SRINIVASAN ENGINEERING COLLEGE
PERAMBALUR-621212

Department of Aeronautical Engineering

AE2406 – AVIONICS LAB

VII SEM
AERONAUTICAL ENGINEERING

PREPARED BY-

Mr. T.AYYASAMY

AE2406 AVIONICS LAB

VII Semester Aero

Syllabus

CYCLE I - 8085 MICROPROCESSORS

1. Addition and Subtraction of 8-bit and 16-bit numbers.
2. Sorting of Data in Ascending & Descending order.
3. Sum of a given series with and without carry.
4. Greatest in a given series & Multi-byte addition in BCD mode.
5. Interface programming with 4 digit 7 segment Display & Switches & LED's.
6. 16 Channel Analog to Digital Converter & Generation of Ramp, Square, Triangular wave by Digital to Analog Converter.

CYCLE II - DIGITAL ELECTRONICS

7. Addition/Subtraction of binary numbers.
8. Multiplexer/Demultiplexer Circuits.
9. Encoder/Decoder Circuits.
10. Timer Circuits, Shift Registers, Binary Comparator Circuits.

CYCLE III - AVIONICS DATA BUSES

11. Study of Different Avionics Data Buses.
12. MIL-Std – 1553 Data Buses Configuration with Message transfer.
13. MIL-Std – 1553 Remote Terminal Configuration**.**

1. INTRODUCTION TO 8085

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

ALU (Arithmetic Logic Unit):

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

- Sign flag

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

- Zero flag

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

- Auxillary carry flag

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

- Parity flag

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

- Carry flag

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

Timing and control unit

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

Instruction register and decoder

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

Register array

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

Accumulator

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

Program counter

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

Stack pointer

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

Communication lines

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

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

2(A). 8 BIT DATA ADDITION

AIM:

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

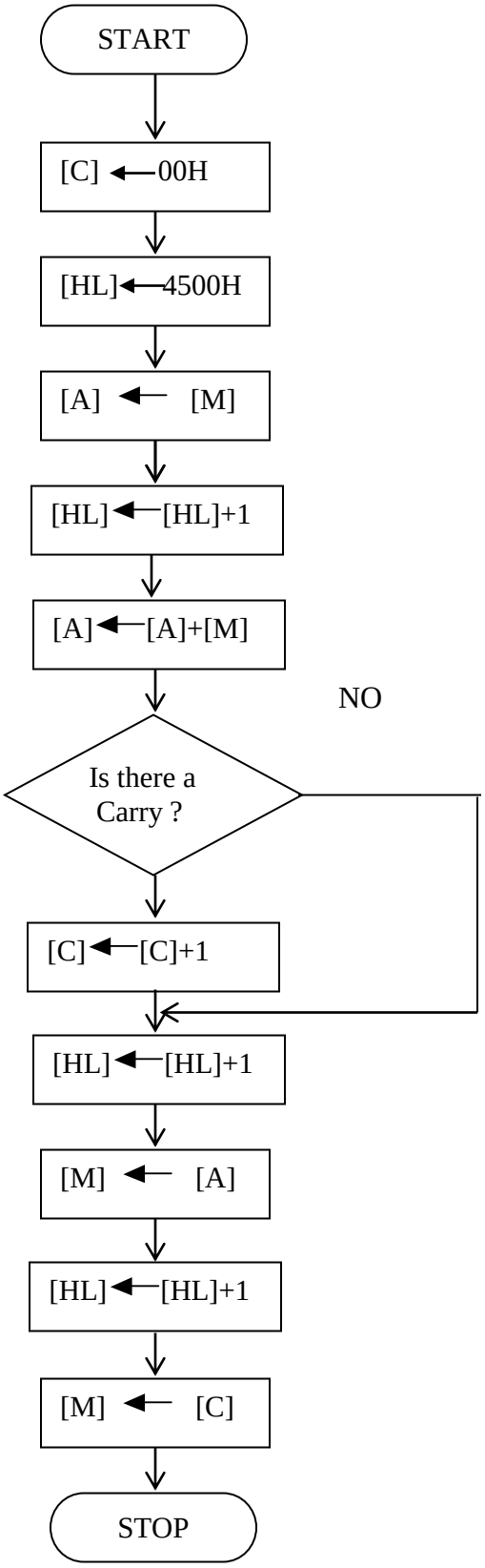
ALGORITHM:

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

RESULT:

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

FLOW CHART:


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

PROGRAM:

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

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

2(B). 8 BIT DATA SUBTRACTION

AIM:

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

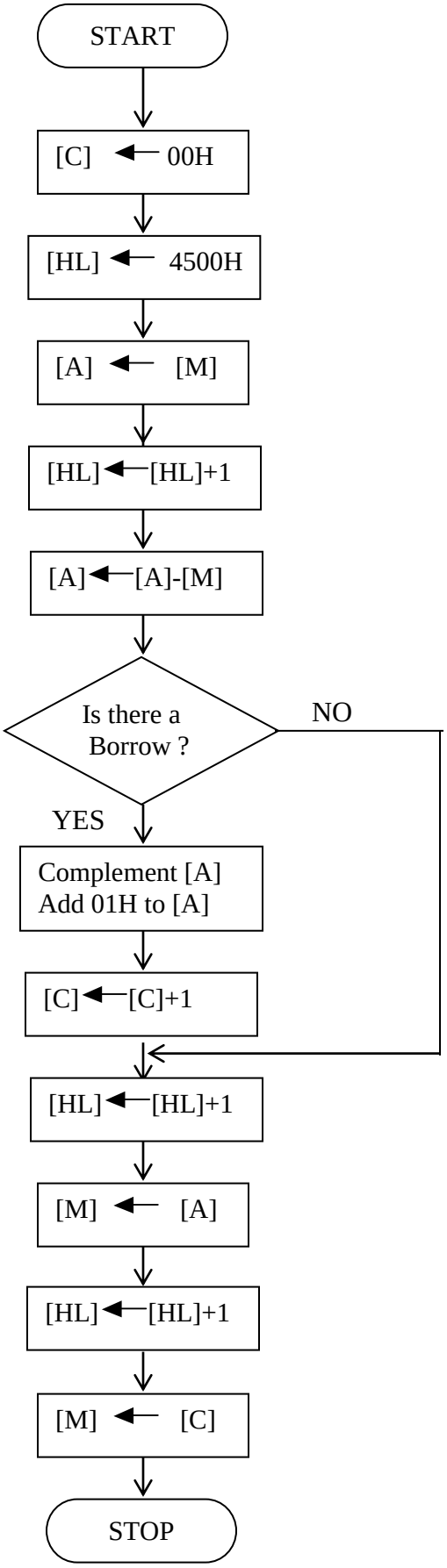
ALGORITHM:

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

RESULT:

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

FLOW CHART:


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

PROGRAM:

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

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

3(A). 8 BIT DATA MULTIPLICATION

AIM:

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

ALGORITHM:

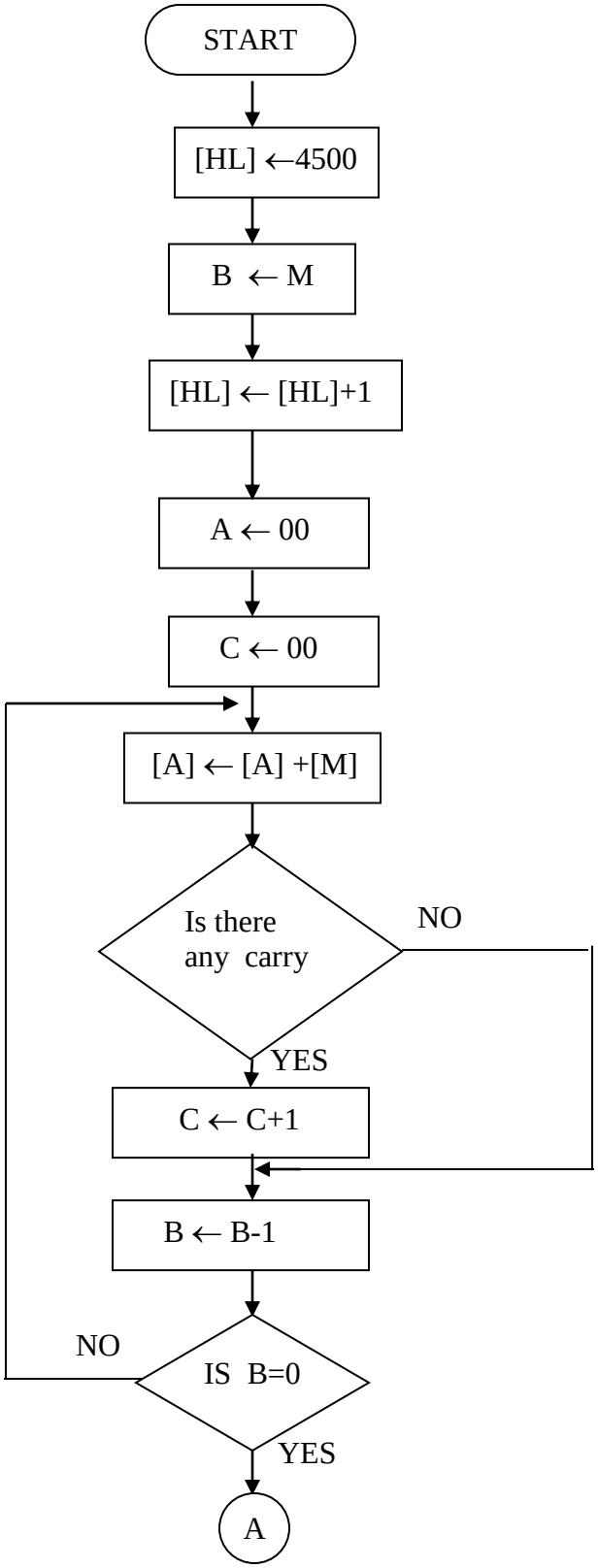
LOGIC: Multiplication can be done by repeated addition.

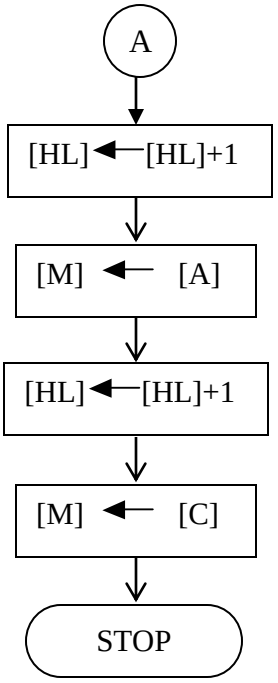
1. Initialize memory pointer to data location.
2. Move multiplicand to a register.
3. Move the multiplier to another register.
4. Clear the accumulator.
5. Add multiplicand to accumulator
6. Decrement multiplier
7. Repeat step 5 till multiplier comes to zero.
8. The result, which is in the accumulator,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

RESULT:

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

FLOW CHART:


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


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

PROGRAM:

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

OBSERVATION:

INPUT		OUTPUT	
4500		4502	
4501		4503	

3(B). 8 BIT DIVISION

AIM:

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

ALGORITHM:

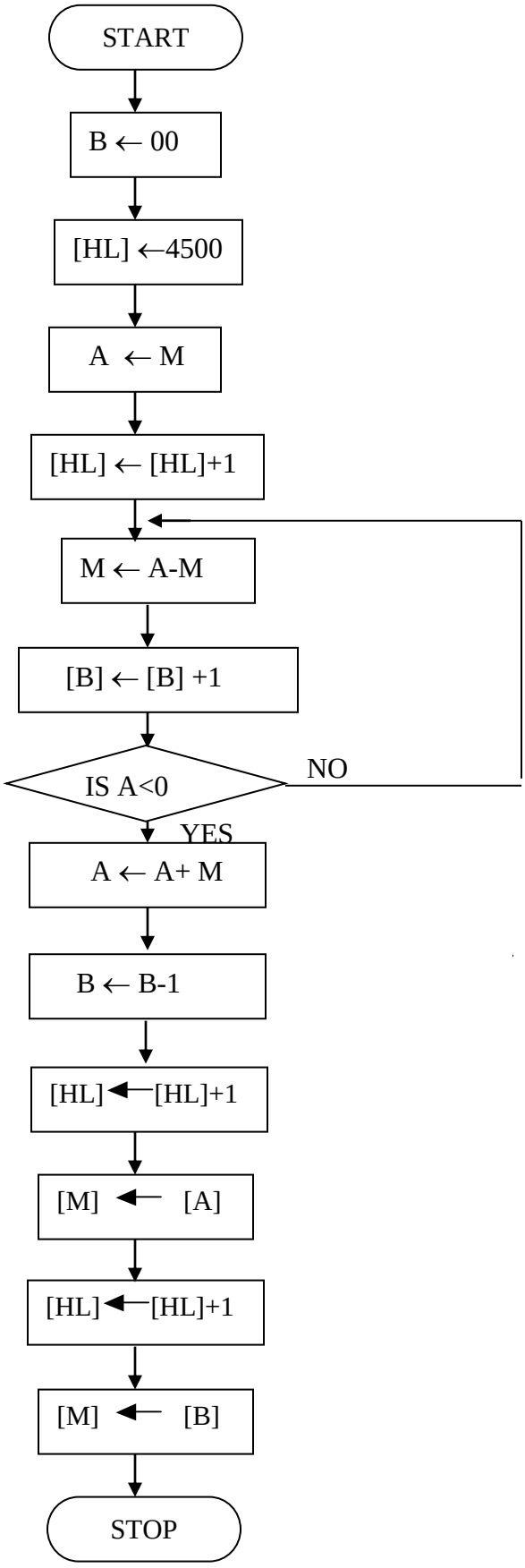
LOGIC: Division is done using the method Repeated subtraction.

