORITHM:

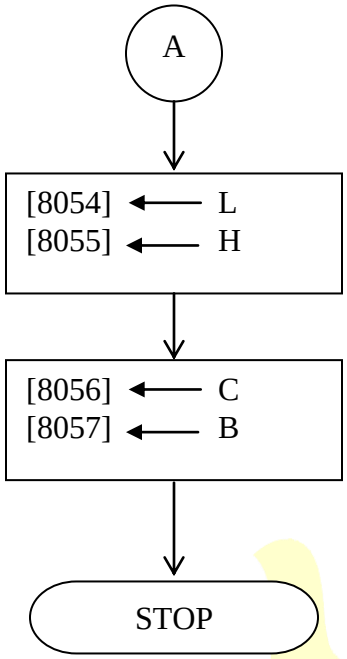
1. Get the multiplier and multiplicand.
2. Initialize a register to store partial product.
3. Add multiplicand, multiplier times.
4. Store the result in consecutive memory locations.

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.

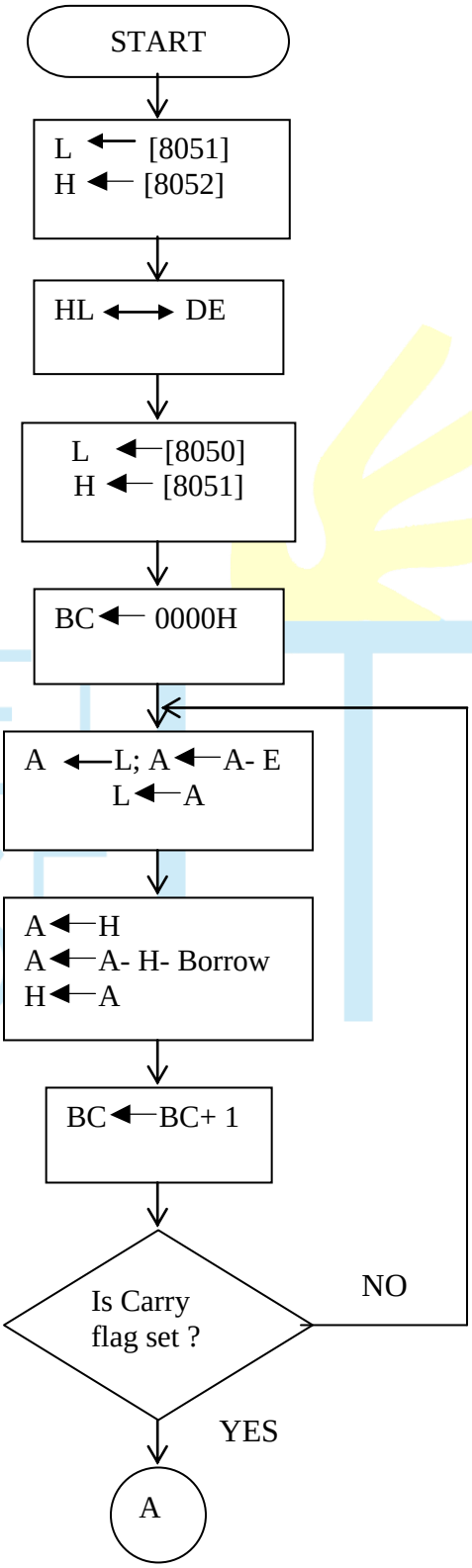
ALGORITHM:

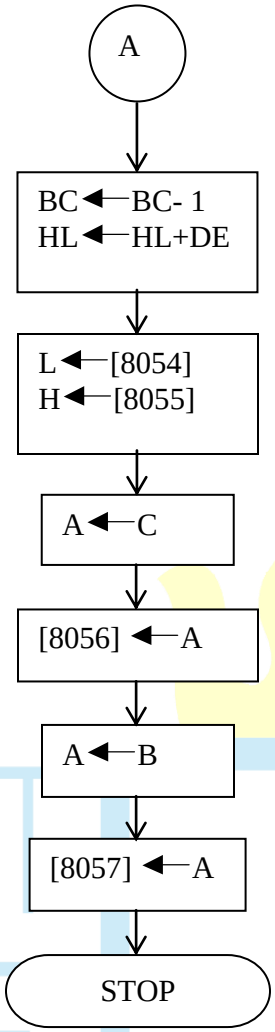
1. Get the dividend and divisor.
2. Initialize the register for quotient.
3. Repeatedly subtract divisor from dividend till dividend becomes less than divisor.
4. Count the number of subtraction which equals the quotient.
5. Store the result in memory.

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.

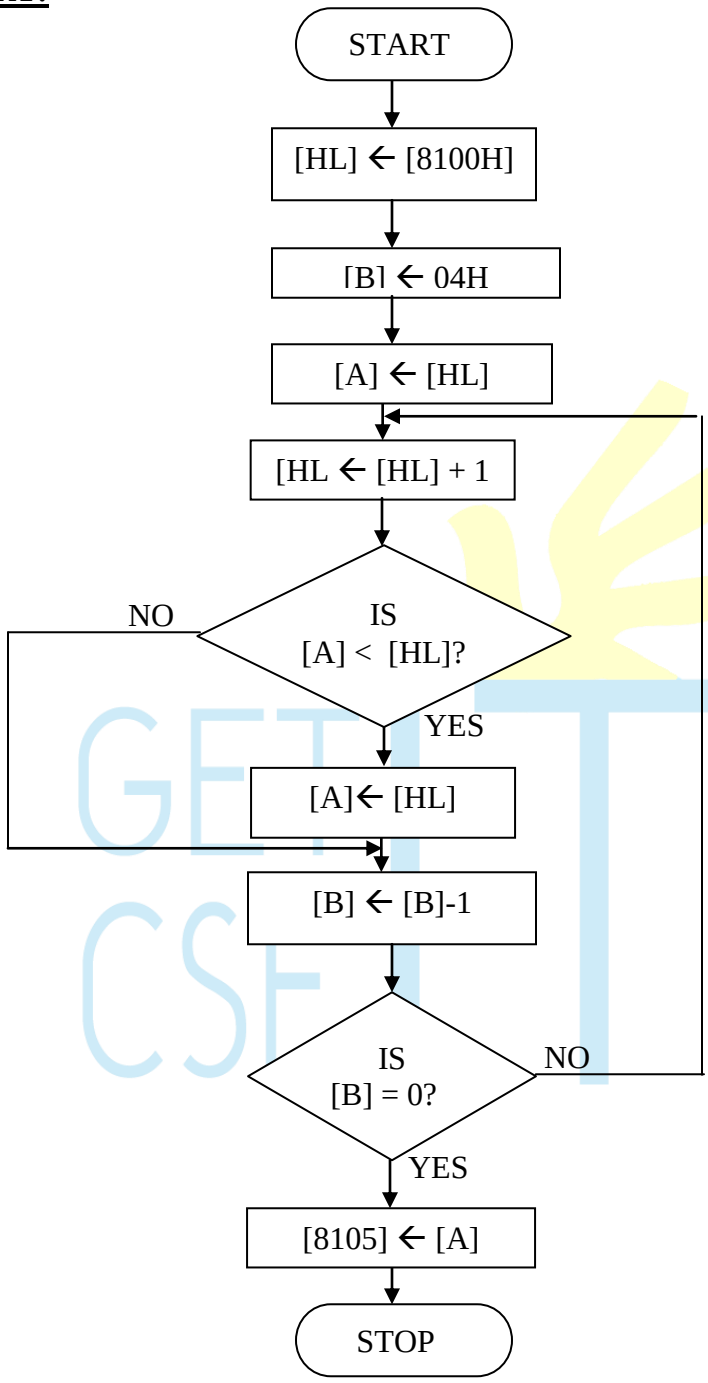
ALGORITHM:

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.

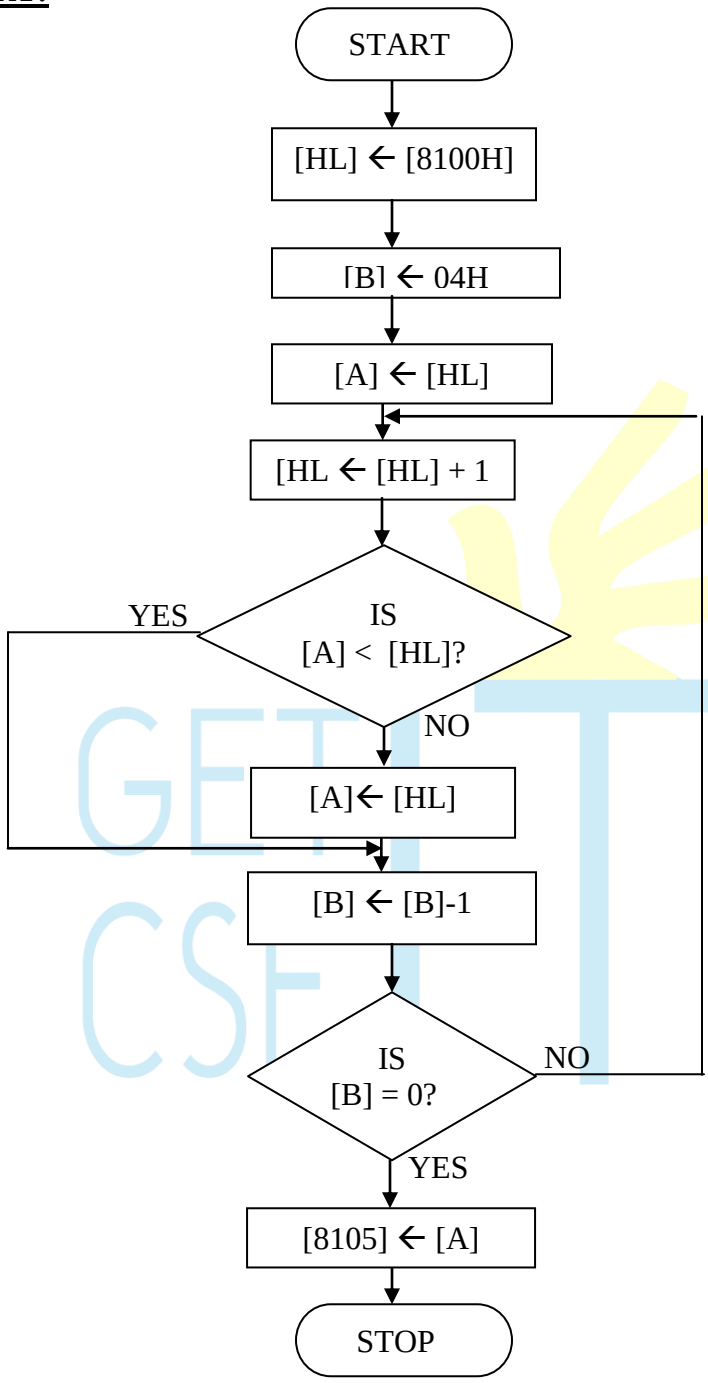
ALGORITHM:

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.

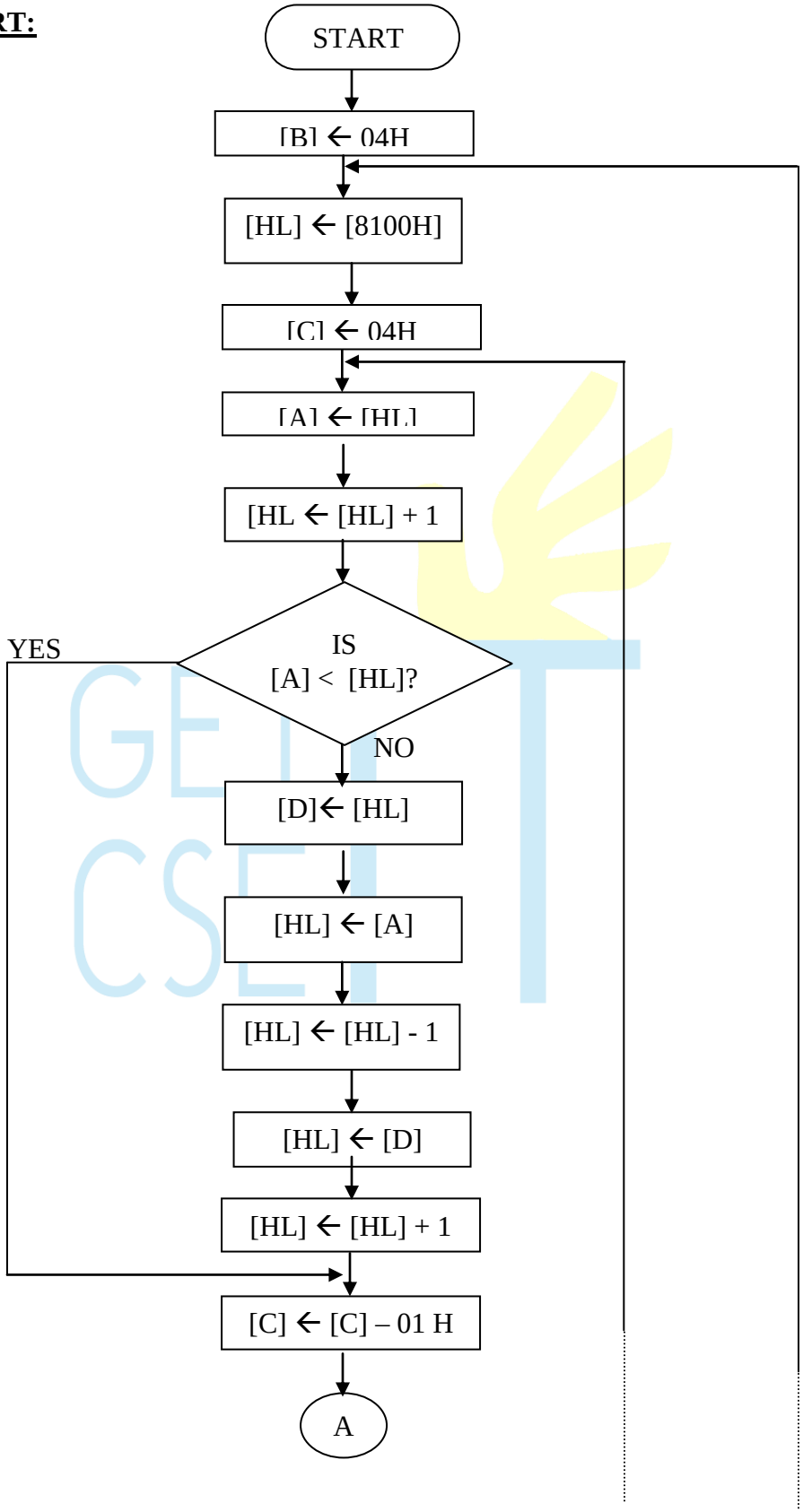
ALGORITHM:

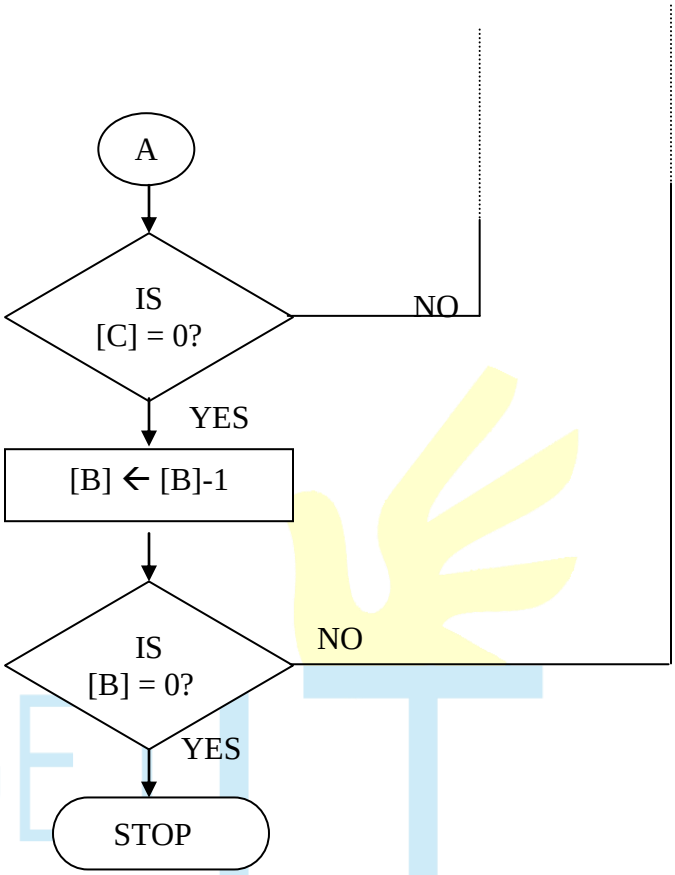
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is larger than second then I interchange the number.
3. If the first number is smaller, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

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.

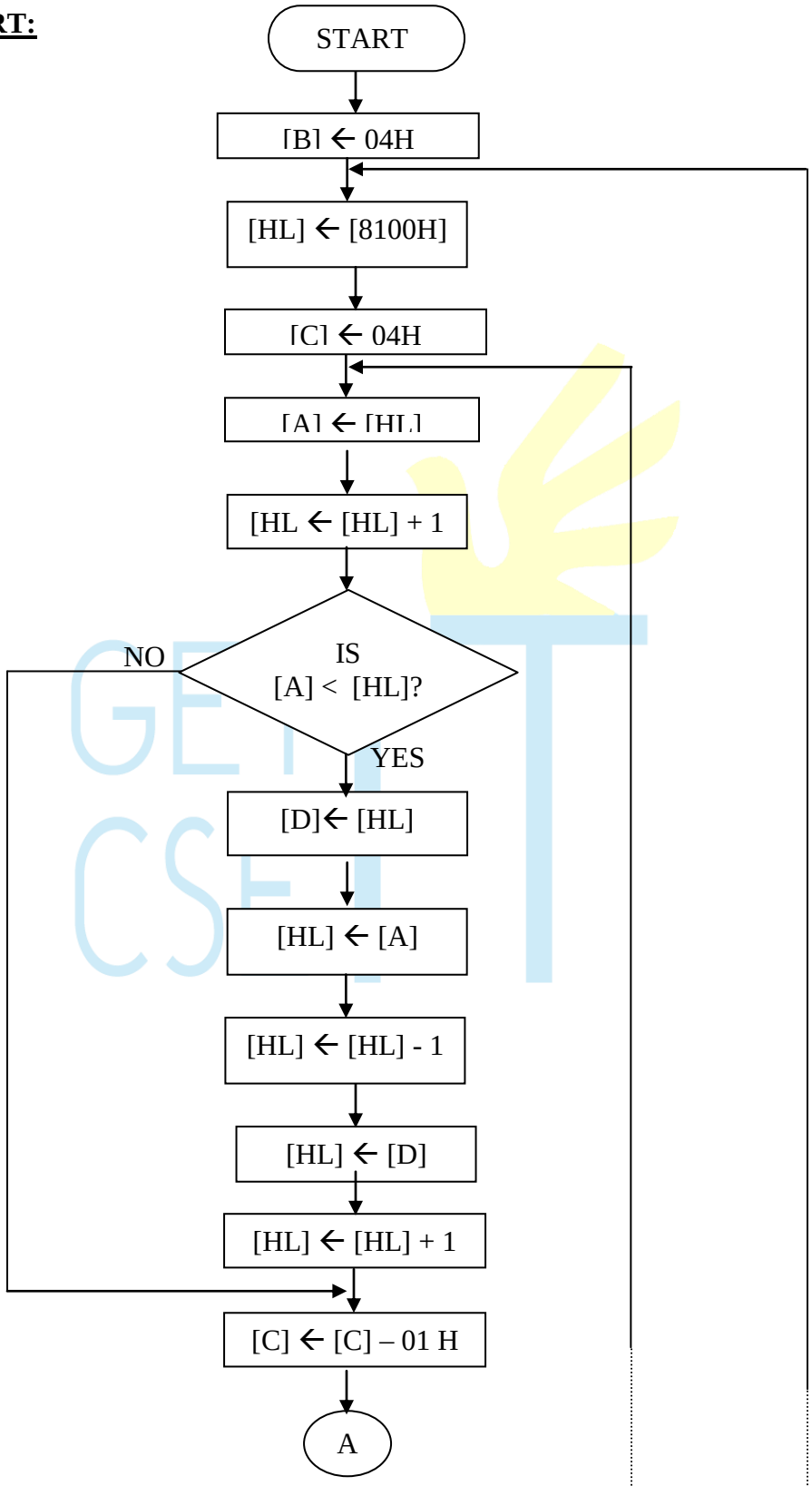
ALGORITHM:

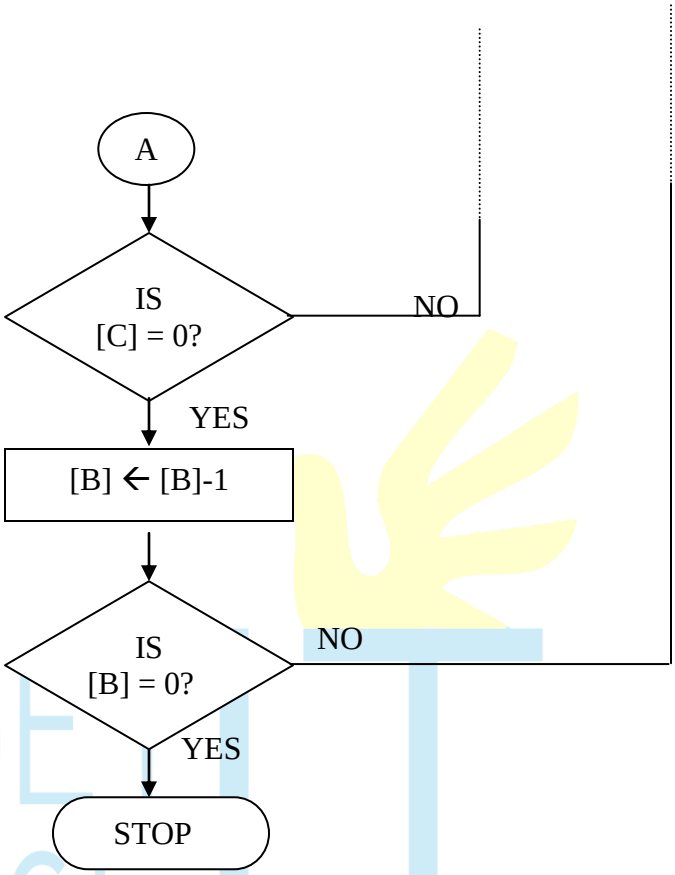
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is smaller than second then I interchange the number.
3. If the first number is larger, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

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.

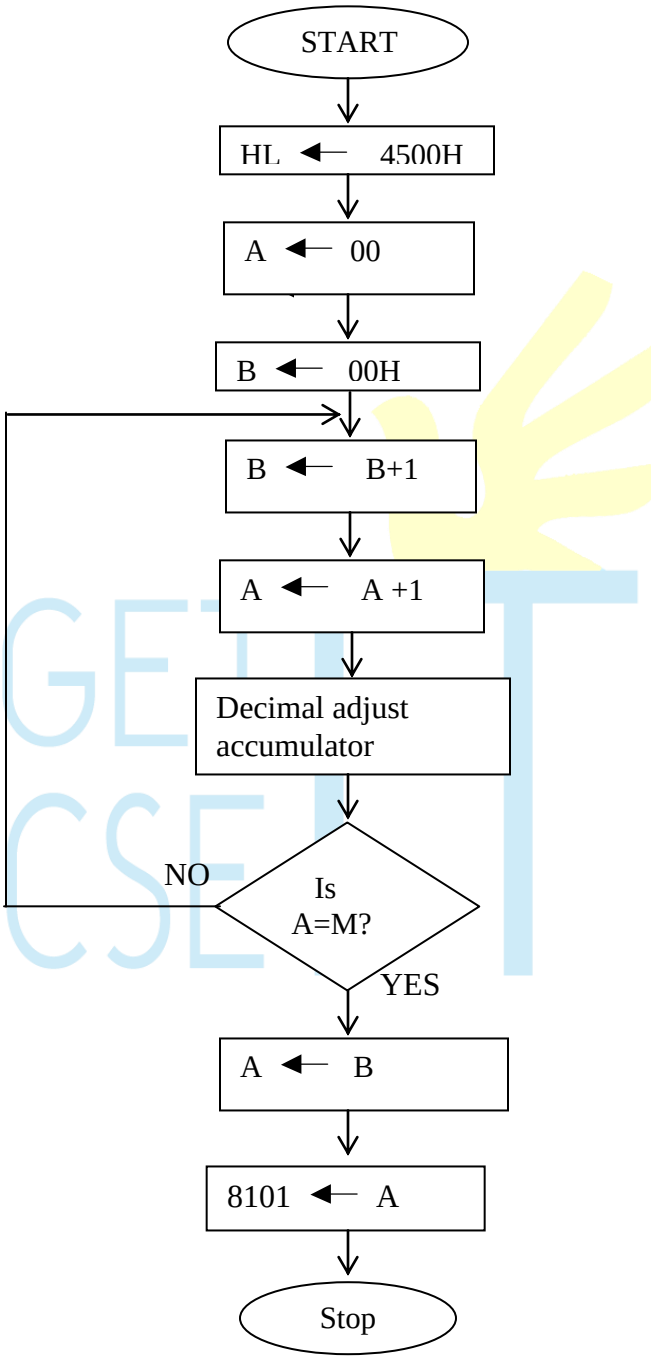
ALGORITHM:

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.

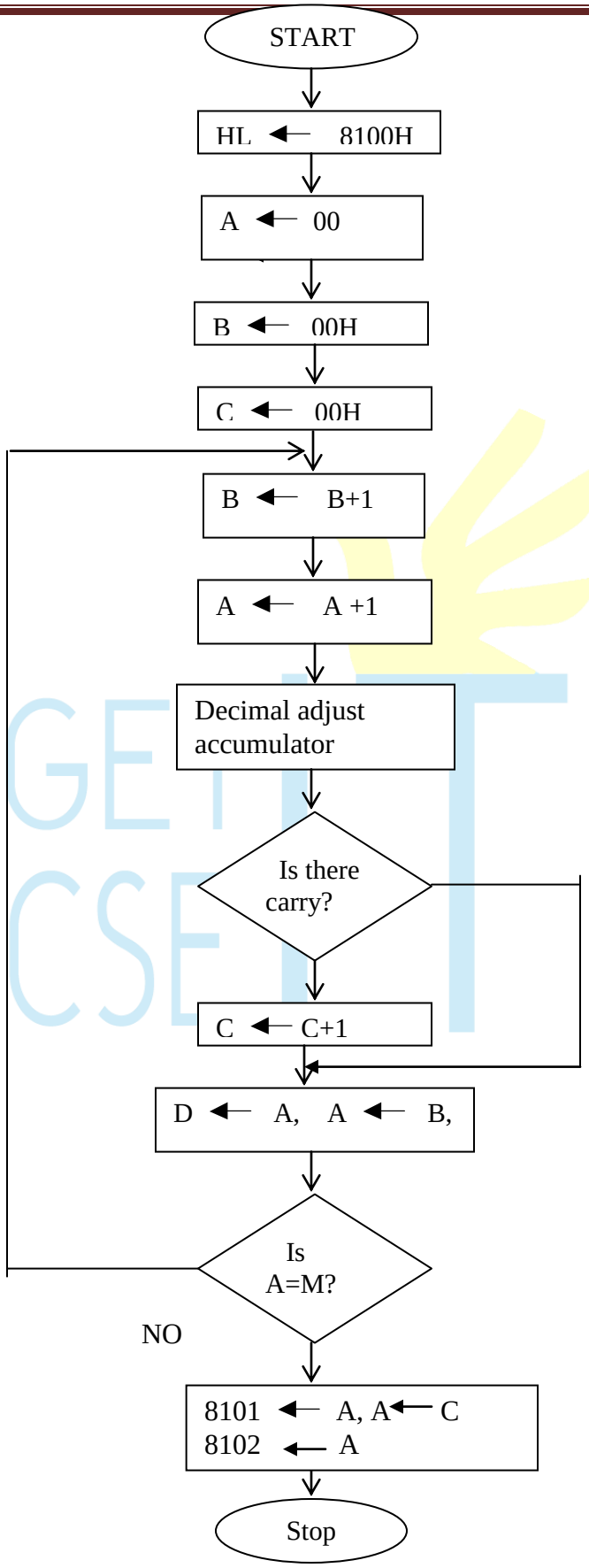
ALGORITHM:

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	

8(C). CODE CONVERSION – BCD TO HEX

AIM:

To convert a given BCD number to hexadecimal.

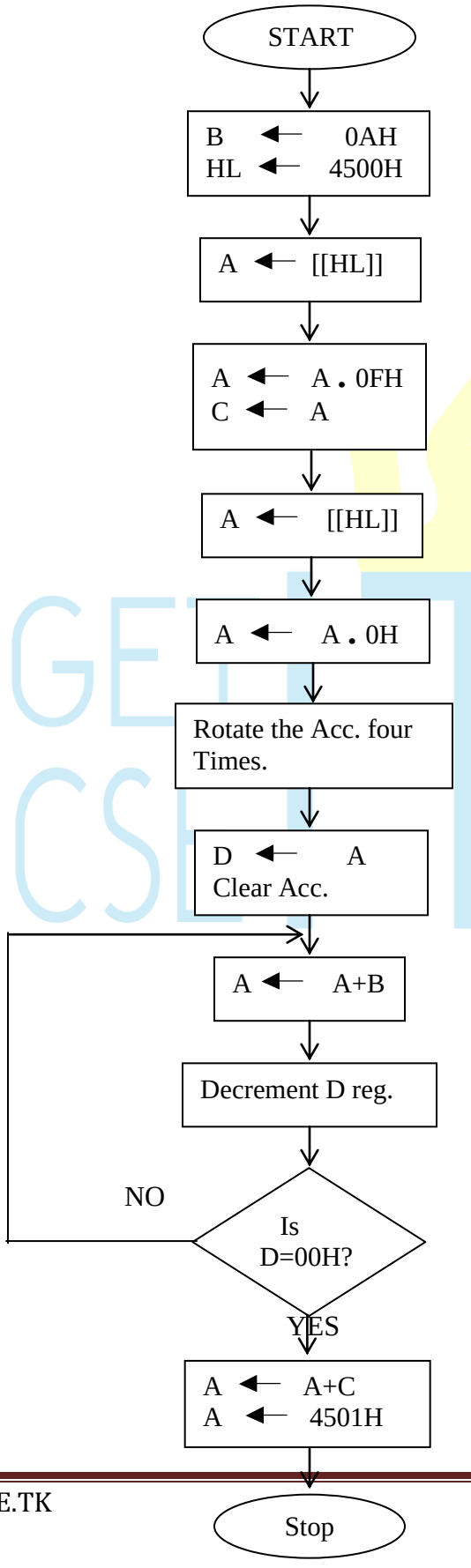
ALGORITHM:

1. Initialize the memory location to the data pointer.
2. Get the BCD number from memory and separate LSB and MSB digits.
3. Multiply MSB No. of BCD to 0AH times and add the LSB to the resultant.
4. Store the resultant in memory location.