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

RESULT:

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

FLOWCHART:


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

PROGRAM:

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

OBSERVATION:

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

4(A). 16 BIT DATA ADDITION

AIM:

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

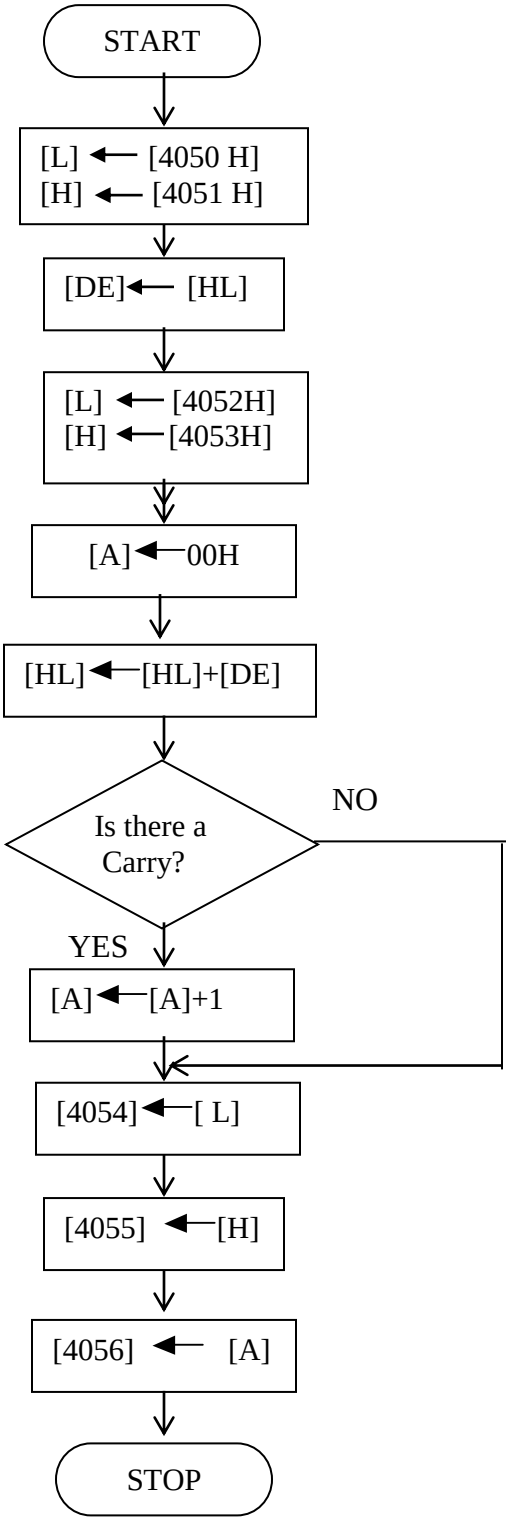
ALGORITHM:

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

RESULT:

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

FLOW CHART:


START
[L] ← [4050 H]
[H] ← [4051 H]
[DE] ← [HL]
[L] ← [4052H]
[H] ← [4053H]
[A] ← 00H
[HL] ← [HL]+[DE]
Is there a Carry?
NO
YES
[A] ← [A]+1
[4054] ← [L]
[4055] ← [H]
[4056] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENT
4000		START	LHLD	4050H	Load the augend in DE pair through HL pair.
4001					
4002					
4003			XCHG		
4004			LHLD	4052H	Load the addend in HL pair.
4005					
4006					
4007			MVI	A, 00H	Initialize reg. A for carry
4008					
4009			DAD	D	Add the contents of HL Pair with that of DE pair.
400A			JNC	LOOP	If there is no carry, go to the instruction labeled LOOP.
400B					
400C					
400D			INR	A	Otherwise increment reg. A
400E		LOOP	SHLD	4054H	Store the content of HL Pair in 4054H(LSB of sum)
400F					
4010					
4011			STA	4056H	Store the carry in 4056H through Acc. (MSB of sum).
4012					
4013					
4014			HLT		Stop the program.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4050H		4054H	
4051H		4055H	
4052H		4056H	
4053H			

4(B). 16 BIT DATA SUBTRACTION

AIM:

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

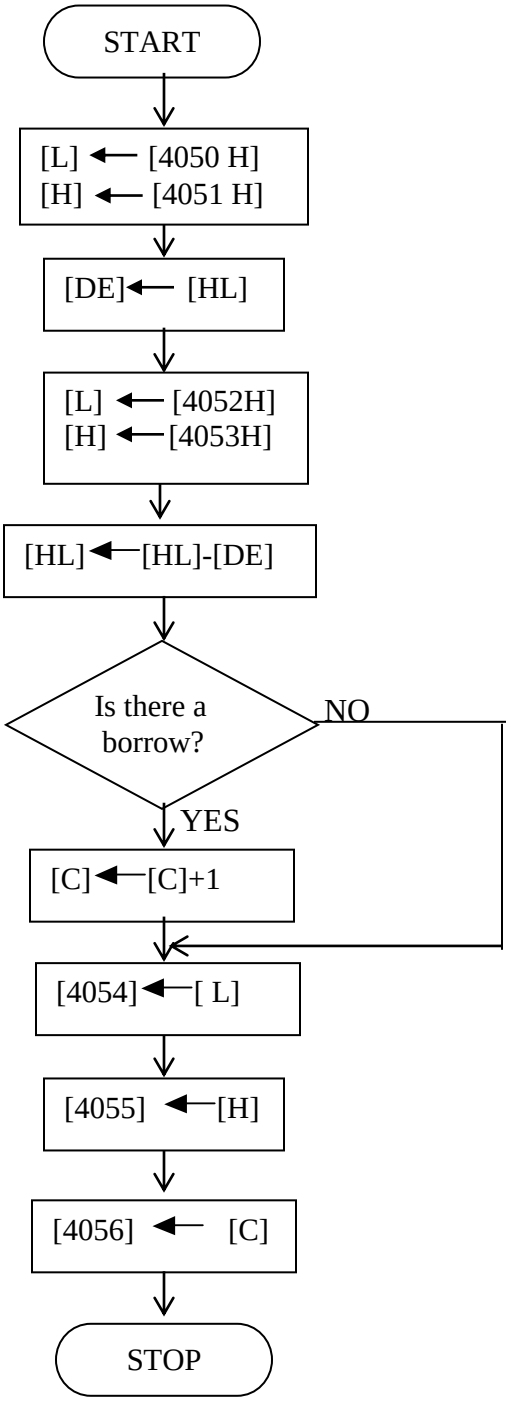
ALGORITHM:

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] ← [4050 H]
[H] ← [4051 H]
[DE] ← [HL]
[L] ← [4052H]
[H] ← [4053H]
[HL] ← [HL]-[DE]
Is there a borrow?
NO
YES
[C] ← [C]+1
[4054] ← [L]
[4055] ← [H]
[4056] ← [C]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4000		START	MVI	C, 00	Initialize C reg.
4001					
4002			LHLD	4050H	Load the subtrahend in DE reg. Pair through HL reg. pair.
4003					
4004					
4005			XCHG		
4006			LHLD	4052H	Load the minuend in HL reg. Pair.
4007					
4008					
4009			MOV	A, L	Move the content of reg. L to Acc.
400A			SUB	E	Subtract the content of reg. E from that of acc.
400B			MOV	L, A	Move the content of Acc. to reg. L
400C			MOV	A, H	Move the content of reg. H to Acc.
400D			SBB	D	Subtract content of reg. D with that of Acc.
400E			MOV	H, A	Transfer content of acc. to reg. H
400F			SHLD	4054H	Store the content of HL pair in memory location 8504H.
4010					
4011					
4012			JNC	NEXT	If there is borrow, go to the instruction labeled NEXT.
4013					
4014					
4015			INR	C	Increment reg. C
4016		NEXT	MOV	A, C	Transfer the content of reg. C to Acc.
4017			STA	4056H	Store the content of acc. to the memory location 4506H
4018					
4019					
401A			HLT		Stop the program execution.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4050H		4054H	
4051H		4055H	
4052H		4056H	
4053H			

5(A). 16 BIT MULTIPLICATION

AIM:

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

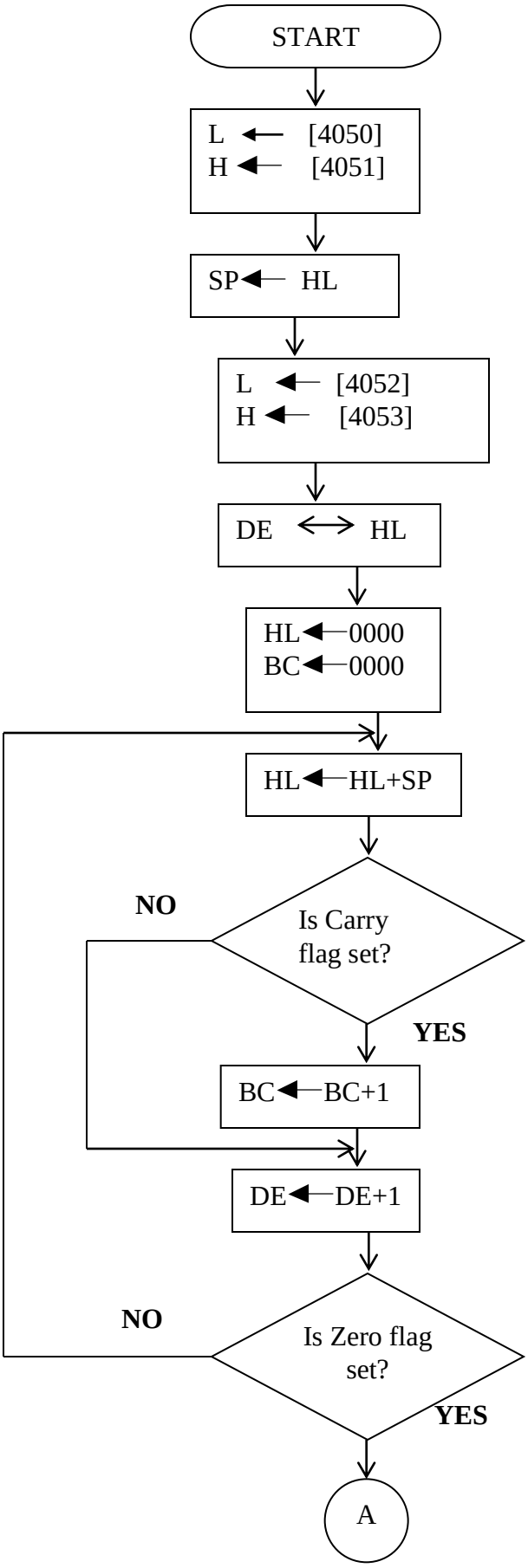
ALGORITHM:

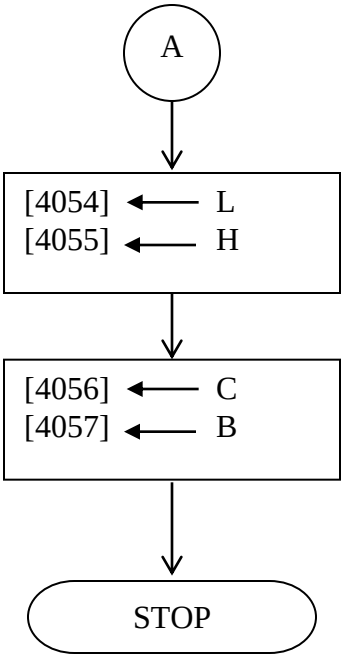
1. Get the multiplier and multiplicand.
2. Initialize a register to store partial product.
3. Add multiplicand, multiplier times.
4. Store the result in consecutive memory locations.