RESULT:

Thus an ALP program for conversion of BCD to HEX was written and executed.

FLOWCHART:


START
B ← 0AH
HL ← 4500H
A ← [[HL]]
A ← A . 0FH
C ← A
A ← [[HL]]
A ← A . 0H
Rotate the Acc. four Times.
D ← A
Clear Acc.
A ← A+B
Decrement D reg.
NO
Is D=00H?
YES
A ← A+C
A ← 4501H
Stop

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100	06	START	MVI	B, 0AH	Move OAH to B reg.
4101	00				
4102	21		LXI	H, 4500H	Load the BCD no. in Acc. through memory pointer.
4103	00				
4104	45				
4105	7E		MOV	A, M	
4106	E6		ANI	0FH	Mask MSB of BCD and store LSB in reg. C
4107	0F				
4108	4F		MOV	C, A	
4109	7E		MOV	A, M	Mask LSB of BCD and store MSB in Acc.
410A	E6		ANI	F0H	
410B	F0				
410C	0F		RRC		Bring MSB to LSB position and store in reg. D.
410D	0F		RRC		
410E	0F		RRC		
410F	0F		RRC		
4110	57		MOV	D, A	
4111	AF		XRA	A	Clear the Acc.
4112	80	L1	ADD	D	Add MSB of BCD OAH times.
4113	05		DCR	B	
4114	C2		JNZ	L1	
4115	12				
4116	41				
4117	81		ADD	C	Add LSB of BCD to sum.
4118	32		STA	4501H	Store the HEX No. in memory location 4501H.
4119	01				
411A	45				
411B	76		HLT		Stop the program execution.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4500		4501	

8(D). CODE CONVERSION –HEX TO BCD

AIM:

To convert a given number hexadecimal to BCD

ALGORITHM:

1. Initialize the memory location to the data pointer.
2. Get the HEX number from memory.
3. Initialize the memory to store the output.
4. Subtract the given HEX No. by 64H(100_{BCD}) .Repeat the subtraction with the resultant & 64H and keep count until there is a carry.
5. Store the count, which is MSB of BCD in a memory location.
6. Subtract the 0AH(10_{BCD}) from the result of the previous step. Repeat the subtraction with the resultant & 0AH and keep count until there is a carry.
7. Store the count, which is next significant bit of BCD in next memory location.
8. Store the result of step 6, which is LSB of BCD in next memory location.

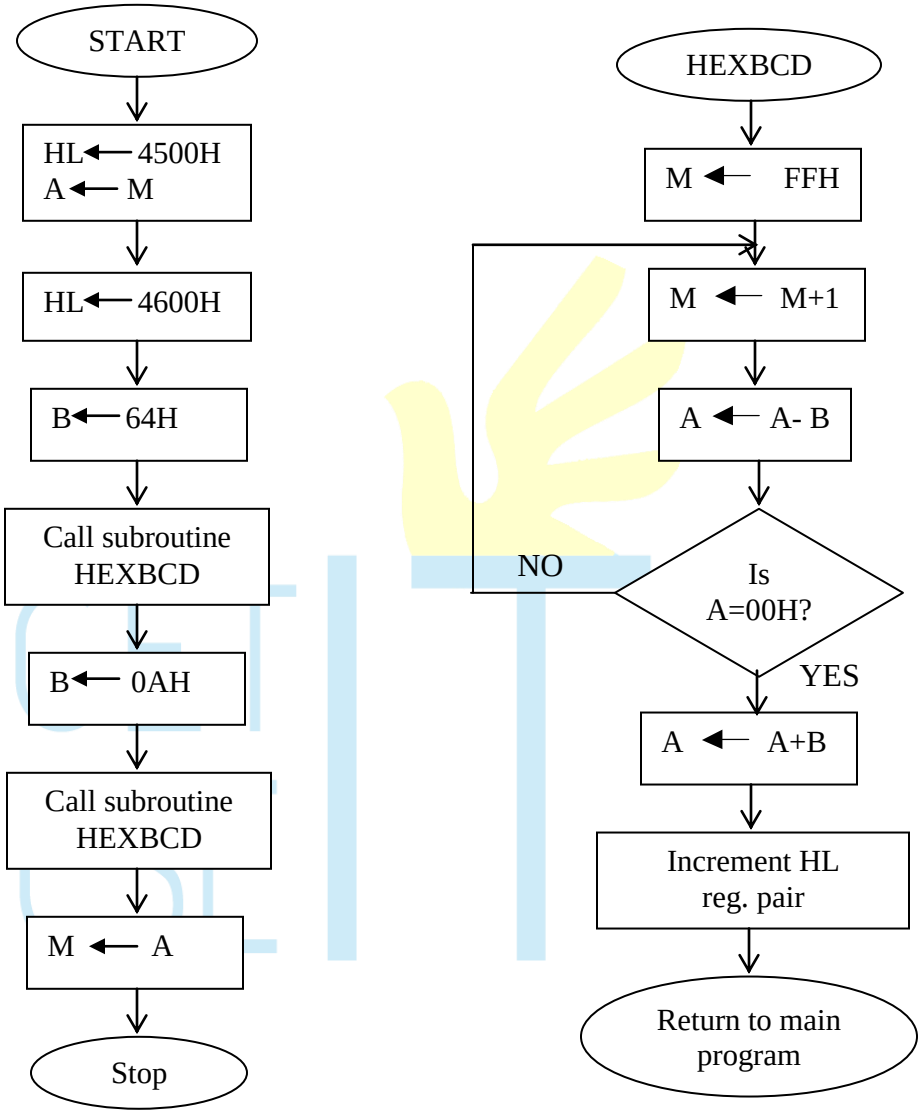
OBSERVATION

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4500		4600	
		4601	
		4602	

RESULT:

Thus an ALP program for conversion of HEX to BCD was written and executed.

FLOW CHART:


START
HL ← 4500H
A ← M
HL ← 4600H
B ← 64H
Call subroutine HEXBCD
B ← 0AH
Call subroutine HEXBCD
M ← A
Stop
HEXBCD
M ← FFH
M ← M+1
A ← A- B
Is A=00H?
NO
YES
A ← A+B
Increment HL reg. pair
Return to main program

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4100	21	START	LXI	H,4500	Load the BCD no. in Acc. through memory pointer.
4101	00				
4102	45				
4103	7E		MOV	A, M	
4104	21		LXI	H, 4600H	Load the memory address 4600H in HL reg. pair
4105	00				
4106	46				
4107	06		MVI	B, 64H	Move data 64H to reg. B.
4108	64				
4109	CD		CALL	HEXBCD	Call the subroutine HEXBCD
410A	1A				
410B	41				
410C	06		MVI	B, 0AH	Move data 0AH to reg. B
410D	0A				
410E	CD		CALL	HEXBCD	Call the subroutine HEXBCD
410F	1A				
4110	46				
4111	77		MOV	M,A	Move the content of Acc. to memory.
4112	76		HLT		Stop the program execution.

SUBROUTINE:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
411A	36	HEXBCD	MVI	M, FFH	Move data FFH to memory.
411B	FF				
411C	34	L1	INR	M	Increment the memory pointer.
411D	90		SUB	B	Subtract the content of B reg. from that of Acc.
411E	D2		JNC	L1	If the content of Acc. is zero go to the instruction labeled L1.
411F	1C				
4110	41				
4111	80		ADD	B	Add the content of Acc. with that of reg. B
4112	23		INX	H	Increment HL reg. Pair.
4113	C9		RET		Return to main program.

8(E) ASCII TO BINARY CONVERSION

AIM:

To convert ASCII number into its binary equivalent.

ALGORITHM:

1. Get the ASCII number in accumulator
2. Check whether the ASCII code is less than 40H.
3. If it is less than 40H subtract 30H from it to get its binary equivalent (because ASCII codes 30 to 39 represent 0 to 9 in binary and 41 to 46 represent A to F)
4. Otherwise subtract 40H from the ASCII code and add 09H to it to get its binary equivalent.
5. Store the result from the ACC in a memory location.
6. Stop the execution.

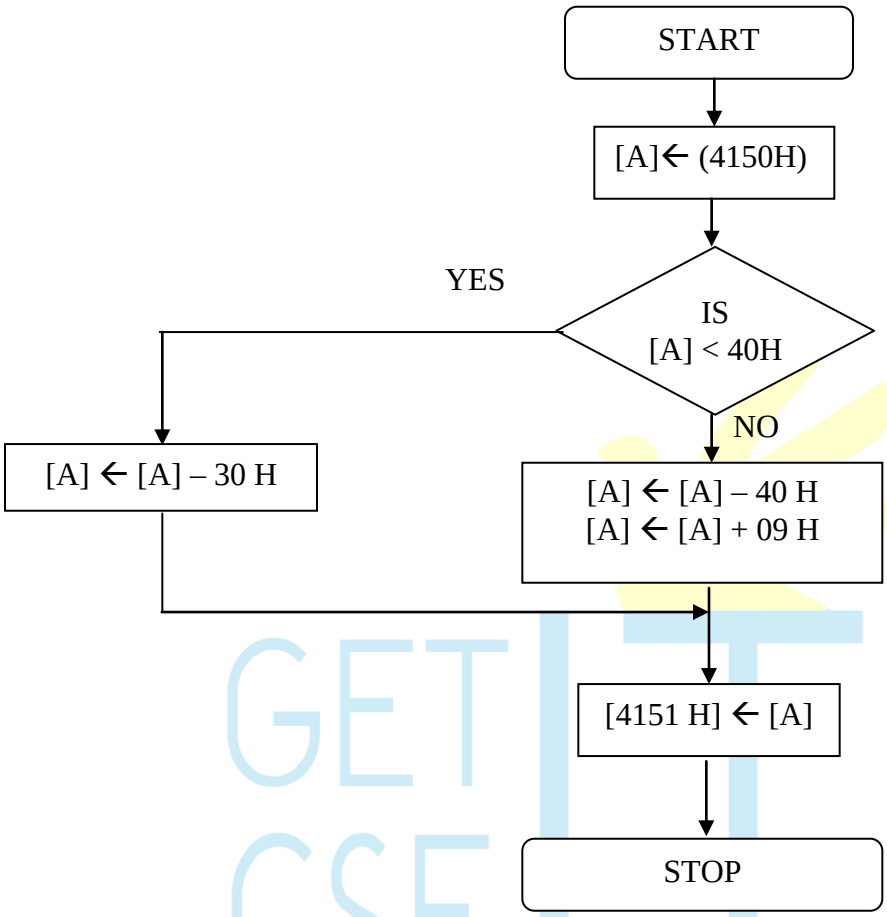
PROGRAM

MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS
4100	3A		LDA	4150H	Get the ASCII code in ACC
4101	50				
4102	41				
4103	FE		CPI	40H	Compare the ASCII code with 40H.
4104	40				
4105	DA		JC	LOOP	If the ASCII code is less than 40H go to the instruction labeled LOOP.
4106	11				
4107	41				
4108	D6		SUI	40	Otherwise subtract 40H form the ASCII code and add 09H to it.
4109	40				
410A	C6		ADI	09	
410B	09				
410C	32	LOOP1	STA	4151H	Store the binary value in 4151H (memory location)
410D	51				
410E	41				
410F	76		HLT		Stop the execution
4111	D6	LOOP	SUI	30	Subtract 30H from the ASCII code
4112	30				
4113	C3		JMP	LOOP1	Go to the instruction labeled LOOP1.
4114	0C				
4115	41				

CONCLUSION:

Thus an ALP for converting an ASCII code to binary was written and executed.

FLOWCHART



OBSERVATION:

	INPUT		OUTPUT	
S. No	ADDRESS	DATA	ADDRESS	DATA
1.	4150H	31H	4151H	01H
2.	4150H	39H	4151H	09H
3.	4150H	42H	4151H	0BH
4.	4150H	46H	4151H	OFH

EXERCISE:

Write an ALP to convert an ASCII code into binary using subroutine.

8(F) BINARY TO ASCII CONVERSION

PROBLEM STATEMENT:

Write an ALP to convert a binary data to its equivalent ASCII code.

ALGORITHM:

Step 1: Get the binary data in Acc and Store it in another reg. (say B)
Step 2: Mask the upper nibble.
Step 3: Call the subroutine "ASCII" to get the ASCII code for lower nibble.
Step 4: Store the Acc contents in memory location.
Step 5: Get the binary data in Acc from Reg B.
Step 6: Mask the lower nibble and move the upper nibble to the lower nibble position to get the ASCII code for upper nibble.
Step 7: Call the subroutine "ASCII" to get the ASCII code for upper nibble and store the Acc contents in another memory location.
Step 8: Stop the execution.