RESULT:

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

FLOWCHART:


START
L ← [4050]
H ← [4051]
SP ← HL
L ← [4052]
H ← [4053]
DE ↔ HL
HL ← 0000
BC ← 0000
HL ← HL+SP
NO
Is Carry flag set?
YES
BC ← BC+1
DE ← DE+1
NO
Is Zero flag set?
YES
A


A
[4054] ← L
[4055] ← H
[4056] ← C
[4057] ← B
STOP

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
8000		START	LHLD	4050	Load the first No. in stack pointer through HL reg. pair
4001					
4002					
4003			SPHL		
4004			LHLD	4052	Load the second No. in HL reg. pair & Exchange with DE reg. pair.
4005					
4006					
4007			XCHG		
4008			LXI	H, 0000H	Clear HL & DE reg. pairs.
4009					
400A					
400B			LXI	B, 0000H	
400C					
400D					
400E		LOOP	DAD	SP	Add SP with HL pair.
400F			JNC	NEXT	If there is no carry, go to the instruction labeled NEXT
4010					
4011					
4012			INX	B	Increment BC reg. pair
4013		NEXT	DCX	D	Decrement DE reg. pair.
4014			MOV	A,E	Move the content of reg. E to Acc.
4015			ORA	D	OR Acc. with D reg.
4016			JNZ	LOOP	If there is no zero, go to instruction labeled LOOP
4017					
4018					
4019			SHLD	4054	Store the content of HL pair in memory locations 4054 & 4055.
401A					
401B					
401C			MOV	A, C	Move the content of reg. C to Acc.
401D			STA	4056	Store the content of Acc. in memory location 4056.
401E					
401F					
4020			MOV	A, B	Move the content of reg. B to Acc.
4021			STA	4057	Store the content of Acc. in memory location 4056.
4022					
4023					
4024			HLT		Stop program execution

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4050		4054	
4051		4055	
4052		4056	
4053		4057	

5(B). 16- BIT DIVISION

AIM:

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

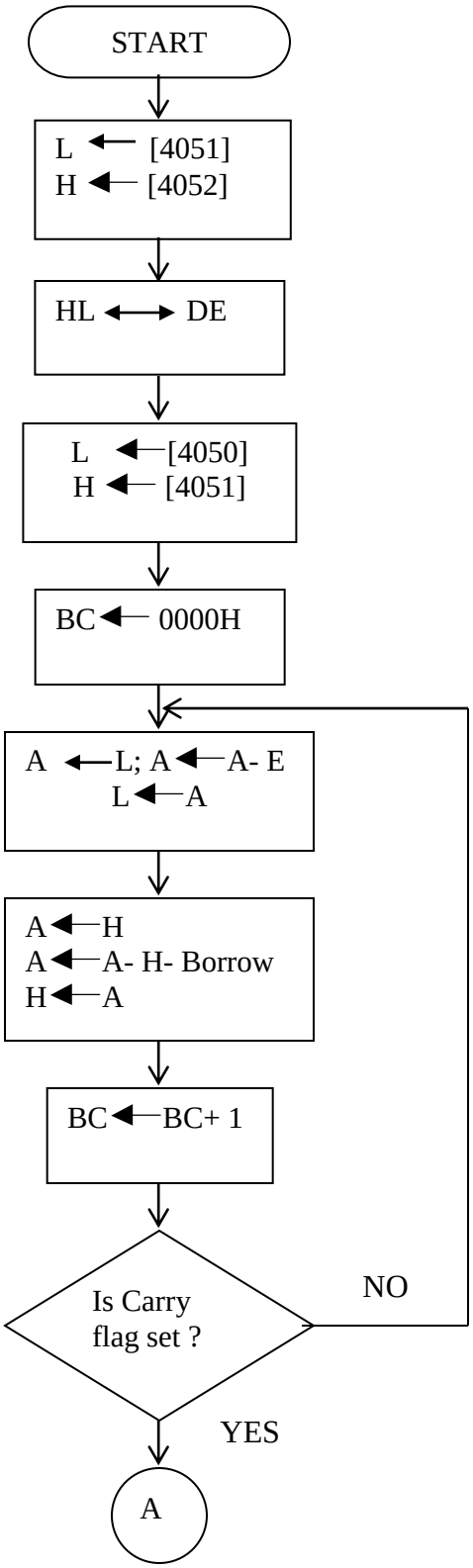
ALGORITHM:

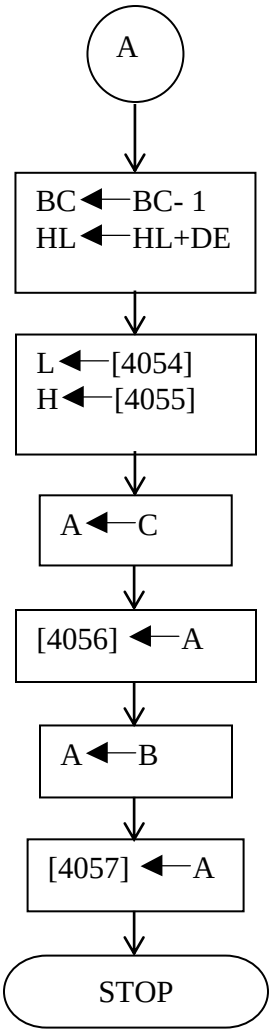
1. Get the dividend and divisor.
2. Initialize the register for quotient.
3. Repeatedly subtract divisor from dividend till dividend becomes less than divisor.
4. Count the number of subtraction which equals the quotient.
5. Store the result in memory.

RESULT:

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

FLOWCHART:


START
L ← [4051]
H ← [4052]
HL ↔ DE
L ← [4050]
H ← [4051]
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 ← [4054]
H ← [4055]
A ← C
[4056] ← A
A ← B
[4057] ← A
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4000		START	LHLD	4052	Load the first No. in stack pointer through HL reg. pair
4001					
4002					
4003			XCHG		
4004			LHLD	4050	Load the second No. in HL reg. pair & Exchange with DE reg. pair.
4005					
4006					
4007			LXI	B, 0000H	Clear BC reg. pair.
4008					
4009					
400A		LOOP	MOV	A, L	Move the content of reg. L to Acc.
400B			SUB	E	Subtract reg. E from that of Acc.
400C			MOV	L, A	Move the content of Acc to L.
400D			MOV	A, H	Move the content of reg. H Acc.
400E			SBB	D	Subtract reg. D from that of Acc.
400F			MOV	H, A	Move the content of Acc to H.
4010			INX	B	Increment reg. Pair BC
4011			JNC	LOOP	If there is no carry, go to the location labeled LOOP.
4012					
4013					
4014			DCX	B	Decrement BC reg. pair.
4015			DAD	D	Add content of HL and DE reg. pairs.
4016			SHLD	4054	Store the content of HL pair in 4054 & 4055.
4017					
4018					
4019			MOV	A, C	Move the content of reg. C to Acc.
401A			STA	4056	Store the content of Acc. in memory 4056
401B					
401C					
401D			MOV	A, B	Move the content of reg. B to Acc.
401E			STA	4057	Store the content of Acc. in memory 4057.
401F					
4020					
4021			HLT		Stop the program execution.

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4050		4054	
4051		4055	
4052		4056	
4053		4057	

6(A). LARGEST ELEMENT IN AN ARRAY

AIM:

To find the largest element in an array.

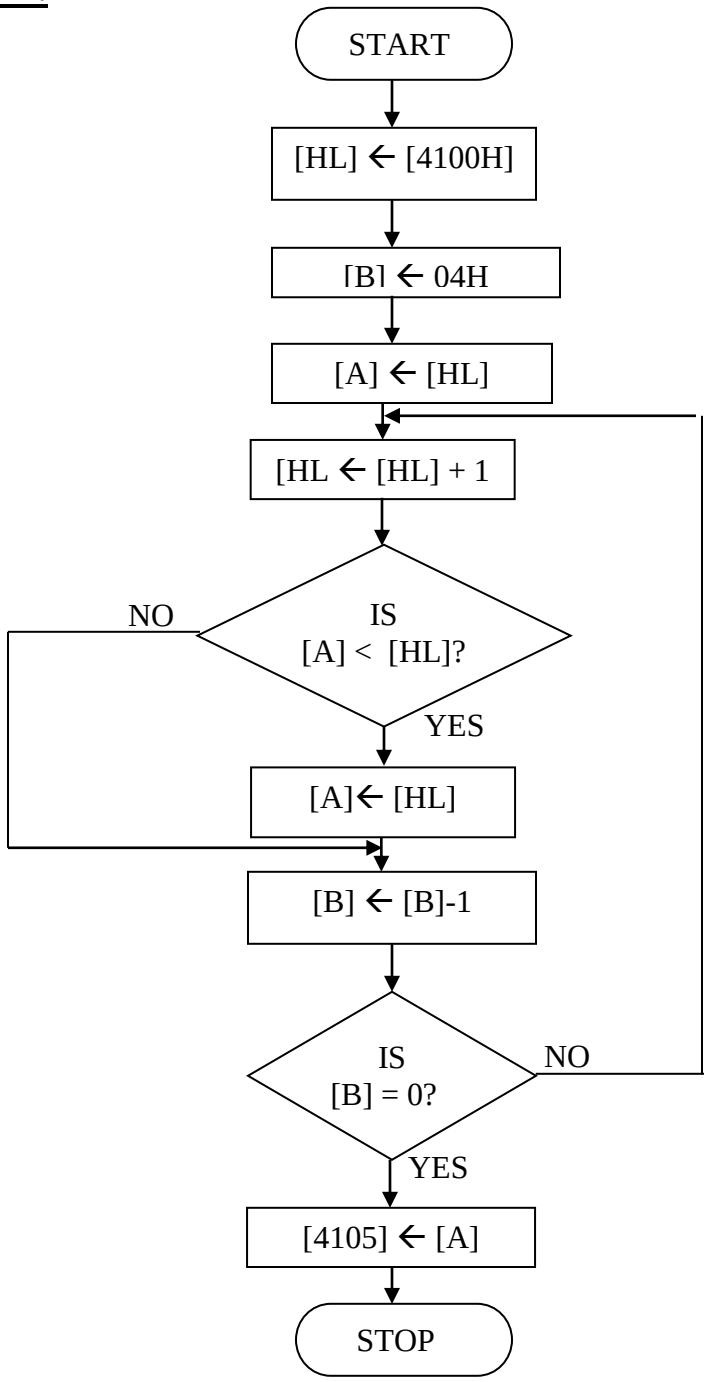
ALGORITHM:

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] ← [4100H]
[B] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
NO
YES
[A]← [HL]
[B] ← [B]-1
IS
[B] = 0?
NO
YES
[4105] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4001			LXI	H,4100	Initialize HL reg. to 4100H
4002					
4003					
4004			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
4005					
4006			MOV	A,M	Transfer first data to acc.
4007		LOOP1	INX	H	Increment HL reg. to point next memory location
4008			CMP	M	Compare M & A
4009			JNC	LOOP	If A is greater than M then go to loop
400A					
400B					
400C			MOV	A,M	Transfer data from M to A reg
400D		LOOP	DCR	B	Decrement B reg
400E			JNZ	LOOP1	If B is not Zero go to loop1
400F					
4010					
4011			STA	4105	Store the result in a memory location.
4012					
4013					
4014			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4100		4105	
4101			
4102			
4103			
4104			

6(B). SMALLEST ELEMENT IN AN ARRAY

AIM:

To find the smallest element in an array.