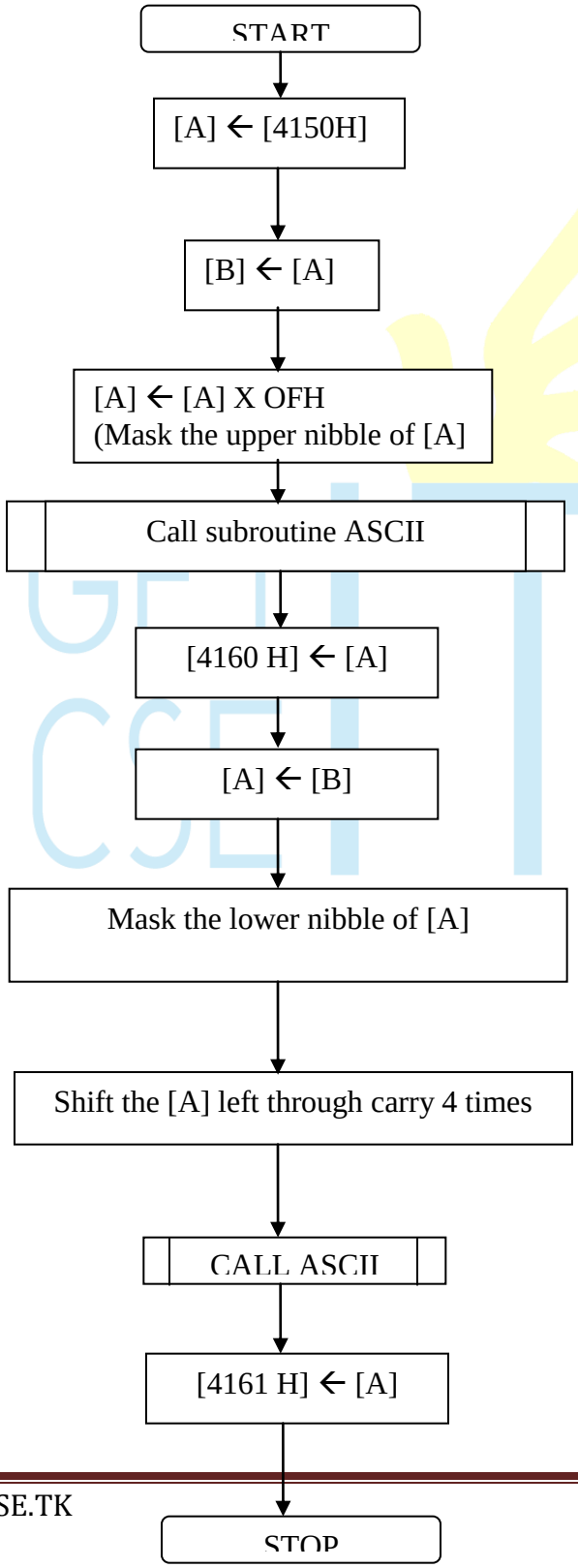
Algorithm for subroutine ASCII:

Step 1: Compare the Acc contents with 0A H.
Step 2: If the Acc contents are lesser than 0A H then add 30H to it and return to the main program.
Step 3: Otherwise add 37H to it and return to the main program.

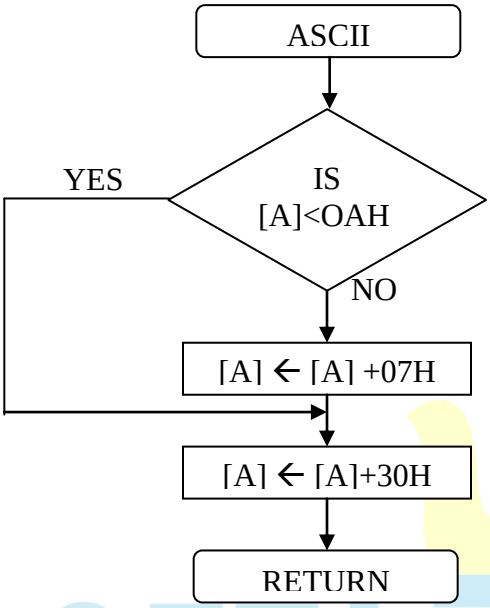
SUBROUTINE PROGRAM

MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS
4180	FE	ASCII	CPI	0A	Compare the Acc contents with 0AH
4181	0A				
4182	DA		JC	LOOP	If there is carry go to instruction labeled "LOOP"
4183	87				
4184	41				
4185	C6		ADI	07H	Otherwise add 07H to acc contents
4186	07				
4187	C6	LOOP	ADI	30H	Add 30H to Acc contents
4188	30				
4189	C9		RET		Return to the main program

FLOWCHART


START
[A] ← [4150H]
[B] ← [A]
[A] ← [A] X OFH
(Mask the upper nibble of [A]
Call subroutine ASCII
[4160 H] ← [A]
[A] ← [B]
Mask the lower nibble of [A]
Shift the [A] left through carry 4 times
CALL ASCII
[4161 H] ← [A]
STOP

SUBROUTINE:



OBSERVATION:

	INPUT		OUTPUT	
S. No	ADDRESS	DATA	ADDRESS	DATA
1.	4150H	05H	4160H	35H
			4161H	30H
2.	4150H	BEH	4160H	45H
			4161H	42H

CONCLUSION:

Thus an ALP to convert binary data to its ASCII equivalent was written and executed.

PROGRAM

MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS
4100	3A		LDA	4150H	Get the binary data in
4101	50				Acc and store it in
4102	41				Reg. B.
4103	47		MOV	B, A	
4104	E6		ANI	OFH	Mask the upper
4105	OF				nibble
4106	CD		CALL	ASCII	Call the subroutine
4107					ASCII to get the
4108					ASCII code for the lower nibble.
4109	32		STA	4160H	Store the ASCII code
410A	60				in 4160H
410B	41				
410C	78		MOV	A, B	Get the binary data in Acc.
410D	E6		ANI	F0H	Mask the lower
410E	F0				nibble
410F	07		RLC		Rotate the contents
4110	07		RLC		left through carry 4
4111	07		RLC		times to move the
4112	07		RLC		upper nibble to lower nibble position
4113	CD		CALL	ASCII	Call the subroutine
4114					ASCII to get the
4115					ASCII code for upper nibble.
4116	32		STA	4161H	Store the ASCII code
4117					in 4161H.
4118					
4119	76		HLT		Stop the execution.

9(A) BCD ADDITION

AIM:

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

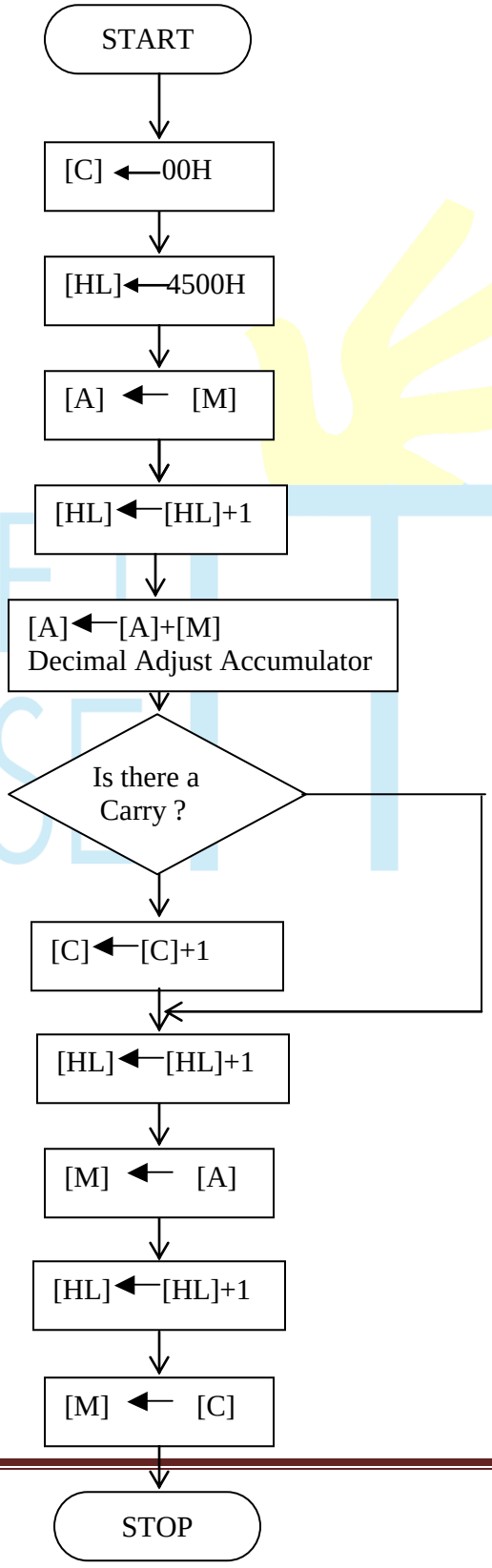
ALGORITHM:

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.

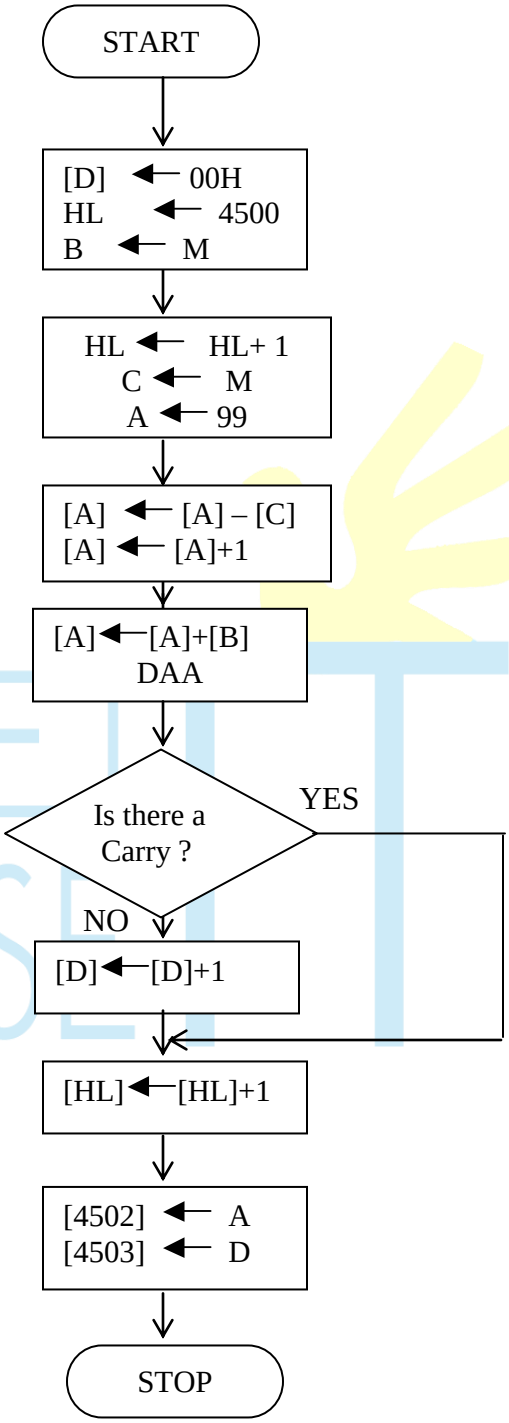
ALGORITHM:

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.

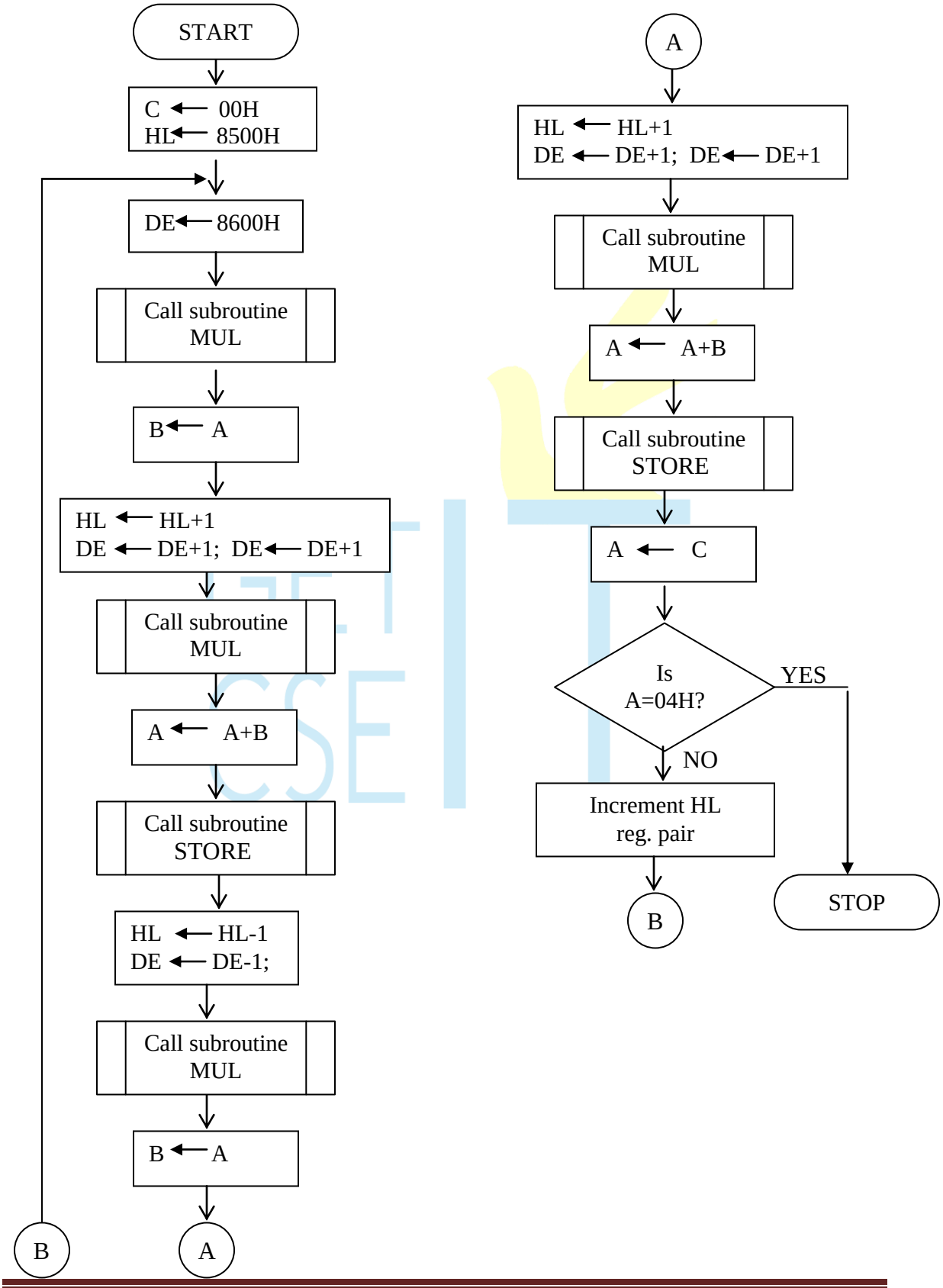
ALGORITHM:

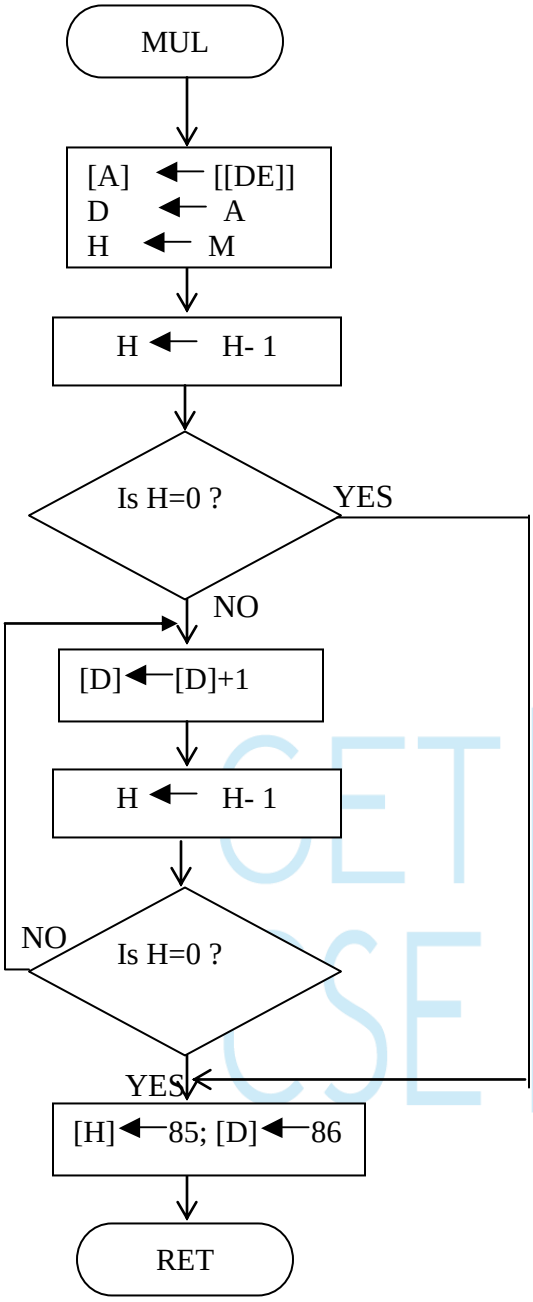
1. Load the 2 input matrices in the separate address and initialize the HL and the DE register pair with the starting address respectively.
2. Call a subroutine for performing the multiplication of one element of a matrix with the other element of the other matrix.
3. Call a subroutine to store the resultant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

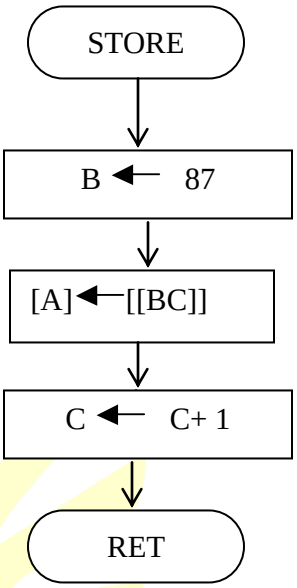
RESULT:

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
NO
Is H=0 ?
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	

EXPERIMENTS– 8086 PROGRAMS

11 . Simple Arithmetic Operation

I. 8-BIT ADDITION

PROBLEM STATEMENT:

Write a program to add the given two 8-bit Nos. in 8086μp.