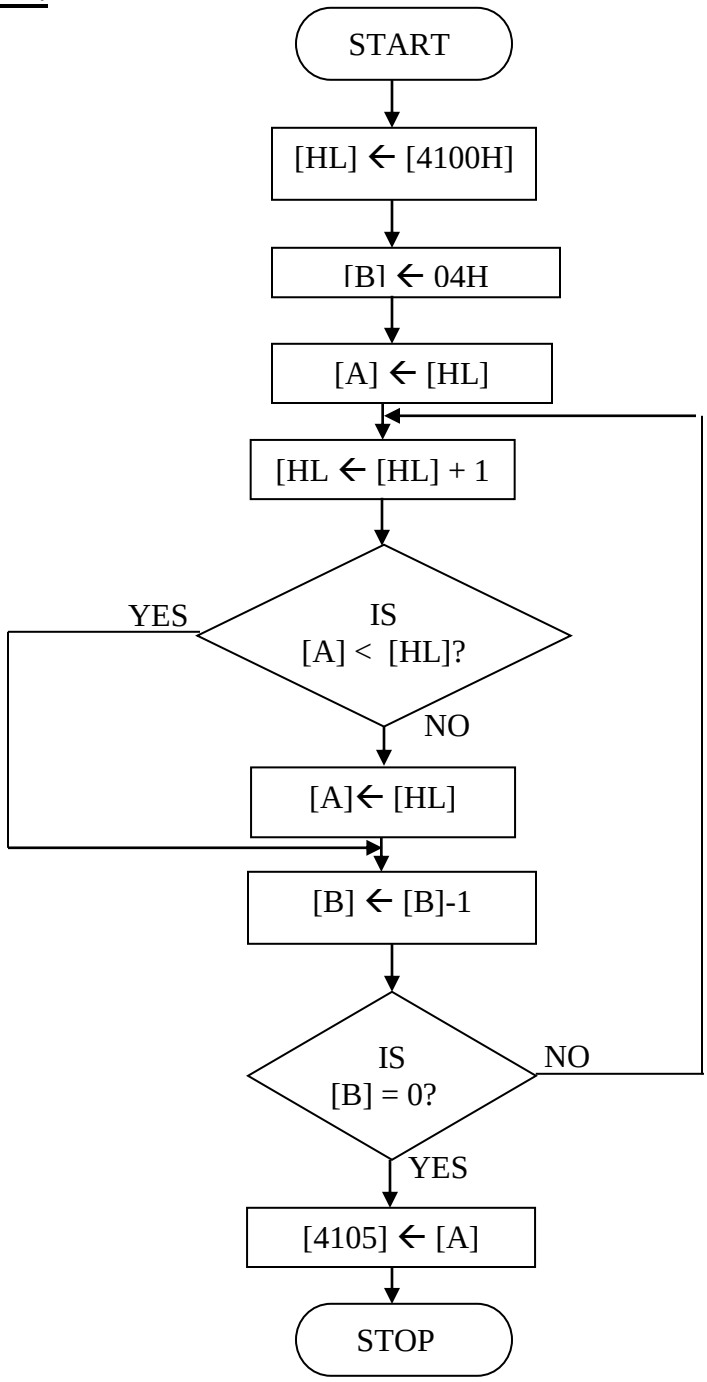
ALGORITHM:

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] ← [4100H]
[B] ← 04H
[A] ← [HL]
[HL ← [HL] + 1
IS
[A] < [HL]?
YES
NO
[A]← [HL]
[B] ← [B]-1
IS
[B] = 0?
NO
YES
[4105] ← [A]
STOP

PROGRAM:

ADDRESS	OPCODE	LABEL	MNEMONICS	OPERAND	COMMENTS
4001			LXI	H,4100	Initialize HL reg. to 4100H
4002					
4003					
4004			MVI	B,04	Initialize B reg with no. of comparisons(n-1)
4005					
4006			MOV	A,M	Transfer first data to acc.
4007		LOOP1	INX	H	Increment HL reg. to point next memory location
4004			CMP	M	Compare M & A
4009			JC	LOOP	If A is lesser than M then go to loop
400A					
400B					
400C			MOV	A,M	Transfer data from M to A reg
400D		LOOP	DCR	B	Decrement B reg
400E			JNZ	LOOP1	If B is not Zero go to loop1
400F					
4010					
4011			STA	4105	Store the result in a memory location.
4012					
4013					
4014			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
ADDRESS	DATA	ADDRESS	DATA
4100		4105	
4101			
4102			
4103			
4104			

7(A).ASCENDING ORDER

AIM:

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

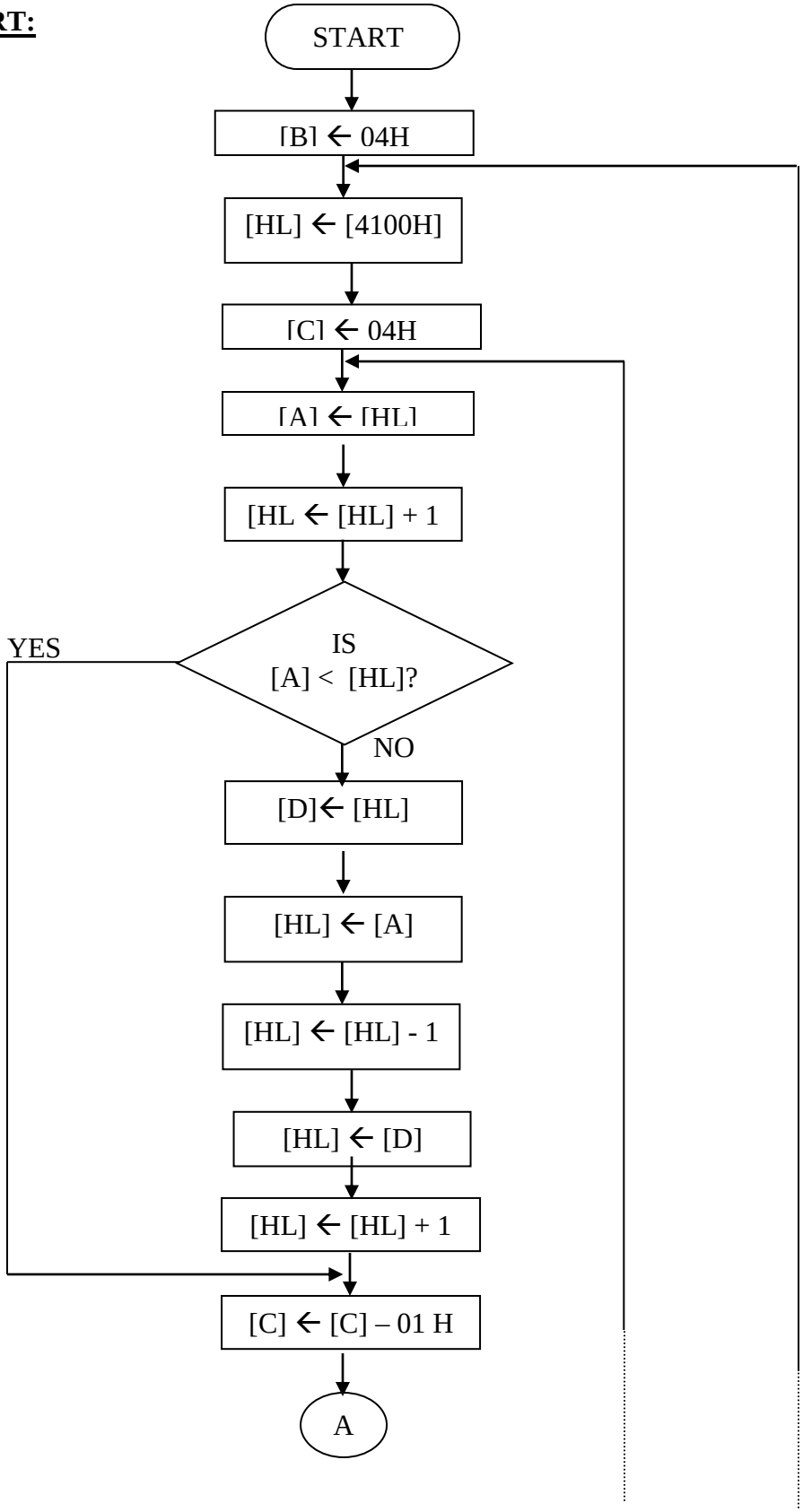
ALGORITHM:

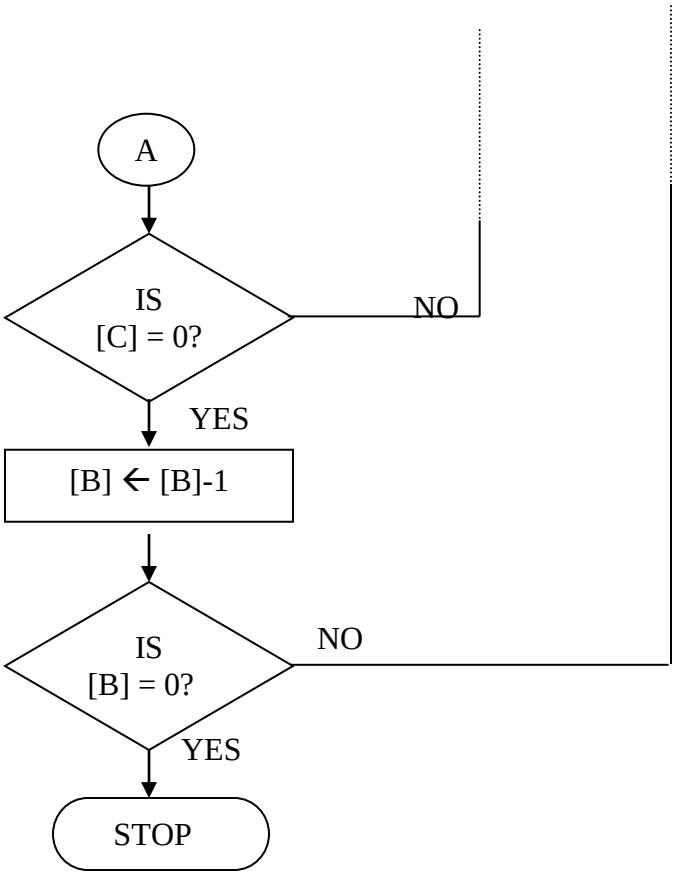
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is larger than second then I interchange the number.
3. If the first number is smaller, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

RESULT:

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

FLOWCHART:


START
[B] ← 04H
[HL] ← [4100H]
[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
4000			MVI	B,04	Initialize B reg with number of comparisons (n-1)
4001					
4002		LOOP 3	LXI	H,4100	Initialize HL reg. to 4100H
4003					
4004					
4005			MVI	C,04	Initialize C reg with no. of comparisons(n-1)
4006					
4007		LOOP2	MOV	A,M	Transfer first data to acc.
4004			INX	H	Increment HL reg. to point next memory location
4009			CMP	M	Compare M & A
400A			JC	LOOP1	If A is less than M then go to loop1
400B					
400C					
400D			MOV	D,M	Transfer data from M to D reg
400E			MOV	M,A	Transfer data from acc to M
400F			DCX	H	Decrement HL pair
4010			MOV	M,D	Transfer data from D to M
4011			INX	H	Increment HL pair
4012		LOOP1	DCR	C	Decrement C reg
4013			JNZ	LOOP2	If C is not zero go to loop2
4014					
4015					
4016			DCR	B	Decrement B reg
4017			JNZ	LOOP3	If B is not Zero go to loop3
4014					
4019					
401A			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4100		4100	
4101		4101	
4102		4102	
4103		4103	
4104		4104	

7(B). DESCENDING ORDER

AIM:

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

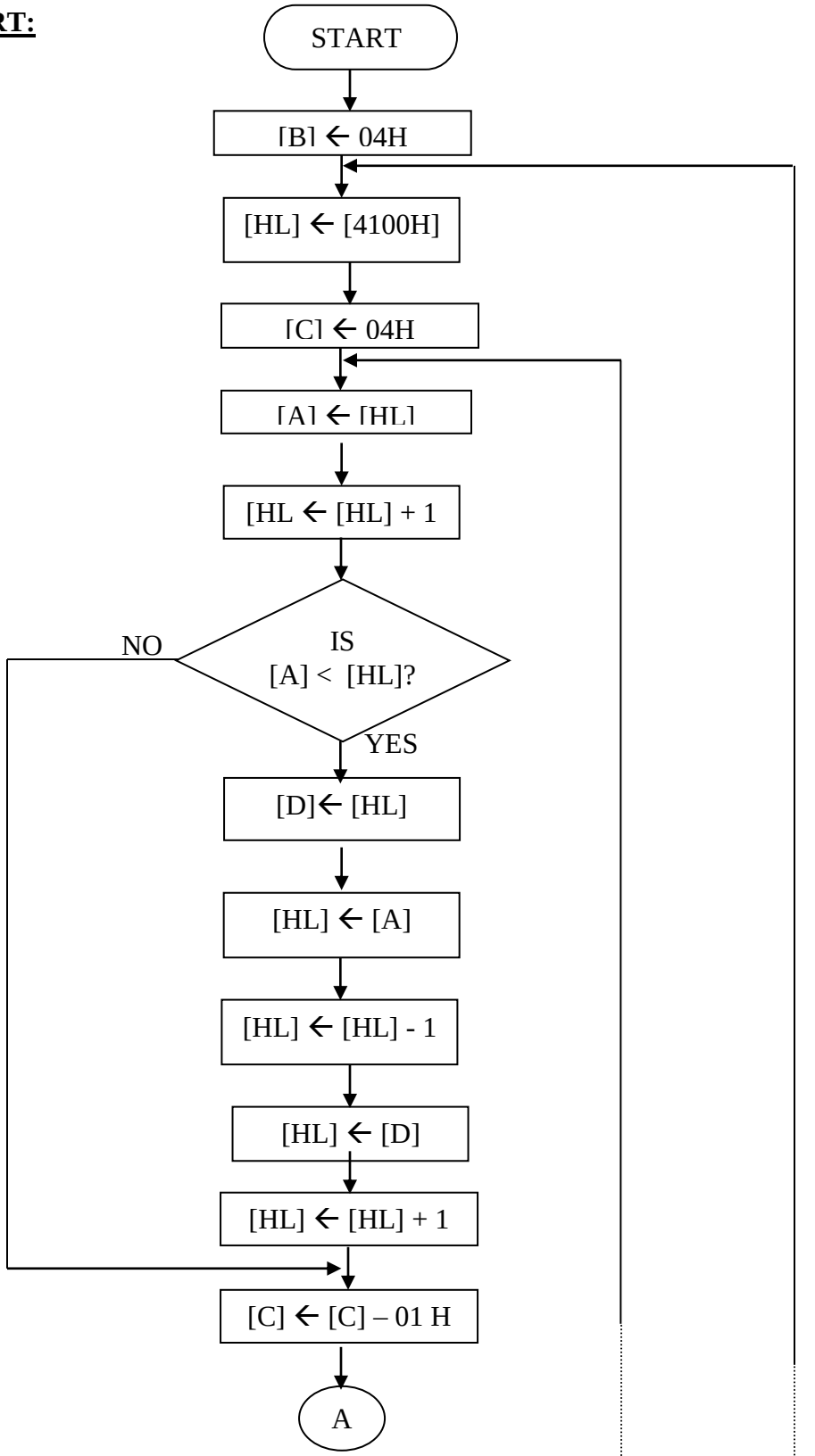
ALGORITHM:

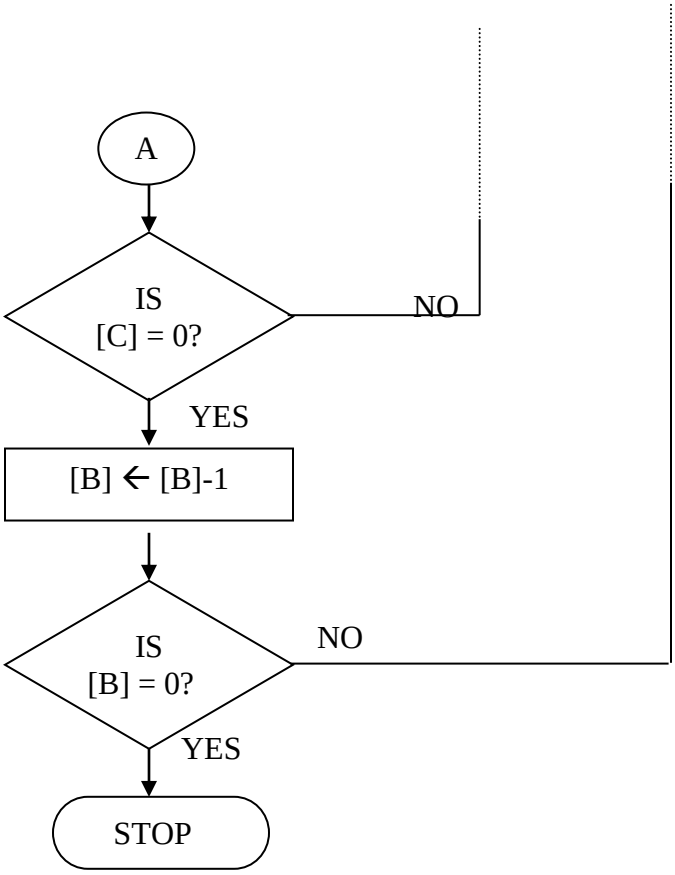
1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is smaller than second then I interchange the number.
3. If the first number is larger, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order

RESULT:

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

FLOWCHART:


START
[B] ← 04H
[HL] ← [4100H]
[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
4000			MVI	B,04	Initialize B reg with number of comparisons (n-1)
4001					
4002		LOOP 3	LXI	H,4100	Initialize HL reg. to 4100H
4003					
4004					
4005			MVI	C,04	Initialize C reg with no. of comparisons(n-1)
4006					
4007		LOOP2	MOV	A,M	Transfer first data to acc.
4004			INX	H	Increment HL reg. to point next memory location
4009			CMP	M	Compare M & A
400A			JNC	LOOP1	If A is greater than M then go to loop1
400B					
400C					
400D			MOV	D,M	Transfer data from M to D reg
400E			MOV	M,A	Transfer data from acc to M
400F			DCX	H	Decrement HL pair
4010			MOV	M,D	Transfer data from D to M
4011			INX	H	Increment HL pair
4012		LOOP1	DCR	C	Decrement C reg
4013			JNZ	LOOP2	If C is not zero go to loop2
4014					
4015					
4016			DCR	B	Decrement B reg
4017			JNZ	LOOP3	If B is not Zero go to loop3
4014					
4019					
401A			HLT		Stop the program

OBSERVATION:

INPUT		OUTPUT	
MEMORY LOCATION	DATA	MEMORY LOCATION	DATA
4100		4100	
4101		4101	
4102		4102	
4103		4103	
4104		4104	

8(A) BCD ADDITION

AIM:

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

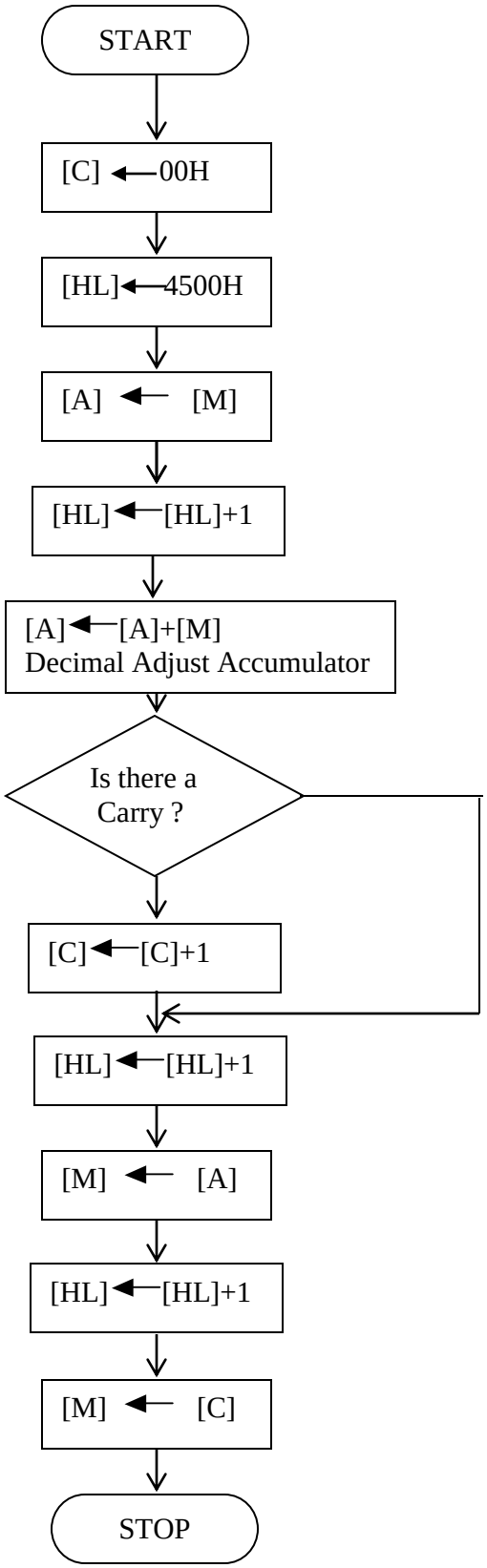
ALGORITHM:

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	

8(B). BCD SUBTRACTION

AIM:

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

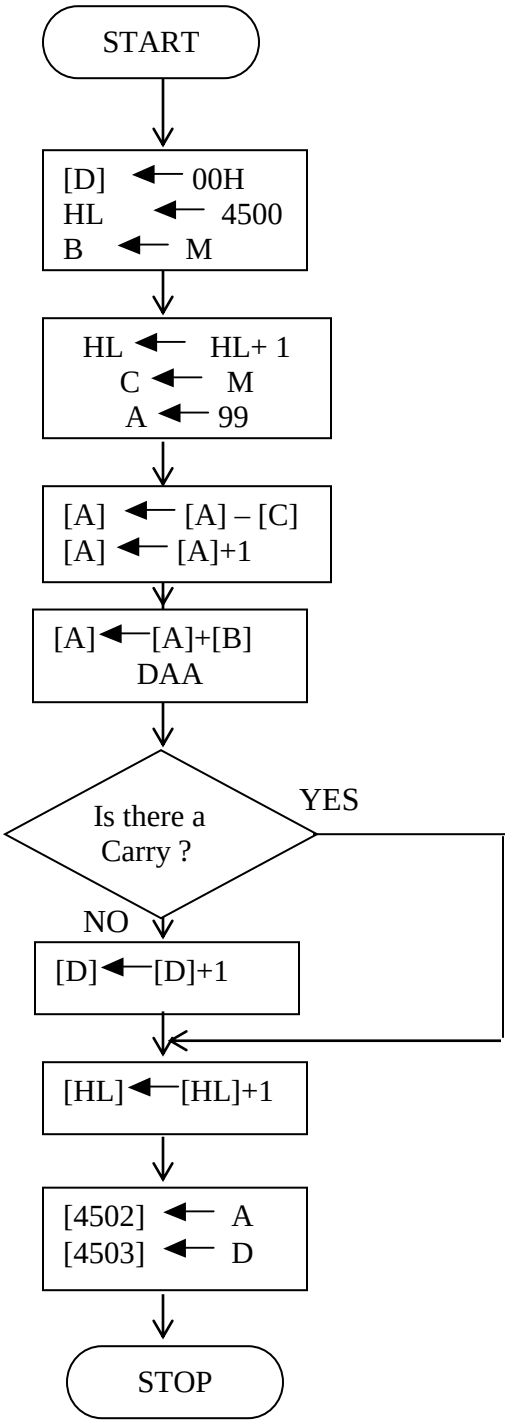
ALGORITHM:

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	

9(A). To Find the Sum of Given Series of Number Without Carry

Aim:

To write an assembly language program to find the sum of series of data without carry.