ALGORITHM:

1. Get the addend and augend.
2. Initialize BL register for carry.
3. Add addend and augend.
4. If there is carry, increment DX register and go to step6 or else directly go to step6.
5. Initialize the memory pointer to output location
6. Store the result & carry in consecutive memory locations.
7. Stop the program execution.

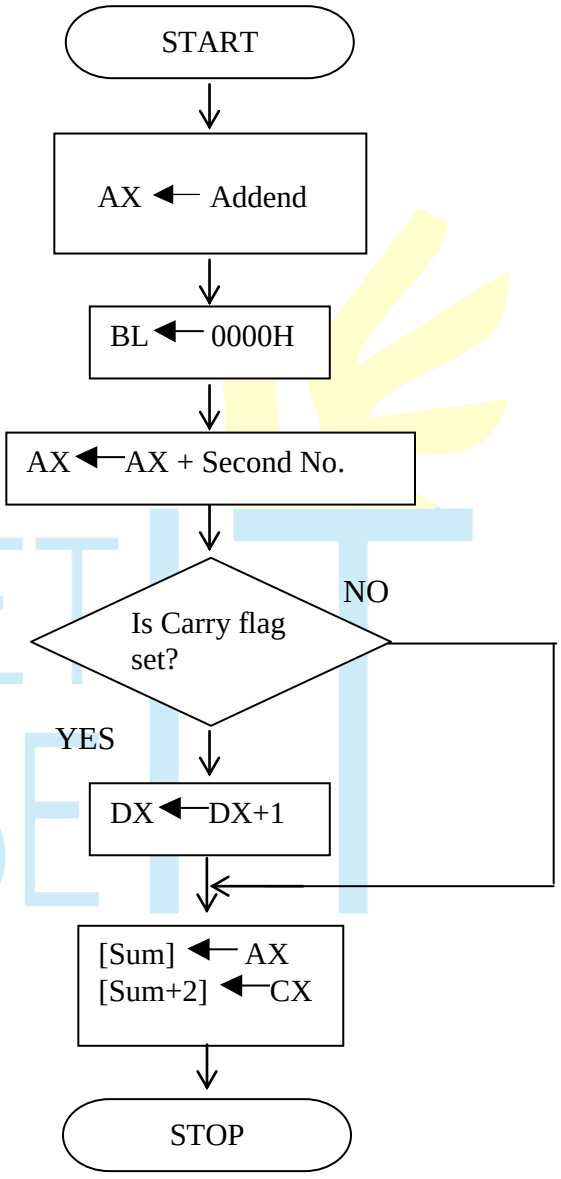
CONCLUSION:

Thus addition of two 8-bit numbers is performed.

EXERCISE:

Write an ALP using INTEL8086 mnemonics to add any two 32-bit numbers.

FLOWCHART:


START
AX ← Addend
BL ← 0000H
AX ← AX + Second No.
Is Carry flag set?
NO
YES
DX ← DX+1
[Sum] ← AX
[Sum+2] ← CX
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
1000			MOV	BL, 00	Clear C reg.
1001					
1002					Move 2000 address into AX register.
1003			MOV	AX,[2000]	
1004					
1005					Move 2002 address into CX register.
1006					
1007			MOV	CX,[2002]	
1008					
1009					
100A					
100B			ADD	AX,CX	Addition
100C					
100D			JNC	LOOP	
100E					
100F			INC	BL	Increment BL register pair
1010					
1011		LOOP			
1012					
1013					Move result into 2102
1014					
1015			MOV	[2102],BL	
1016					
1017					
1018					
1019			HLT		

II. 8-BIT SUBTRACTION

AIM:

Write a program to subtract given two, 8 bit numbers.

ALGORITHM:

1. Get the minuend and subtrahend.
2. Compare the minuend and subtrahend. If minuend is lesser than subtrahend, interchange the numbers and increment Dx register.
3. Subtract subtrahend from minuend.
4. Initialize the memory pointer to output memory location.
5. Store the results in two memory locations and DX register content in the next memory location.
6. Stop the program execution.

CONCLUSION:

Thus, subtraction of two 8-bit numbers was performed.

EXERCISE:

1. Write an ALP to subtract any two 32-bit numbers using INTEL8086 mnemonics.
2. Write an ALP to subtract any two 16-bit numbers.

(HINT: If subtrahend is greater than minuend, take 2's complement of the result and indicate it by putting 01 in DL register.)

FLOWCHART:

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
1000			MOV	BL, 00	Clear C reg.
1001					
1002					Move 2000 address into AX register.
1003			MOV	AX,[2000]	
1004					
1005					Move 2002 address into CX register.
1006					
1007			MOV	CX,[2002]	
1008					
1009					
100A					
100B			SUB	AX,CX	Sbbtraction
100C					
100D			JNC	LOOP	
100E					
100F			INC	BL	Increment BL register pair
1010					
1011			NEG	AX	
1012					
1013		LOOP	MOV	[2100],AX	Move result into 2102
1014					
1015					
1016					
1017			MOV	[2102],BL	Move result into 2102
1018					
1019					
101A					
101B			HLT		

III. 8-BIT MULTIPLICATION

PROBLEM STATEMENT:

Write a program to multiply two, 8-bit numbers using 8086.

ALGORITHM:

1. Get the multiplicand and multiplier
2. Multiply the multiplicand with multiplier using repeated addition method.
3. Initialize the memory pointer to output memory location.
4. Store the results in memory locations.
5. Stop the program execution.

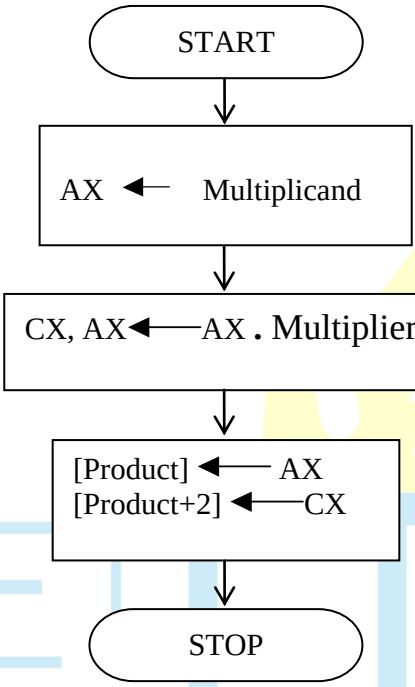
CONCLUSION:

Thus, multiplication of two, 8-bit numbers is performed using INTEL 8086 Mnemonics.

EXERCISE:

Write an ALP using INTEL8086 mnemonics to multiply two signed 16-bit numbers.

FLOWCHART:



PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
1000			MOV	BL, 00	Clear C reg.
1001					
1002					Move 2000 address into AX register.
1003			MOV	DX,0000	
1004					
1005					Move 2002 address into CX register.
1006					
1007			MOV	AX,[2000]	
1008					
1009					
100A					
100B			MOV	CX,[2002]	Sbbtraction
100C					
100D					
100E					
100F			MUL	CX	Increment BL register pair

1010					
1011			JNC	LOOP	
1012					
1013			INC	BL	Move result into 2102
1014					
1015		LOOP	MOV	[2100],AX	
1016					
1017					Move result into 2102
1018					
1019			MOV	[2102],DX	
101A					
101B					
101C					
101D			MOV	[2104],BL	
101E					
101F					
1020					
1021			HLT		

IV. 16-BIT DIVISION

PROBLEM STATEMENT:

Write a program to Divide two, 8-bit numbers using 8086.

ALGORITHM:

1. Get the dividend and divisor.
2. Divide dividend by divisor.
3. Initialize the memory pointer to output memory location.
4. Store the results in memory locations.
5. Stop the program execution.

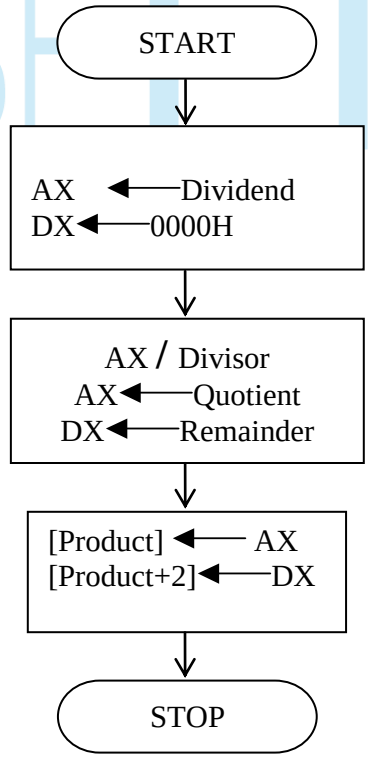
CONCLUSION:

Thus, division of two, unsigned 8-bit numbers is performed using INTEL 8086 Mnemonics.

EXERCISE:

Write an ALP using INTEL8086 mnemonics to divide two signed 16-bit numbers.

FLOWCHART:



PROGRAM

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
1000			MOV	DX,0000	Clear C reg.
1001					
1002					Move 2000 address into AX register.
1003					
1004			MOV	AX,0008 H	
1005					Move 2002 address into CX register.
1006					
1007					
1008			MOV	CX,0004 H	
1009					
100A					
100B					
100C			DIV	CX	
100D					
100E			MOV	[2100],AX	
100F					Increment BL register pair
1010					
1011					
1012			MOV	[2102],DX	
1013					Move result into 2102
1014					
1015					
1016			HLT		

10.3 X 3 MATRIX ADDITION

AIM:

To perform the 3 x 3 matrix addition.

ALGORITHM:

1. Load the 3 input matrices in the separate address and initialize the AX and the BX register pair with the starting address respectively.
2. Call a subroutine for performing the addition 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 3 x 3 matrix multiplication is performed and the result is stored at memory locations.

PROGRAM

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
1000			MOV	CL,09	Clear C reg.
1001					
1002					Move 2000 address into AX register.
1003			MOV	SI,1500	
1004					
1005					Move 2002 address into CX register.
1006					
1007			MOV	DI,2500	
1008					
1009					
100A					
100B		LOOP	MOV	AL,[SI]	
100C					
100D			MOV	BL, [DI]	
100E					
100F			ADD	AL ,BL	Increment BL register pair
1010					

1011			MOV	[DI], AL	
1012					
1013			INC	SI	Move result into 2102
1014			INC	DI	
1015			DEC	CL	
1016			HLT		
1017			JNZ	LOOP	
1018					
1019			HLT		

12 . BIOS/DOS CALLS – DISPLAY

AIM:

To display a message on the CRT screen of a microcomputer using DOS calls.

ALGORITHM:

1. Initialize the data segment and the message to be displayed.
2. Set function value for display.
3. Point to the message and run the interrupt to display the message in the CRT.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
MSG DB 0DH, 0AH, "GOOD MORNING" , ODH, OAH, "$"
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        MOV AH, 09H
        MOV DX, OFFSET MSG
        INT 21H
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

RESULT:

A message is displayed on the CRT screen of a microcomputer using DOS calls

13. BIOS/DOS CALLS – FILE MANIPULATION

AIM:

To open a file using DOS calls.

ALGORITHM:

1. Initialize the data segment, file name and the message to be displayed.
2. Set the file attribute to create a file using a DOS call.
3. If the file is unable t o create a file display the message

PROGRAM:

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT

FILENAME DB “SAMPLE.DAT”, “$”

MSG DB 0DH, 0AH, “FILE NOT CREATED”, ODH, OAH, “$”

DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV DX, OFFSET FILENAME
            MOV CX, 00H
            MOV AH, 3CH
            INT 21H
            JNC LOOP1
            MOV AX, DATA
            MOV DS, AX
            MOV DX, OFFSET MSG
            MOV AH, 09H
            INT 21H
LOOP1       MOV AH, 4CH
            INT 21H