Apparatus Required:

Microprocessor 8085 kit, power supply.

Mnemonics:

```
XRA A
LXI H, A001
MOV C, M
INX H
MOV A, M
DCR C
INX H
ADD M
DCR C
JNZ 8008
STA C001
RST 1
```

Result:

Thus an assembly language for sum of given numbers without carry is written and executed.

9(B). To Find the Sum of Given Series of Number With Carry

Date:

Aim:

To write an assembly language program to find the sum of series of data with carry.

Apparatus Required:

Microprocessor 8085 kit, power supply.

Mnemonics:

```
MVI C, 00
XRA A
LXI H, A001
MOV B, M
INX H
MOV A, M
DCR B
INX H
ADD M
JNC 8010
INR C
DCR B
JNZ 800A
STA C001
MOV A, C
STA C002
RST 1
```

Result:

Thus an assembly language for sum of given numbers with carry is written and executed.

INTERFACING ADC WITH 8085 PROCESSOR

AIM:

To write a program to initiate ADC and to store the digital data in memory

PROGRAM:

```
        MVI     A,10
        OUT     C8
        MVI     A,18
        OUT     C8
        MVI     A,10
        OUT     D0
        XRA     A
        XRA     A
        XRA     A
        MVI     A,00
        OUT     D0
LOOP:   IN      D8
        ANI     01
        CPI     01
        JNZ     LOOP
        IN      C0
        STA     4150
        HLT
```

OBSERVATION:

Compare the data displayed at the LEDs with that stored at location 4150

RESULT:

Thus the ADC was initiated and the digital data was stored at desired location

INTERFACING DAC WITH 8085

AIM:

To interface DAC with 8085 to demonstrate the generation of square, saw tooth and triangular wave.

APPARATUS REQUIRED:

- 8085 Trainer Kit
- DAC Interface Board

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V.The output voltage varies in steps of 10/256 = 0.04 (appx.). The digital data input and the corresponding output voltages are presented in the Table1.

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

Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc,. The port address of DAC is 08 H.

ALGORITHM:

(a) Square Wave Generation

1. Load the initial value (00) to Accumulator and move it to DAC
2. Call the delay program
3. Load the final value(FF) to accumulator and move it to DAC
4. Call the delay program.
5. Repeat Steps 2 to 5

(b) Saw tooth Wave Generation

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. Repeat Steps 3 and 4.

(c) Triangular Wave Generation

2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:

(a) Square Wave Generation

```
START:  MVI   A,00
        OUT   Port address of DAC
        CALL  DELAY
        MVI   A,FF
        OUT   Port address of DAC
        CALL  DELAY
        JMP   START

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

(b) Saw tooth Wave Generation

```
START:  MVI   A,00
L1:     OUT   Port address of DAC
        INR   A
        JNZ   L1
        JMP   START
```

(c) Triangular Wave Generation

```
START:  MVI   L,00
L1:     MOV   A,L
        OUT   Port address of DAC
        INR   L
        JNZ   L1
        MVI   L,FF
L2:     MOV   A,L
        OUT   Port address of DAC
        DCR   L
        JNZ   L2
        JMP   START
```

RESULT:

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8085 trainer kit.

CYCLE II

Ex. No. 10. Study of Multiplexer IC 74151

Date:

Aim:

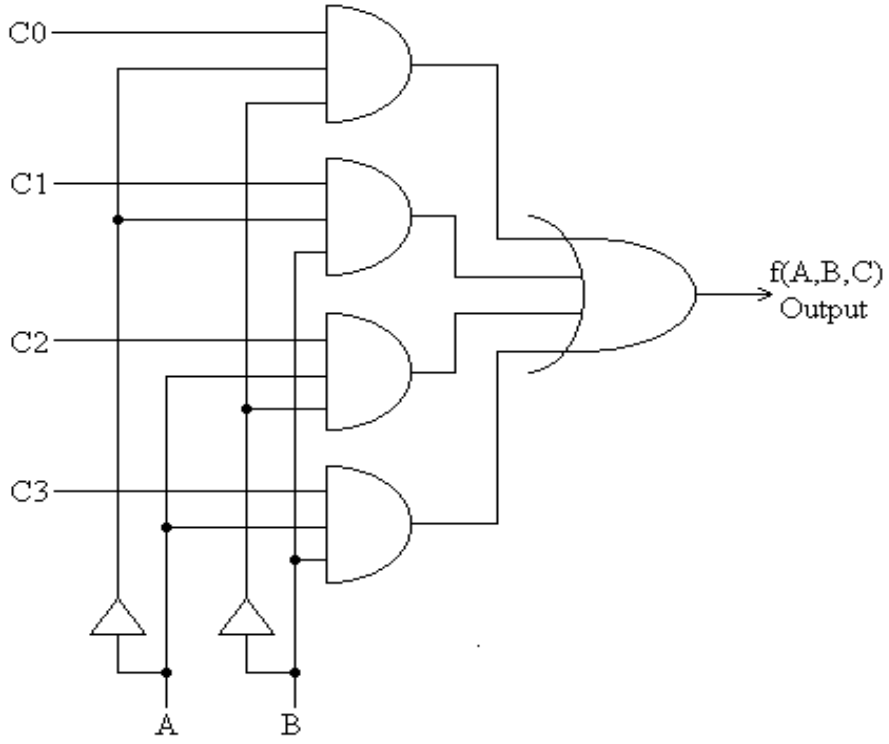
To study 4:1 multiplexer operation using IC 74151components.

Study:

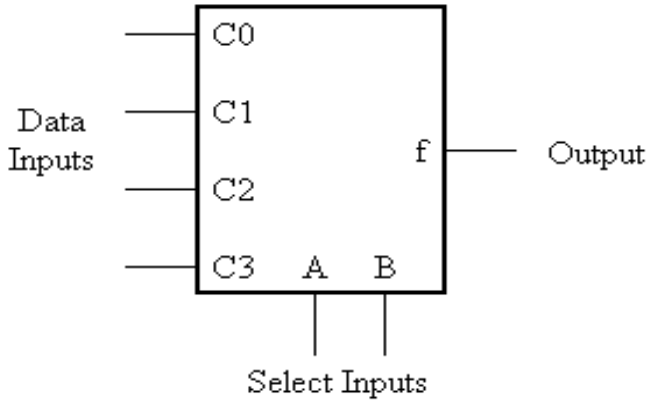
MULTIPLEXERS

A multiplexer performs the function of selecting the input on any one of 'n' input lines and feeding this input to one output line. Multiplexers are used as one method of reducing the number of integrated circuit packages required by a particular circuit design. This in turn reduces the cost of the system.

Assume that we have four lines, ***C*0, *C*1, *C*2 and *C*3**, which are to be multiplexed on a single line, ***Output (f)***. The four input lines are also known as the ***Data Inputs***. Since there are four inputs, we will need two additional inputs to the multiplexer, known as the ***Select Inputs***, to select which of the ***C*** inputs is to appear at the output. Call these select lines ***A*** ***and*** ***B***.
The gate implementation of a 4-line to 1-line multiplexer is shown below:



The circuit symbol for the above multiplexer is:



Result:

Thus the truth table for multiplexer IC 74151 is studied

Ex. No. 11. **Study of Demultiplexer IC 74154**

Date:

Aim:

To study 1:8 Demultiplexer operation using IC 74154 components.

Study:

DEMULTIPLEXERS

A Demultiplexer (DMUX) is a device which essentially performs the opposite operation to the MUX. That is, it functions as an electronic switch (/data distributor) to route an incoming data signal to one of several outputs. Figure 2-15 shows the logic symbol for the 1-line-to-4-line Demultiplexer circuit and Table 2-15 list the associated Truth table. The corresponding logic circuit implementation is then shown in Figure 2-16.

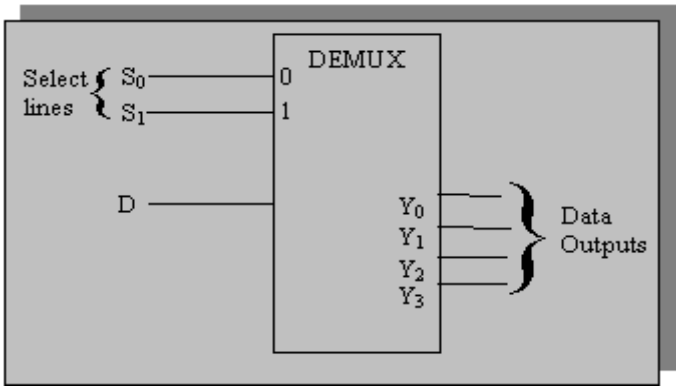


Figure 2-15 Logic symbol for 1-line-to-4-line Demultiplexer

	Address		**Outputs**			
Data	**S_1**	**S_0**	**Y_0**	**Y_1**	**Y_2**	**Y_3**
D	0	0	D	0	0	0
D	0	1	0	D	0	0
D	1	0	0	0	D	0
D	1	1	0	0	0	D

Table 2 -15 Demultiplexer Function Tables

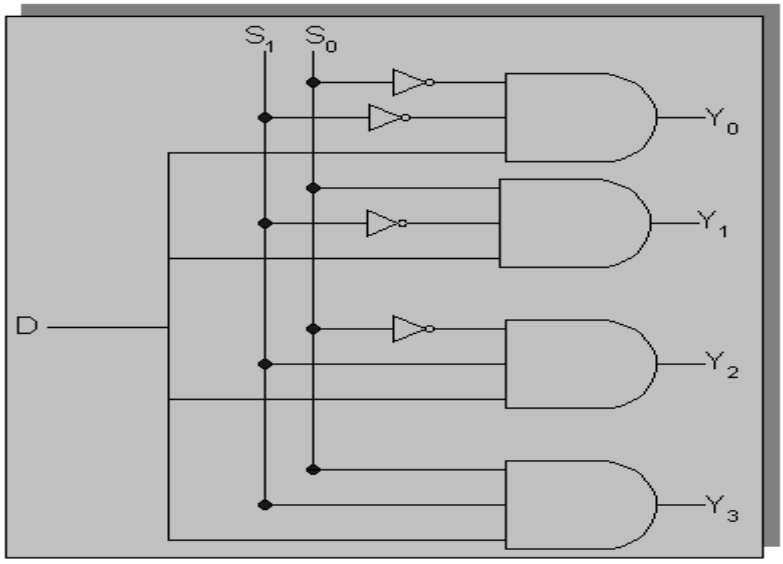


Figure 2-16 1-line-to-4-line Demultiplexer

Result:

Thus the truth table for Demultiplexer IC 74154 is studied

Ex. No. 13. **Study of Encoder**

Date:

Aim:

To study the encoder function using IC 74147.

Study:

An **encoder** is a device used to change a signal (such as a bit stream) or data into a code. The code may serve any of a number of purposes such as compressing information for transmission or storage, encrypting or adding redundancies to the input code, or translating from one code to another. This is usually done by means of a programmed algorithm, especially if any part is digital, while most analog encoding is done with analog circuitry.

Single bit 4 to 2 Encoder

A single bit 4 to 2 encoder takes in 4 bits and outputs 2 bits. It is assumed that there are only 4 types of input signals these are : 0001, 0010, 0100, 1000.

I_3	I_2	I_1	I_0	O_1	O_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

4 to 2 encoder

Priority encoder

A priority encoder prioritizes more significant bits in the data stream, and once it finds a high signal will ignore all other bits. An example of a single bit 4 to 2 encoder is shown.

I_3	I_2	I_1	I_0	O_1	O_0
0	0	0	d	0	0
0	0	1	d	0	1
0	1	d	d	1	0
1	d	d	d	1	1

4 to 2 priority encoder

Ex. No. 14. **Study of Decoder**

Date:

Aim:

To study the decoder function using IC 74147.

Study:

DECODER

A decoder is a device which does the reverse of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode.

In digital electronics this would mean that a decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. e.g. n-to-2^n, BCD decoders.

Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding.

The simplest decoder circuit would be an AND gate because the output of an AND gate is "High" (1) only when all its inputs are "High".

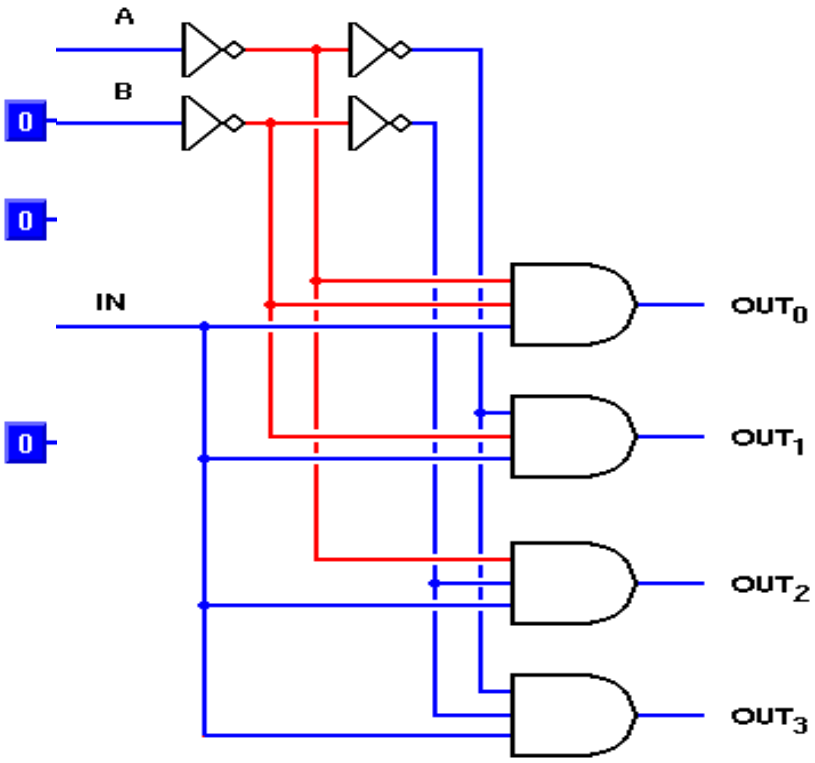
Example: A 2-to-4 Line Single Bit Decoder

A slightly more complex decoder would be the n-to-2^n type binary decoders. These type of decoders are combinational circuits that convert binary information from 'n' coded inputs to a maximum of 2^n unique outputs. We say a *maximum* of 2^n outputs because in case the 'n' bit coded information has unused bit combinations, the decoder may have less than 2^n outputs. We can have 2-to-4 decoder, 3-to-8 decoder or 4-to-16 decoder. We can form a 3-to-8 decoder from two 2-to-4 decoders (with enable signals).

A 2-to-4 line decoder/demultiplexer is shown below.

As a decoder, this circuit takes an *n*-bit binary number and produces an output on one of 2^n output lines. It is therefore commonly defined by the number of addressing input lines and the number of data output lines. Typical decoder/demultiplexer ICs might contain two 2-to-4 line circuits, a 3-to-8 line circuit, or a 4-to-16 line circuit. One exception to the binary nature of this circuit is the 4-to-10 line

decoder/demultiplexer, which is intended to convert a BCD (Binary Coded Decimal) input to an output in the 0-9 range.



If you use this circuit as a demultiplexer, you may want to add data latches at the outputs to retain each signal while the others are being transmitted.

Result:

Thus the Encoder is studied using IC 74147

CYCLE III Study MIL STD 1553B data bus

Aim:

To study architecture, transfer modes and coupling methods of MIL STD 1553B data bus.

MIL-STD-1553 & 1773 DATA BUS

PURPOSE

In recent years, the use of digital techniques in aircraft equipment has greatly increased, as have the number of avionics subsystems and the volume of data processed by them.

Because analog point-to-point wire bundles are inefficient and cumbersome means of interconnecting the sensors, computers, actuators, indicators, and other equipment onboard the modern military vehicle, a serial digital multiplex data bus was developed. MIL-STD-1553 defines all aspects of the bus, therefore, many groups working with the military tri-services have chosen to adopt it.

The 1553 multiplex data bus provides integrated, centralized system control and a standard interface for all equipment connected to the bus. The bus concept provides a means by which all bus traffic is available to be accessed with a single connection for testing and interfacing with the system. The standard defines operation of a serial data bus that interconnects multiple devices via a twisted, shielded pair of wires. The system implements a command-response format.

MIL-STD-1553, "Aircraft Internal Time-Division Command/Response Multiplex Data Bus," has been in use since 1973 and is widely applied. MIL-STD-1553 is referred to as "1553" with the appropriate revision letter (A or B) as a suffix. The basic difference between the 1553A and the 1553B is that in the 1553B, the options are defined rather than being left for the user to define as required. It was found that when the standard did not define an item, there was no coordination in its use. Hardware and software had to be redesigned for each new application. The primary goal of the 1553B was to provide flexibility without creating new designs for each new user. This was accomplished by specifying the electrical interfaces explicitly so that compatibility between designs by different manufacturers could be electrically interchangeable.