CODE ENDS
END START
```

RESULT : A file is opened using DOS calls.

14. BIOS/DOS CALLS – DISK INFORMATION

AIM:

To display the disk information.

ALGORITHM:

1. Initialize the data segment and the message to be displayed.
2. Set function value for disk information.
3. Point to the message and run the interrupt to display the message in the CRT.

PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
MSG DB 0DH, 0AH, "GOOD MORNING" , ODH, OAH, "$"
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        MOV AH, 36H
        MOV DX, OFFSET MSG
        INT 21H
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

RESULT:

The disk information is displayed.

15 STRING MANIPULATION

I. 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

II.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

III. 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

IV.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

16. INTERFACING 8255 WITH 8085

AIM:

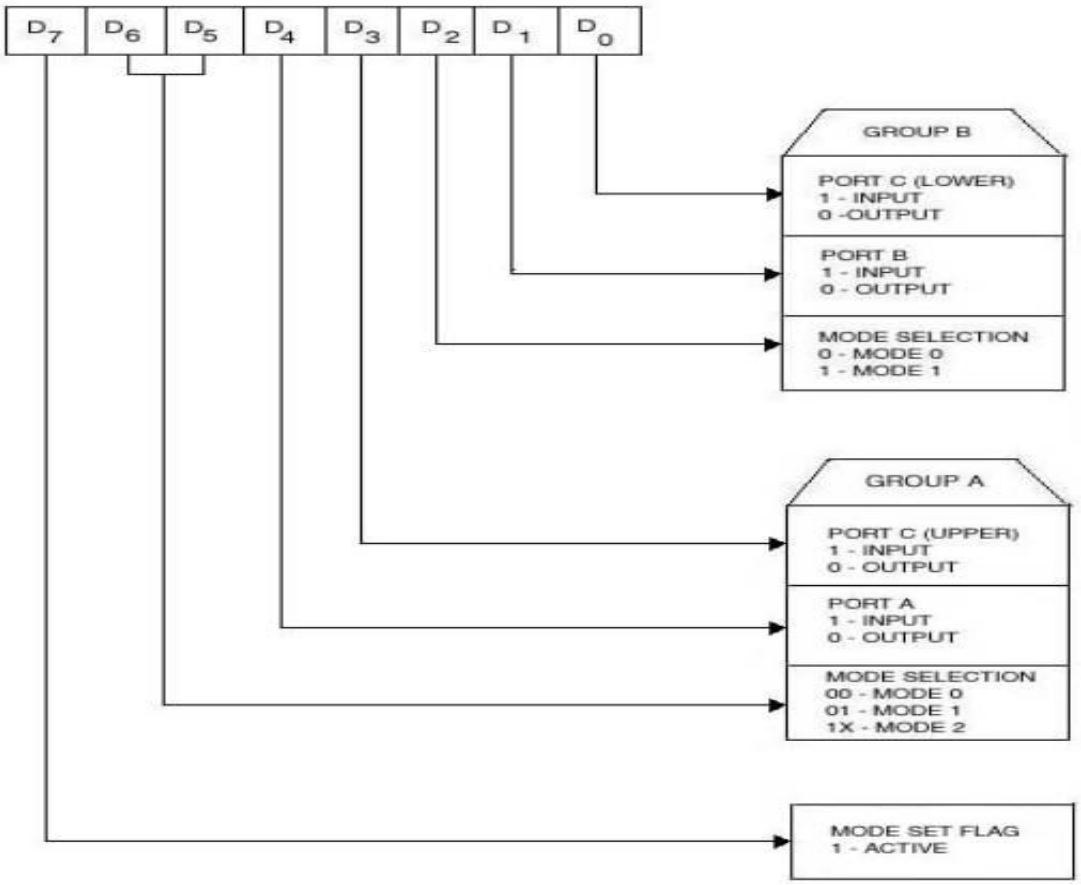
To interface programmable peripheral interface 8255 with 8085 and study its characteristics in mode0,mode1 and BSR mode.

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.

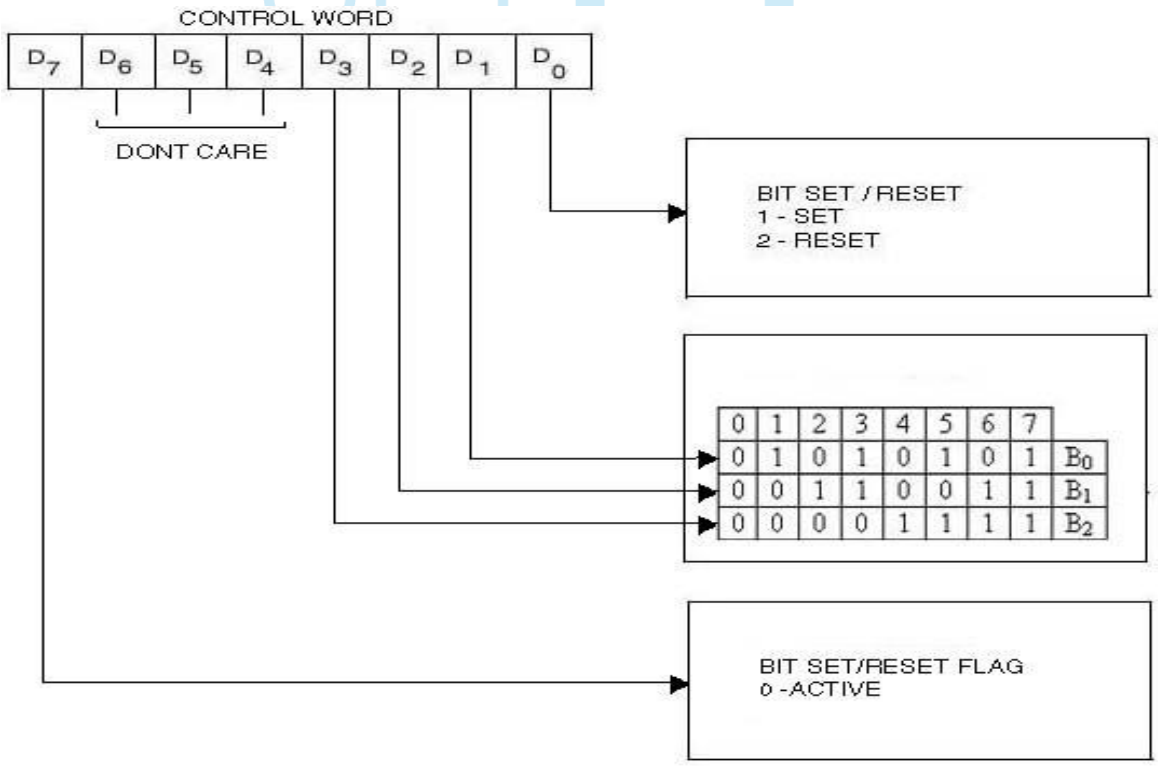
ISR (Interrupt Service Routine)

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.

17. INTERFACING 8253 TIMER WITH 8085

Interfacing 8253 Programmable Interval Timer with 8085 μp

AIM:

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

APPARATUS REQUIRED:

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 retriggerable.

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.

18. INTERFACING 8279 WITH 8085

Aim:

To interface 8279 programmable Keyboard/Display Controller to 8085 μp

Apparatus Required:

8085 μp , 8279 Interface board , Power supply , vxt parallel bus

Theory:

The Intel 8279 is responsible for debouncing of the keys, coding of the keypad matrix and refreshing of the display elements in a microprocessor based development system. Its main features are :

- Simultaneous keyboard and display operation.
- 3 Input modes such as scanned keyboard mode, scanned sensor mode and strobed input entry mode.
- 2 output modes such as 8 or 16 character multiplexed displays , right entry or left entry display formats.
- Clock prescaler
- Programmable scan timing
- 2 key lock_ *out or N*_key roll_over with contact debounce
- Auto increment facility for easy programming.

Program 1:

To initialize 8279 and to display the character "A" in the first digit of the display.

```
MVI   A,00      ; mode and display set
OUT   C2
MVI   A,CC      ; clear display
OUT   C2
MVI   A,90      ; write display RAM
OUT   C2
MVI   A,88      ; Display 'A'
OUT   C0
MVI   A,FF      ; blank the rest of the display
OUT   C0
OUT   C0
OUT   C0
OUT   C0
OUT   C0
HLT
```

Program 2:

To read a key and store the key code in memory location 4200H

```
IN      C2          ; FIFIstatus
ANI     07          ; check for a key closure
JZ      4150
MVI     A,40        ; set 8279 for a read
OUT     C2          ; of FIFO RAM
IN      C0
STA     4200        ; keycode at 4200
HLT
```

Procedure:

Enter the above two programs from the address specified and execute it.
The display is "A" in the first digit and the rest are left blank for program-1.
If a key closure is encountered , read the data from FIFO RAM , and store this data(key code) in memory location 4200H.

Exercise:

Program 8279 to display the rolling message 'HELP US' in the display.

Result:

Thus the 8279 was interfaced to 8085 μp to interface Hex keyboard and 7-Segment Display.

19. INTERFACING 8251 WITH 8085

Communication Between two 8085 Microprocessors

Aim:

To transmit and receive a Character between two 8085 μps using 8251A

Apparatus Required:

8085 μp Kit – 2 No.s , RS 232C cable , Power supply – 2 No.s

Theory:

The program first initializes the 8253 to give an output clock frequency of 150KHz at channel 0 which will give a 9600 baud rate of 8251A. Then the 8251A is initialized to a dummy mode command. The internal reset to 8251A is then provided, since the 8251A is in the command mode now. Then 8251A is initialized as follows.

Initializing 8251A using the Mode instruction to the following.

8 bit data
No parity
16x Baud rate factor
1 stop bit
B2 , B1 = 1 , 0
L2 , L1 = 1 ,1
PEN = 0
EP = 0
S2 , S1 = 0 , 1

gives a Mode command word of 4E.

When 8251A is initialized as follows using the command instruction,

Reset Error flags,
Enable transmission and reception,
Make RTS and DTR active low.

EH = 0	SBRK = 0
IR = 0	RxE = 1
RTS = 1	DTR = 1
ER = 1	TxEN = 1

We get a command word of 37

The program after initializing , will read the status register and check for TxEMPTY. If the transmitter buffer is empty then it will send 41 to the serial port and then check for a character in the receive buffer. If some character is present then, it is received and stored at location 4200H.

Program:

```
            ORG     4100H
UATCNT      EQU     05

UATDAT      EQU     04
TMRCNT      EQU     0B
TMRCH0      EQU     08

            MVI     A,36        ;Initialization of 8253
            OUT     TMRCNT
            MVI     A,0A
            OUT     TMRCH0
            XRA     A
            OUT     TMRCH0
            XRA     A           ;Resetting the 8251A
            OUT     UATCNT
            MVI     A,40
            OUT     UATCNT
            MVI     A,4E        ;Initialization of 8251A
            OUT     UATCNT
            MVI     A,37
            OUT     UATCNT
```

Program for Transmitter:

```
LOOP:       IN      UATCNT      ;Check 8251As TxEMPTY
            ANI     04          ;and then send the data 41
            JZ      LOOP
            MVI     A,41
            OUT     UATDAT
```

Program for Receiver:

```
LOOP1:      IN      UATCNT      ;Check 8251As RxRDY and then
            ANI     2           ;get the data and store at 4200
            JZ      LOOP1
            IN      UATDAT
            STA     4200
            HLT
```

Procedure:

Feed the above program in two 8085 μps (One acts as Transmitter and the other acts as Receiver). Execute the two programs simultaneously. Check the Receiver at location 4200H. It ‘s content will be 41.

Exercise:

Write a program to transmit a block of data from transmitter and receive them at the receiver.

Result:

Thus the communication between two microprocessors has been established.

20. 0800 DAC Interfacing to 8085μp

Aim:

To interface 0800 DAC to 8085 μp and generate various waveforms.

Apparatus Required:

8085 μp kit , DAC interface board , VXT parallel bus , power supply , CRO

Theory:

DAC 0800 is a monolithic, high speed, current output Digital to Analog converter. The DAC interface board consists of two 8- bit DAC 0800. Its output voltage variation is between –5V and +5V. The output voltage varies in steps of 10/256 = 0.04 (approx). The digital data input and the corresponding output voltages are presented in the following table.

Input data in Hex	Output Voltage
00	-5.00
01	– 4.96
02	– 4.92
.	
.	
7F	0.00
.	
.	
FD	4.92
FE	4.96
FF	5.00

Address for DAC1 is C0,
and for DAC2 is C8

Program:

To generate square-wave at the DAC2 ouput.

```
        ORG     4100H
START:  MVI     A,00        ; load minimum data
        OUT     C8H
        CALL    DELAY       ; delay subroutine
        MVI     A,FF        ; load maximum data
```

```
                OUT     C8H
CALL    DELAY   ; delay subroutine
                JMP     START

        DELAY:  MVI     B,05
        L1:     MVI     C,FF
        L2:     DCR     C
                JNZ     L2
                DCR     B
                JNZ     L1
                RET
```

Procedure:

Execute the above program and using CRO, verify that the waveform at the DAC2 output is a square wave. Modify the frequency of the square wave , by varying the time delay.

To create a saw-tooth wave at the output of DAC1

```
        ORG     4200H
START:  MVI     A,00
L1:     OUT     C0H
        INR     A
        JNZ     L1
        JMP     START
```

Exercise:

1) Write a program to generate triangular waveform at DAC2 output.
2) Write a program to generate sine-wave at DAC1 ouput.

Result:

Thus the DAC 0800 interface board has been interfaced to 8085μp and various waveforms have been generated.

21.Interfacing 16-channel 0809 8 bit ADC interface board to 8085 μp

Aim:

To interface 16-channel 0809 8 bit ADC interface board to 8085 μp

Apparatus Required:

8085 μp kit , 16-channel 0809 8 bit ADC interface board , VXT parallel bus , power supply.

Theory:

ADC 0809 is a monolithic CMOS device , with an 8 bit analog-to-digital converter , 8 channel multiplexer and microprocessor compatible control logic.In the interface board the channel select address pins ADD A , ADD B , and ADD C are connected to data bus through a latch 74LS174. The buffer 74LS244 which transfers the converted data outputs to data bus is selected when the address is C0h. The I/O address for the latch 74LS174 which latches the data bus to ADD A , ADD B, ADD C and ALE 1 and ALE 2 is C8H. The flip flop 74LS74 which transfers the D0 line status to the SOC pin of ADC 0809 is selected when the address is D0H. The EOC output of ADC1 and ADC2 is transferred to D0 line by means of two tristate buffers. The EOC 1 is selected when the address is D8H and the EOC 2 is selected when the address is E0H.

Program:

```
        ORG     4100H
START:  MVI     A,10        ;Select Channel 0
        OUT     C8H
        MVI     A,18
        OUT     C8H
        HLT
```

Procedure:

The above program selects Channel 0. Execute the program. Start the analog to digital conversion process by pressing the SOC switch. ADC 0809 converts the analog input at channel 0 to a digital value and 74LS374 latches the data to glow the LEDs accordingly. Thus you can see the converted data output.