The Department of Defense chose multiplexing because of the following advantages:

- Weight reduction
- Simplicity
- Standardization
- Flexibility

Some 1553 applications utilize more than one data bus on a vehicle. This is often done, for example, to isolate a Stores bus from a Communications bus or to construct a bus system capable of interconnecting more terminals than a single bus could accommodate. When multiple buses are used, some terminals may connect to both buses, allowing for communication between them.

MULTIPLEXING

Multiplexing facilitates the transmission of information along the data flow. It permits the transmission of several signal sources through one communications system.

BUS

The bus is made up of twisted-shielded pairs of wires to maintain message integrity. MIL-STD-1553 specifies that all devices in the system will connect to a redundant pair of buses. This provides a second path for bus traffic should one of the buses be damaged. Signals are only allowed to appear on one of the two buses at a time. If a message cannot be completed on one bus, the bus controller may switch to the other bus. In some applications more than one 1553 bus may be implemented on a given vehicle. Some terminals on the bus may actually connect to both buses.

BUS COMPONENTS

There are only three functional modes of terminals allowed on the data bus: the bus controller, the bus monitor, and the remote terminal. Devices may be capable of more than one function. Figure 1 illustrates a typical bus configuration.

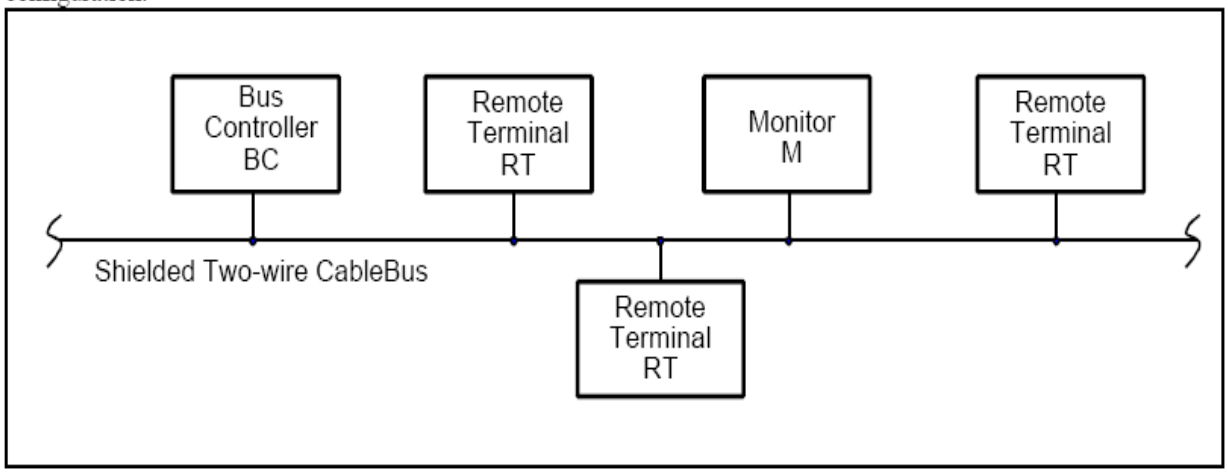


Figure 1. 1553 Bus Structure

- Bus Controller - The bus controller (BC) is the terminal that initiates information transfers on the data bus. It sends commands to the remote terminals which reply with a response. The bus will support multiple controllers, but only one may be active at a time. Other requirements, according to 1553, are: (1) it is "the key part of the data bus system," and (2) "the sole control of information transmission on the bus shall reside with the bus controller."

- Bus Monitor - 1553 defines the bus monitor as "the terminal assigned the task of receiving bus traffic and extracting selected information to be used at a later time." Bus monitors are frequently used for instrumentation.

- Remote Terminal - Any terminal not operating in either the bus controller or bus monitor mode is operating in the remote terminal (RT) mode. Remote terminals are the largest group of bus components.

MODULATION

The signal is transferred over the data bus using serial digital pulse code modulation.

DATA ENCODING

The type of data encoding used by 1553 is Manchester II biphase.

- A logic one (1) is transmitted as a bipolar coded signal 1/0 (in other words, a positive pulse followed by a negative pulse).
- A logic zero (0) is a bipolar coded signal 0/1 (i.e., a negative pulse followed by a positive pulse).

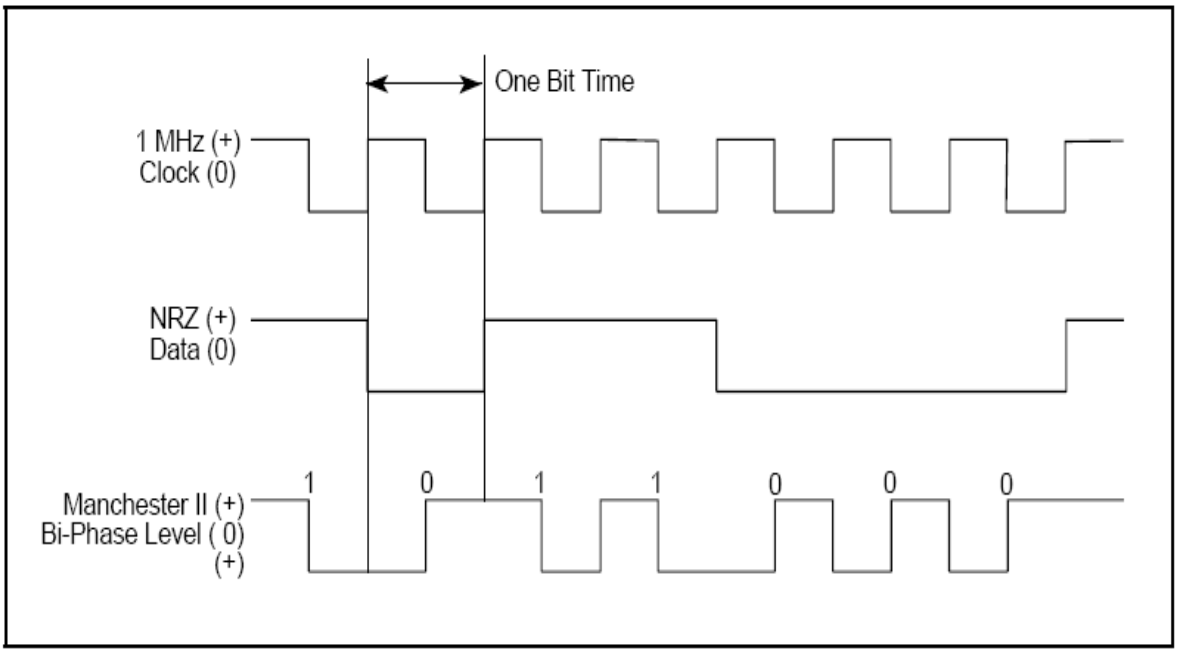


Figure 2. Data Encoding

A transition through zero occurs at the midpoint of each bit, whether the rate is a logic one or a logic zero. Figure 2 compares a commonly used Non Return to Zero (NRZ) code with the Manchester II biphase level code, in conjunction with a 1 MHz clock.

BIT TRANSMISSION RATE

The bit transmission rate on the bus is 1.0 megabit per second with a combined accuracy and long-term stability of +/- 0.1%. The short-term stability is less than 0.01%.

There are 20 1.0-microsecond bit times allocated for each word. All words include a 3 bit-time sync pattern, a 16-bit data field that is specified differently for each word type, and 1 parity check bit.

WORD FORMATS

Bus traffic or communications travels along the bus in words. A word in MIL-STD-1553 is a sequence of 20 bit times consisting of a 3 bit-time sync wave form, 16 bits of data, and 1 parity check bit. This is the word as it is transmitted on the bus; 1553 terminals add the sync and parity before transmission and remove them during reception. Therefore, the nominal word size is 16 bits, with the most significant bit (MSB) first.

There are three types of words: command, status, and data. A packet is defined to have no intermessage gaps. The time between the last word of a controller message and the return of the terminal status byte is 4-12 microseconds. The time between status byte and the next controller message is undefined. Figure 3 illustrates these three formats.

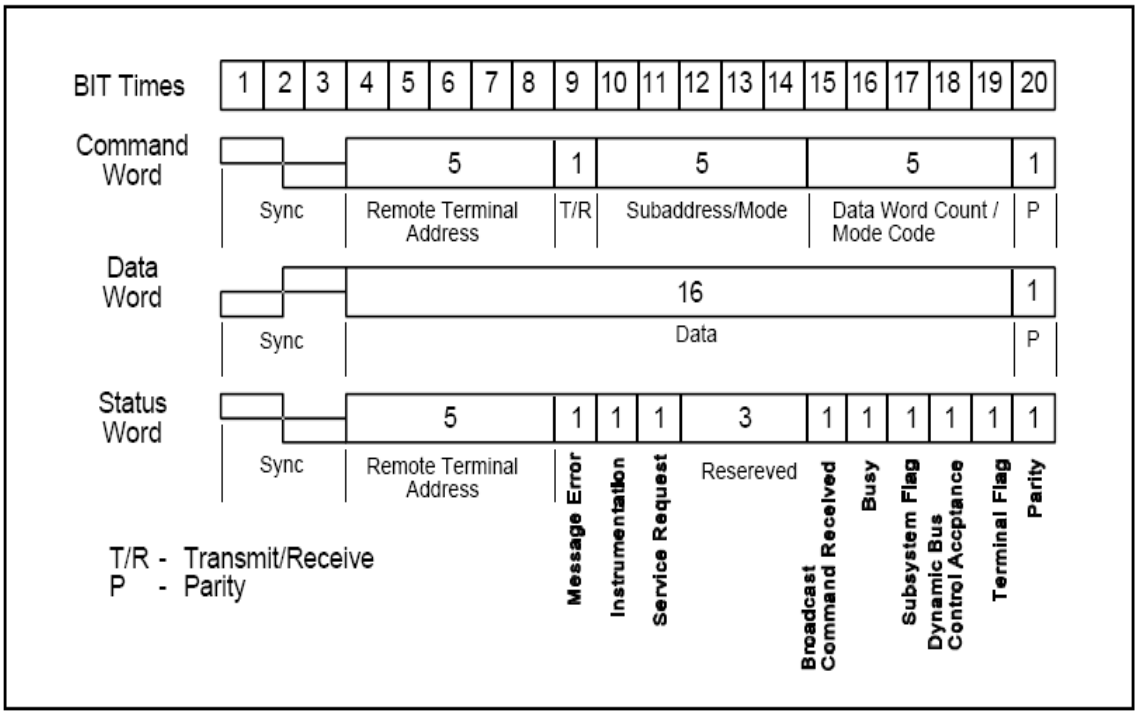


Figure 3. 1553 Word Formats

COMMAND WORD

Command words are transmitted only by the bus controller and always consist of:

- 3 bit-time sync pattern
- 5 bit RT address field
- 1 Transmit/Receive (T/R) field
- 5 bit subaddress/mode field
- 5 bit word count/mode code field
- 1 parity check bit.

DATA WORD

Data words are transmitted either by the BC or by the RT in response to a BC request. The standard allows a maximum of 32 data words to be sent in a packet with a command word before a status response must be returned. Data words always consist of:

- 3 bit-time sync pattern (opposite in polarity from command and status words)
- 16 bit data field
- 1 parity check bit.

STATUS WORD

Status words are transmitted by the RT in response to command messages from the BC and consist of:

- 3 bit-time sync pattern (same as for a command word)
- 5 bit address of the responding RT
- 11 bit status field
- 1 parity check bit.

The 11 bits in the status field are used to notify the BC of the operating condition of the RT and subsystem.

INFORMATION TRANSFERS

Three basic types of information transfers are defined by 1553:

- Bus Controller to Remote Terminal transfers
- Remote Terminal to Bus Controller transfers
- Remote Terminal to Remote Terminal transfers

These transfers are related to the data flow and are referred to as messages. The basic formats of these messages are shown in Figure 4.

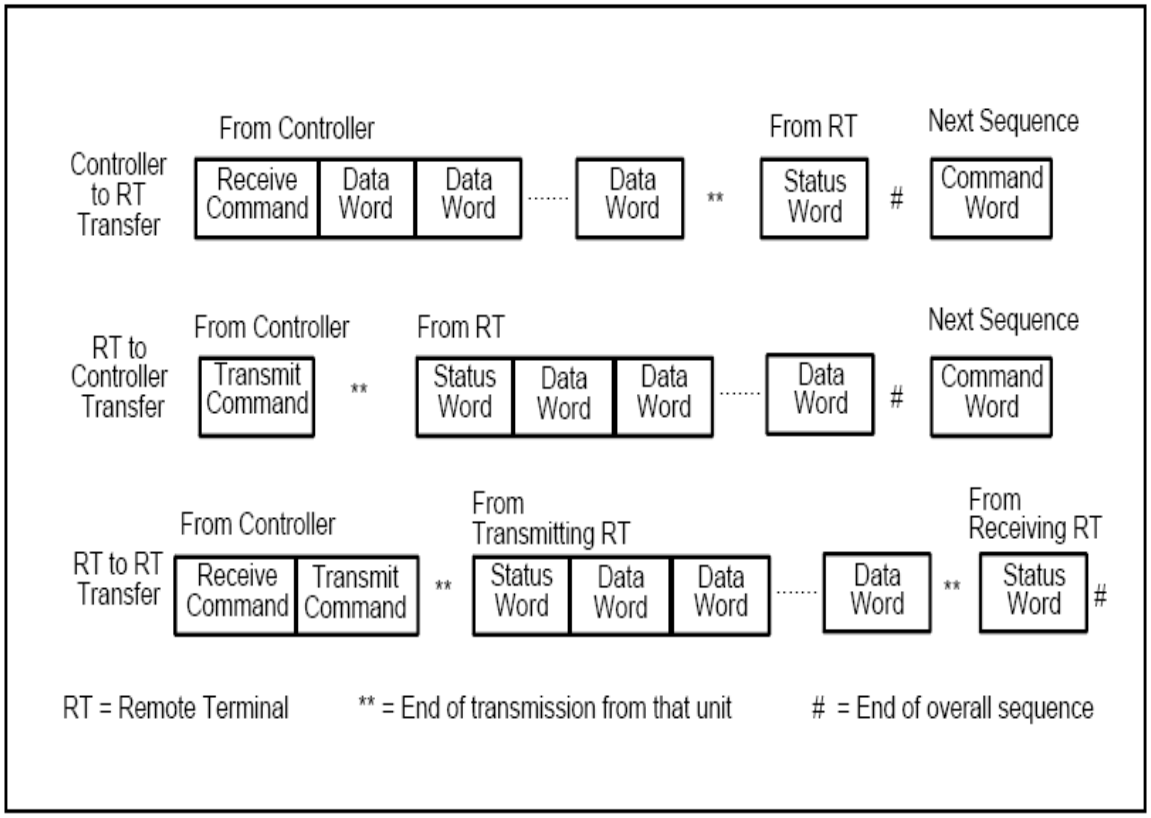


Figure 4. 1553 Data Message Formats

The normal command/response operation involves the transmission of a command from the BC to a selected RT address. The RT either accepts or transmits data depending on the type (receive/transmit) of command issued by the BC. A status word is transmitted by the RT in response to the BC command if the transmission is received without error and is not illegal.

Figure 5 illustrates the 1553B Bus Architecture in a typical aircraft.

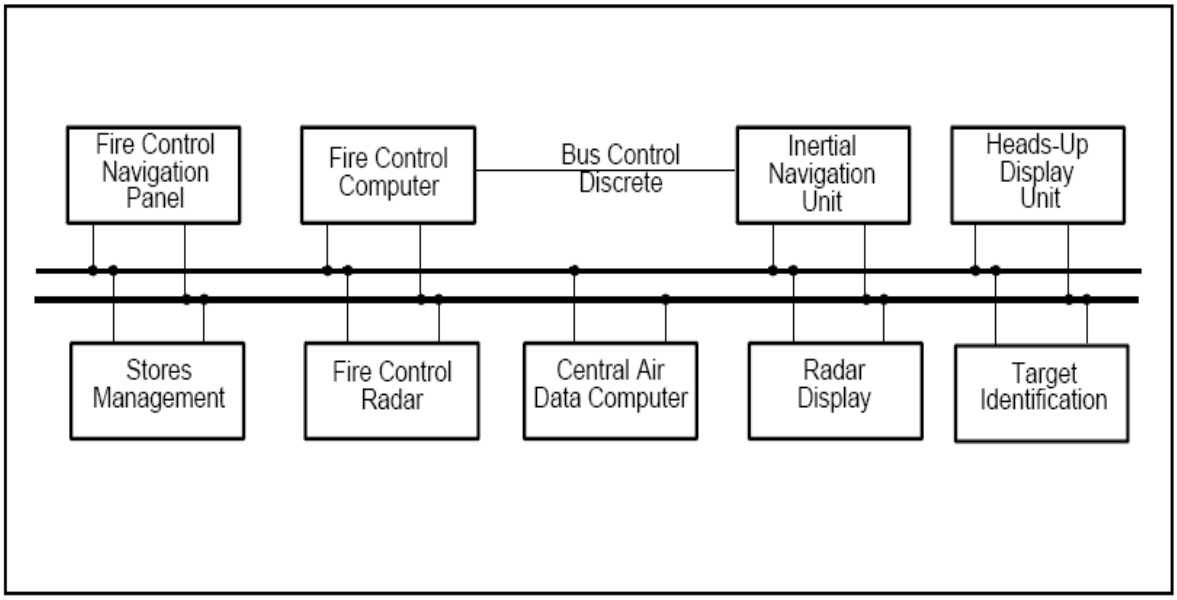


Figure 5. Typical Bus Architecture

MIL-STD-1773

MIL-STD-1773 contains the requirements for utilizing a fiber optic "cabling" system as a transmission medium for the MIL-STD-1553B bus protocol. As such, the standard repeats MIL-STD-1553 nearly word-for-word. The standard does not specify power levels, noise levels, spectral characteristics, optical wavelength, electrical/optical isolation or means of distributing optical power. These must be contained in separate specifications for each intended use.

Data encoding and word format are identical to MIL-STD-1553, with the exception that pulses are defined as transitions between 0 (off) and 1 (on) rather than between + and - voltage transitions since light cannot have a negative value.

Since the standard applies to cabling only, the bus operates at the same speed as it would utilizing wire. Additionally, data error rate requirements are unchanged.

Different environmental considerations must be given to fiber optic systems. Altitude, humidity, temperature, and age affects fiber optics differently than wire conductors. Power is divided evenly at junctions which branch and connectors have losses just as wire connectors do.