```
        ORG     4100H
START:  MVI     A,10        ;Select Channel 0
        OUT     C8H
        MVI     A,18H
        OUT     C8H
        MVI     A,01
        OUT     D0H
        XRA     A
```

```
XRA     A
XRA     A
MVI     A,00        ;SOC pulse
OUT     D0H
HLT
```

Procedure:

The above program initiates the analog to digital conversion process by means of software. Execute the program, which converts the anolog input at Channel 0 and displays the output with the LEDs.

Exercise:

Write a program to convert the analog input to digital output by checking EOC pin of ADC 0809 , whether the conversion is over or not.

conclusion:

Thus the ADC interfacing board has been interfaced to 8085 μp.

22. TRAFFIC LIGHT CONTROLLER

PROBLEM STATEMENT:

Write an ALP to control the traffic light signal using the microprocessor 8085.

THEORY:

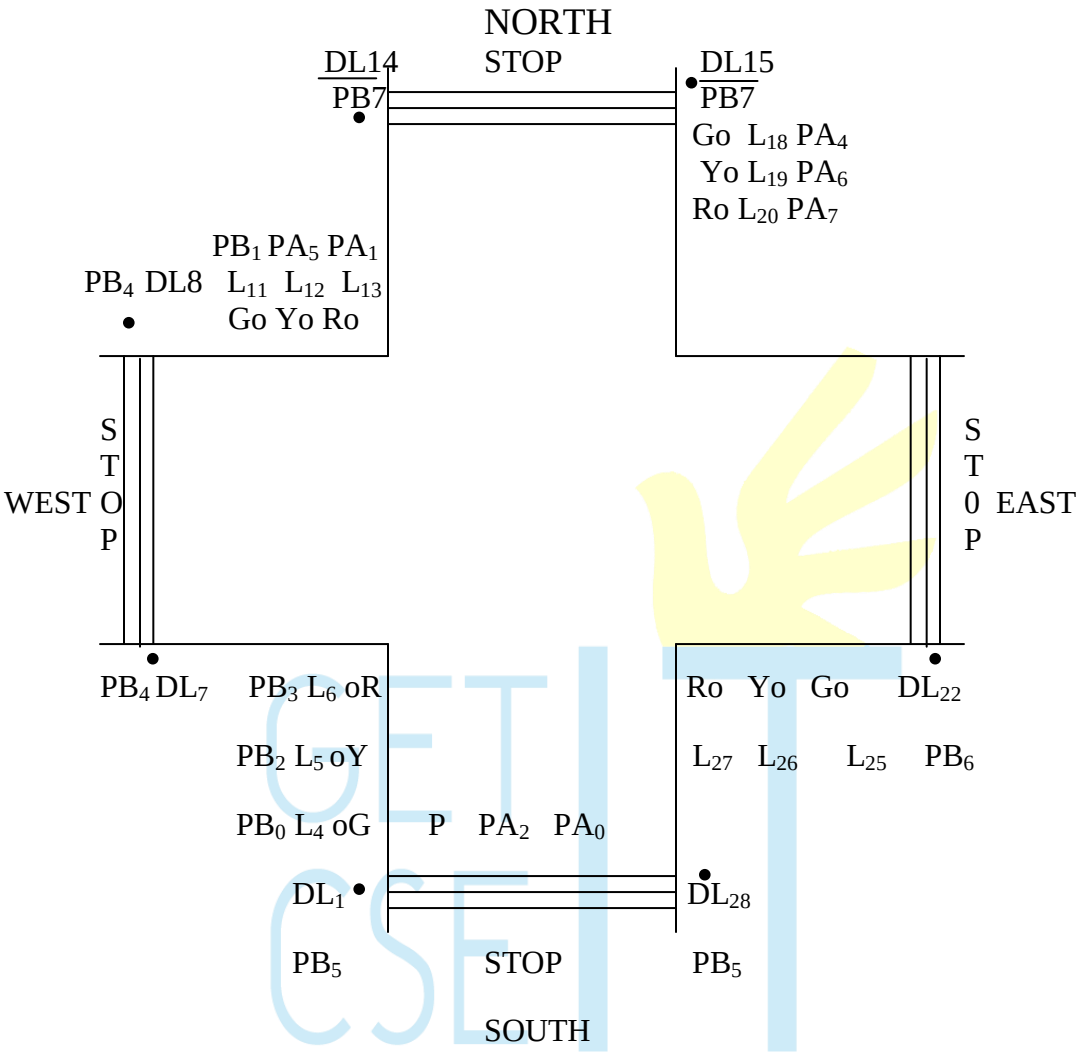
A simple contraption of a traffic control system is shown in the figure where the signaling lights are simulated by the blinking or ON – OFF control of LED's. The signaling lights for the pedestrian crossing are simulated by the ON – OFF control of dual colour LED's.

A model of a four road – four lane junction, the board has green, yellow and red LED's which are the green, orange and red signals of an actual systems. 12 LEDs are used on the board. In addition 8 dual colour LEDs are used which can be made to change either to red or to green.

The control of the LEDs is as follows:

The board communicates with the microprocessor trainer by means of a 26 core cable which is connected to the output pins of any parallel port of trainer.

The outputs (i.e. port) are the inputs to buffers 7406 whose outputs drive the LEDs. The buffered output applied to the cathode of the LEDs decides whether it is ON or OFF.

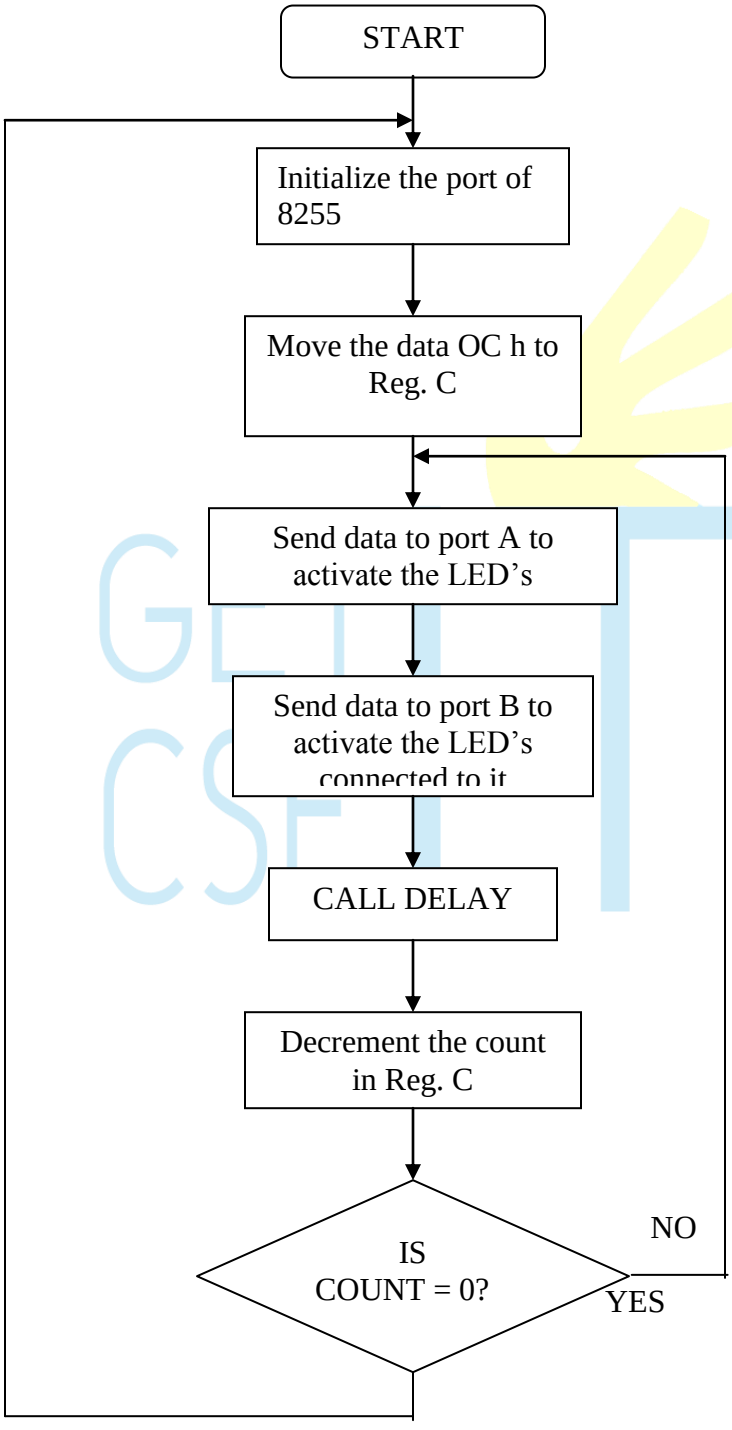


(• ⇒ Dual Colour LED)

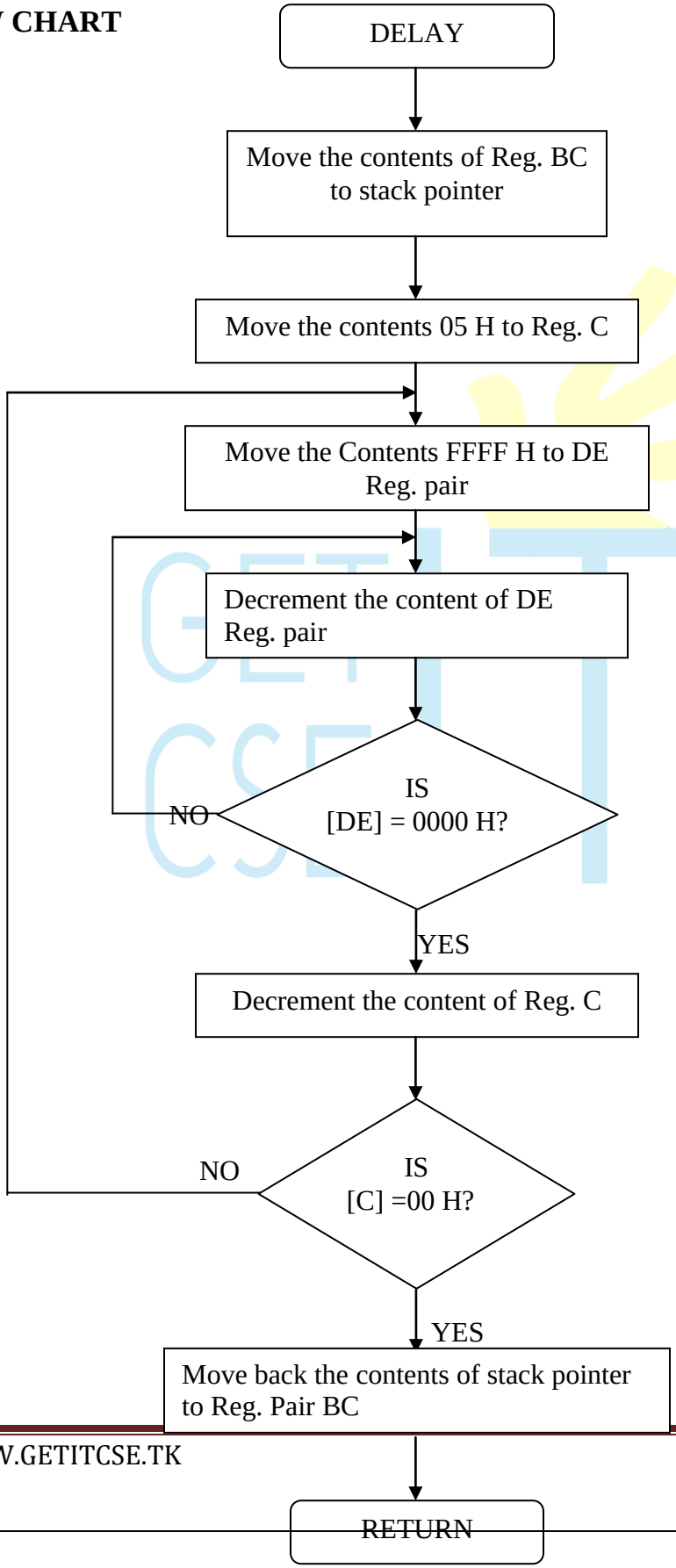
I/O Decoding:

Address	selection
OF H	Control word Register
OC H	Port A
OD H	Port B

FLOW CHART


START
Initialize the port of 8255
Move the data OC h to Reg. C
Send data to port A to activate the LED's
Send data to port B to activate the LED's connected to it
CALL DELAY
Decrement the count in Reg. C
IS COUNT = 0?
NO
YES

FLOW CHART


DELAY
Move the contents of Reg. BC to stack pointer
Move the contents 05 H to Reg. C
Move the Contents FFFF H to DE Reg. pair
Decrement the content of DE Reg. pair
IS [DE] = 0000 H?
NO
YES
Decrement the content of Reg. C
IS [C] =00 H?
NO
YES
Move back the contents of stack pointer to Reg. Pair BC
RETURN

PROGRAM

MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS
4100	21	START	LXI	H, 4500 H	Initialize the
4101	00				HLReg. Pair to
4102	45				4500 H
4103	0E		MVI	C, OC H	
4104	0C				Initialize the count
4105	7E		MOV	A, M	Reg. C to OC H
4106	D3		OUT	OF H	Initialize the ports
4107	0F				of 8255.
4108	23		INX	H	
4109	7E	LOOP1	MOV	A, M	Send the data to
410A	D3		OUT	OC H	switch ON / OFF
410B	0C				The LED's through
410C	23		INX	H	port A.
410D	7E		MOV	A, M	Switch ON / OFF
410E	D3		OUT	OD H	the LED's through
410F	0D				port B
4110	CD				
4111	1B		CALL	DELAY	
4112	41				
4113	23		DELAY	H	Call the subroutine
4114	0D		DCR	C	delay
4115	C2		JNZ	LOOP 1	Get the next data
4116	09				and decrement the
4117	41				count.
4118	C3		JMP	START	If the count is not
4119	00				zero go to
411A	41				instruction labeled
					"LOOP 1"
					Jump
					unconditionally to
					the instruction
					labeled "START"

SUBROTINE

411B	C5	DELAY	PUSH	B	
411C					
411D	0E		MVI	C, 05	Save the contents of BC Reg. Pair to stack pointer. Initialize Reg. C to hold data 05 H.
411E	05				
411F	11	LOOP3	LXI	D, FFFF	
4120	FF				
4121	FF				
	1B	LOOP2	DCX	D	Get the data FFFF H in DE Reg. Pair.
4122					
4123	7A		MOV	A, D	Decrement the content of DE Reg. Pair
	B3		ORA	E	
4124					
4125	C2		JNZ	LOOP 2	Check whether the contents of DE Reg. Pair is zero. If the contents of DE Reg. Pair is not zero go to instruction labeled LOOP 2
4126	21				
	41				
4127	0D		DCR	C	Decrement the content of Reg. C
4128					
4129	C2		JNZ	LOOP 3	If the contents of Reg. C is not zero go to instruction labeled “LOOP 3”
	1E				
412A					
412B	41				Move the contents of stack pointer to BC Reg. Pair
	C1		POP	B	
412C					
	C9		RET		Return to the main program.

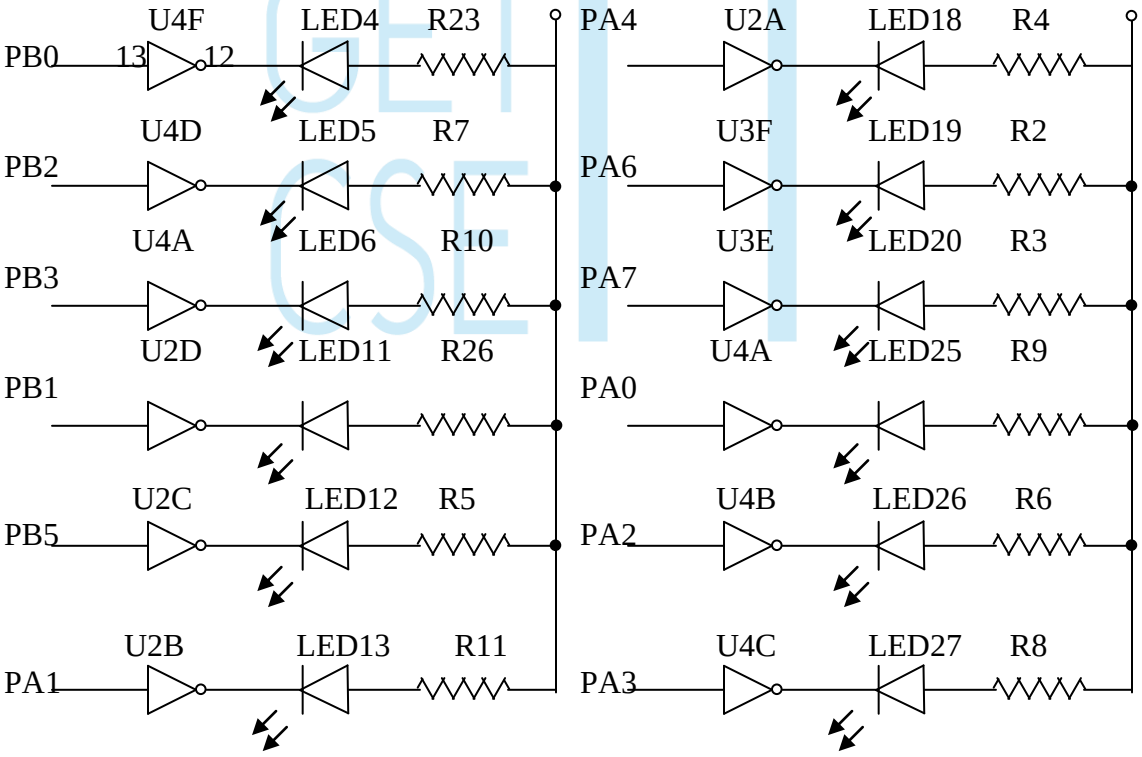
LOOK UP TABLE:

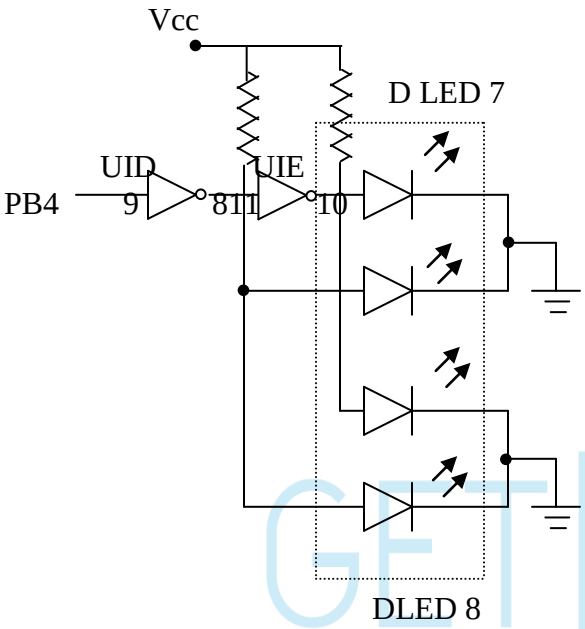
Address	Machine Code	Address	Machine Code
4500	80	450C	B4
4501	1A	450D	88
4502	A1	450E	DA
4503	64	450F	68
4504	A4	4510	D8
4505	81	4511	1A
4506	5A	4512	E8
4507	64	4513	46
4508	54	4514	E8
4509	8A	4515	83
450A	B1	4516	78
450B	A8	4517	86
		4518	74

OBSERVATION:

The traffic controller is simulated.

C I R C U I T D I A G R A M





CONCLUSION:

Thus an ALP of to control the traffic light signal was written and executed.

23. STEPPER MOTOR CONTROL

PROBLEM STATEMENT:

Write an ALP to drive and control a stepper motor.

THEORY:

Stepper motor control is one of the popular applications of microprocessor in control area. Stepper motor are capable of accepting a sequence of pulse from the microprocessor and step accordingly. They are used to control numerical – controlled machines, computer peripheral equipment, business machines, process control etc.

INTERFACE DRIVE CIRCUIT:

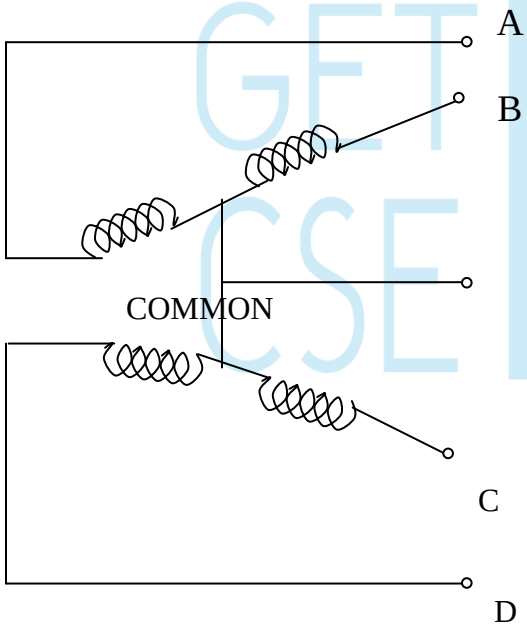
Stepper motor requires logic signals of relatively high power. Silicon Darlington pair TRSL100 with 2N3005 transistors are used to supply the power. The driving pulses are generated by the interface circuit and given to the four coils of the stepper motor. The inputs for the interface circuit are TTL pulses generated under software control using microprocessor kit. The TTL levels of pulse sequence at the output ports of 8255 are translated to high voltage output pulses of 12V using buffers (IC 7406). The Darlington pair transistor TRSL100 drives the stepper motor as they withstand higher current. A 620Ω resistor and a diode connected between power supply and Darlington pair collector are used for supporting fly back current. $PA_0 – PA_3$ of port A are used for driving the transistor TRSL100. The four collector points of each transistor are brought to a 5 pair connector for connecting to a stepper motor.

PROGRAM

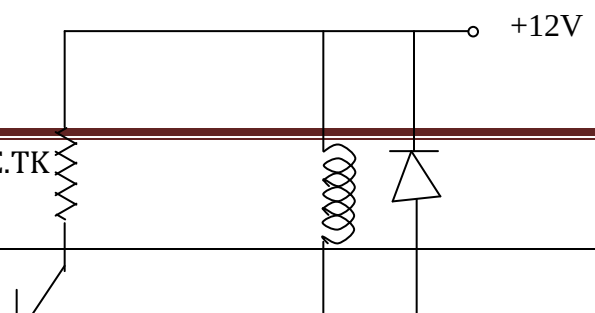
MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS

3E 80		MVI	A, 80	Control word to initialize the port A
D3 43		OUT	43 H	of 8255 as output port
3E 88		MVI	A, 88	Data sent to port A to energize the winding
D3 40	LOOP	OUT	40 H	of stepper motor.
CD 50 41		CALL	DELAY	Call the subroutine "DELAY"
0F		RRC		Rotate the Acc – contents rights carry by 1 bit.
C3 06 41		JMP	LOOP	Jump unconditionally to the instruction labeled "LOOP"

WINDING CONNECTION OF STEPPER MOTOR



DRIVER CIRCUIT FOR ENERGISING EACH WINDING



620Ω W_A 1N4001

PA 220Ω

SL 100

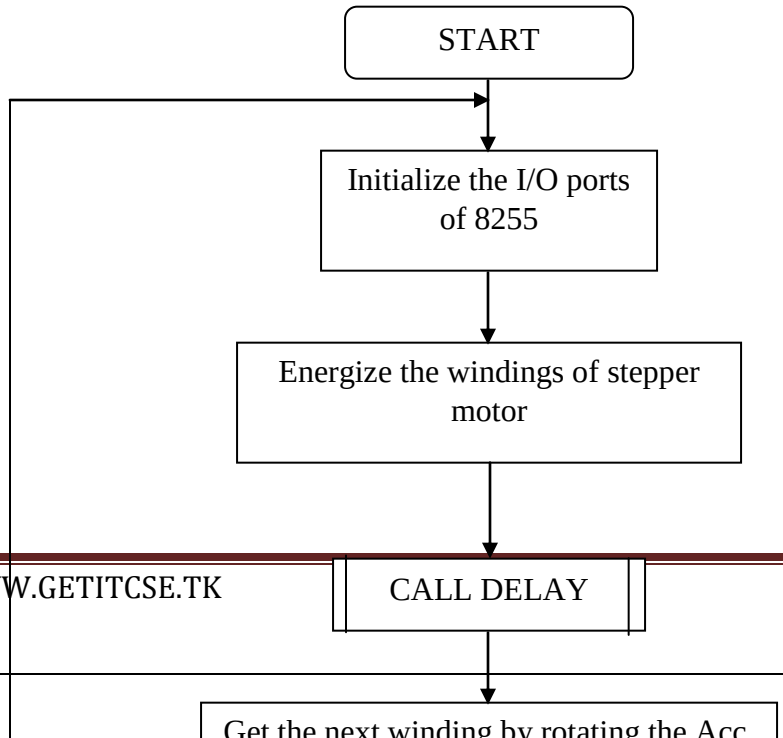
220Ω

2N 3055

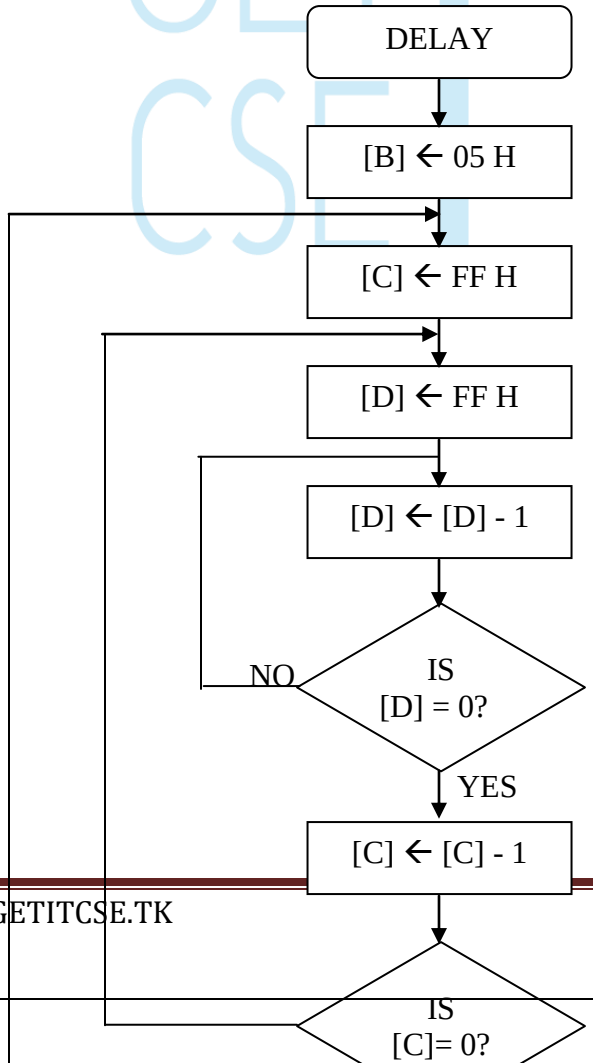
I / O decoding:

Address	selection
43 H	Port A is selected
40 H	Stepper motor is selected

FLOW CHART:



SUBROUTINE:



NO

YES

NO

YES

SUBROUTINE PROGRAM:

MEMORY ADDRESS	OPCODE	LABEL	MNEMONIC	OPERAND	COMMENTS
4150	06	DELAY	MVI	B, 05	Get the data 05 H in
4151	05				Reg. B
4152	0E	LOOP – 1	MVI	C, FF	Get the data FF H in
4153	FF				Reg. C
4154	16	LOOP – 2	MVI	D, FF	Get the data FF H in
4155	FF				Reg. D
4156	15	LOOP – 3	DCR	D	Decrement the contents of Reg. D
4157	C2		JNZ	LOOP – 3	If the contents of
4158	56				Reg. D is not zero go
4159	41				to instruction labeled LOOP - 3
415A	0D		DCR	C	Decrement the contents of Reg. C
415B	C2		JNZ	LOOP – 2	If the contents of
415C	54				Reg. C is not zero go
415D	41				to instruction labeled LOOP - 2
415E	05		DCR	B	Decrement the contents of Reg. B
415F	C2		JNZ	LOOP – 1	If the contents of
4160	52				Reg. B is not zero go
4161	41				to instruction labeled

4162	C9	RET	LOOP –1 Return to the main program.

CONCLUSION:

Thus an ALP to drive and control a stepper motor was written and executed.

24. 8051 - SUM OF ELEMENTS IN AN ARRAY

AIM:

To find the sum of elements in an array.

ALGORITHM:

1. Load the array in the consecutive memory location and initialize the memory pointer with the starting address.

2. Load the total number of elements in a separate register as a counter.
3. Clear the accumulator.
4. Load the other register with the value of the memory pointer.
5. Add the register with the accumulator.
6. Check for carry, if exist, increment the carry register by 1. otherwise, continue
7. Decrement the counter and if it reaches 0, stop. Otherwise increment the memory pointer by 1 and go to step 4.

RESULT:

The sum of elements in an array is calculated.

PROGRAM:

```
MOV DPTR, #4200
MOVX A, @DPTR
MOV R0,  A
MOV B, #00
MOV R1, B
INC DPTR
```

```
LOOP2:  CLR C
        MOVX A, @DPTR
        ADD A, B
        MOV B, A
        JNC LOOP
        INC R1
LOOP:   INC DPTR
        DJNZ R0, LOOP2
        MOV DPTR, #4500
        MOV A, R1
        MOVX @DPTR, A
        INC DPTR
        MOV A, B
        MOVX @DPTR, A
HLT:    SJMP HLT
```

INPUT		**OUTPUT:**	
4200	04	4500	0F
4201	05	4501	00
4201	06		
4202	03		
4203	02		

25(A).8051 - HEXADECIMAL TO DECIMAL CONVERSION

AIM:

To perform hexadecimal to decimal conversion.

ALGORITHM:

1. Load the number to be converted into the accumulator.
2. If the number is less than 100 (64H), go to next step; otherwise, subtract 100 (64H) repeatedly until the remainder is less than 100 (64H). Have the count(100's value) in separate register which is the carry.
3. If the number is less than 10 (0AH), go to next step; otherwise, subtract 10 (0AH) repeatedly until the remainder is less than 10 (0AH). Have the count(ten's value) in separate register.
4. The accumulator now has the units.
5. Multiply the ten's value by 10 and add it with the units.
6. Store the result and carry in the specified memory location.

RESULT

The given hexadecimal number is converted into decimal number.

PROGRAM:

```
        MOV DPTR, #4500
        MOVX A, @DPTR
        MOV B, #64
        DIV A, B
        MOV DPTR, #4501
        MOVX @DPTR, A
        MOV A, B
        MOV B, #0A
        DIV A, B
        INC DPTR
        MOVX @DPTR, A
        INC DPTR
        MOV A, B
        MOVX @DPTR, A
HLT:    SJMP HLT
```

INPUT

4500 D7

OUTPUT:

4501 15

4502 02

25(B).8051 - DECIMAL TO HEXADECIMAL CONVERSION

AIM:

To perform decimal to hexadecimal conversion

ALGORITHM:

1. Load the number to be converted in the accumulator.
2. Separate the higher order digit from lower order.
3. Multiply the higher order digit by 10 and add it with the lower order digit.
4. Store the result in the specified memory location.

RESULT:

The given decimal number is converted to hexadecimal number.

PROGRAM:

```
        MOV DPTR, #4500
        MOVX A, @DPTR
        MOV B, #0A
        MUL A, B
        MOV B, A
        INC DPTR
        MOVX A, @DPTR
        ADD A, B
        INC DPTR
        MOVX @DPTR, A
HLT:    SJMP HLT
```

INPUT		OUTPUT	
4500	23	4501	17

26. STEPPER MOTOR INTERFACING WITH 8051

AIM:

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

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.

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.

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		DELAY1:	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

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.