ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

V – SEMESTER

CSCP509 – MICROPROCESSORS LAB

Name : ____

Reg. No. : ____

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

V – SEMESTER

CSCP509 – MICROPROCESSORS LAB

Certified that this is a bonafide record of work done by Mr./Ms.____

Reg. No.____

of B.E. (Computer Science and Engineering) in the CSCP508 – Microprocessors Lab during the odd semester of the academic year 2021 – 2022.

Staff in-charge　　　　Internal Examiner　　　　External Examiner

Annamalai nagar

Date: / / 2021

INDEX

EX.NO: 1 DATE:	STUDY OF 8086 MICROPROCESSORREGISTERS AND INSTRUCTION SETS

AIM:

To study 8086 microprocessor registers in 8086 and its instruction set.

DESCRIPTION:

The Intel 8086 is a 16-bit microprocessor used as a CPU in a microcomputer it has a 16-bit data bus. So that it can read 16 data from (or) write 16 data to memory. The 8086 has a 20-bit address bus and can address any one of the 2^{20} memory location words will be stored in bus consecutive memory location. If the 1st byte of a word is at an even address then the 8086 can read entire word in one operation. If the 1st byte of a word is at an odd address then the 8086 will read first byte in one operation.

INTERNAL ARCHITECTURE:

The 8086 CPU is divided into 2 independent functional part the bus interface unit and execution unit.

BUS INTERFACE UNIT:

The bus interface unit sends out addresses, fetches instructions from memory, reads data from ports and memory and writes data into ports and memory. It handles all transfer of data and addresses on the buffer for execution.

SEGMENT REGISTERS:

The BIU contains four 16-bit segment registers. They are

1. The code segment register.
2. The stack segment register.
3. The extra segment register.
4. The data segment register.

QUEUE:

To speed up program execution the BIU fetches as many as six instructions ahead of time from memory. The projected instruction bytes are held for the EU in a queue fetching the next instruction while the current instruction execution is called pipelining.

INSTRUCTION POINTER:

This register holds the 16-bit address of the next code byte within this code segment. The value contained in the instruction pointer is called OFFSET.

EXECUTION UNIT:

The EU of the 8086 tells the BIU where to fetch the instruction or data from decoder instruction to execute the instruction. The component of the EU are control circuitry, instruction decoder and ALU. The EU has 16-bit ALU which can Add, Subtract, AND, OR, XOR and INC etc.

FLAG REGISTER:

A 16-bit flag register in the EU contains active flags. A flag is a Flip-flop, which indicates some condition, produced by the execution of an instruction.

They are,

(i) C-Carry (ii) A-Auxiliary carry (iii) S-Sign (iv) P-Parity (v) Z-Zero

(vi) T-Trap (vii) I-Interrupt (viii) D-Direction (ix) O-Overflow.

GENERAL PURPOSE REGISTERS (GPR):

The EU has 8 general-purpose registers to store 8-bit data. They can be combined to store 16-bit data for 8-bit data storage; the registers are AH, AL, BH, BL, CH, CL, DH, and DL. For 16-bit data the registers are AX, BX, CX, DX, where AX is the accumulator.

POINTER AND INDEX REGISTER:

The EU register contains 16-bit source index (SI) 16-bit destination index registers (DI) and 16-bit base pointer registers. These can be used for temporary storage of data. But their main use is to hold the 16-bit offset of a data word in one of the registers.

ADDRESSING MODES:

The way in which the operand is specified is called an addressing mode. The addressing modessupported by 8086 are:

1. Immediate addressing mode
2. Direct addressing mode
3. Register addressing mode
4. Register indirect addressing mode
5. Register relative addressing mode
6. Based index addressing mode
7. Relative based indexed addressing mode.

8086 INSTRUCTION SET:

After we get the structure of a program worked and written down, the next step is todetermine the instruction statements required to do each part of the program.

DATA TRANSFER INSTRUCTIONS:

MOV : Copy byte from specified source to destination.

PUSH : Copy specified used to top of stack.

POP : Copy word from top of stack.

XCHG : Exchange bytes.

XLAT : Translate a byte in ALU using a table a memory.

INPUT/OUTPUT PORT TRANSFER INSTRUCTIONS:

IN Copy a byte from port to accumulator.

OUT Copy a byte from accumulator to port.

SPECIAL ADDRESS TRANSFER INSTRUCTIONS:

LEA Load effective address of operand into register.

LDS Load DS register and other register from memory.

LES Load ES register and other register from memory.

FLAG TRANSFER INSTRUCTIONS:

LAHF Load alt with the low byte of the register.

SAHF Store alt register to low byte of flag register.

ARITHMETIC INSTRUCTIONS:

ADD Add specified byte to byte.

ADC Add byte+byte+carry flag.

INC Increment specified byte.

SUBTRACTION INSTRUCTIONS:

SUB. Subtract byte from byte.

SBB Subtract byte and carry flag from byte.

CMP Compare two specified bytes or two specified words.

MULTIPLICATION INSTRUCTIONS:

MUL Multiply unsigned byte by byte.

IMUL Multiply signed byte by byte.

DIVISION INSTRUCTIONS:

DIV Divide unsigned byte by byte.

IDIV Divide signed byte by byte.

LOGICAL INSTRUCTIONS:

NOT Invert each bit of a word.

AND AND each bit in a byte with the cell. But in another byte.

OR OR each bit in a byte with the cell. But in another byte.

SHIFT INSTRUCTIONS:

SHL/SAL Shift bit of word left, put zeroes in LSB.

SHR Shift bit of byte right, put zeroes in MSB.

SAR Shift bits of word right, copy old MSB to LSB.

ROTATE INSTRUCTIONS:

ROL Rotate bits of byte, left MSB to LSB and CF.

ROR Rotate bits of byte, right, MSB to LSB and CF.

STRING INSTRUCTIONS:

REP An instruction prefix. Repeat following instruction until CX=0

REPE An instruction prefix. Repeat following instruction until CX=0,and ZF=/!.

OUTS/OUTSB/OUTSW Output string byte or word to port.

PROGRAM EXECUTION TRANSFER INSTRUCTIONS:

(i) Uncondition Instruction Transfer:

CALL Call a procedure save return address on stack.

RET Return from procedure to calling program.

JMP Goto specified address to get next instruction.

(ii) Conditional Transfer Instructions:

JA Jump if above.

JAE Jump if above or equal.

JBE Jump if below or equal.

JC Jump if carry.

JE Jump if equal.

JNC Jump if no carry.

ITERATION CONTROL INSTRUCTIONS:

These instructions can be used to execute a series of instructions and number of iterationsthat recall the specified instruction.

LOOP Loop three a sequence of instruction until LX=0.

JNZ Jump to specified address if CX=0.

INTERRUPT INSTRUCTIONS:

INT Interrupt program execution call service procedure.

INTO Interrupt program execution if OF=1.

HIGH LEVEL LANGUAGE INTERFACE INSTRUCTION:

ENTER Enter procedure.

LEAVE Leave procedure.

BOUND Check if effective address within specified array bound.

EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS:

HLT: Halt until interrupt or reset.

WAIT: Wait until signal on the test pin.

ESC: Escape to external coprocessor.

RESULT:

Thus, the internal architecture and instruction set of 8086 microprocessor registers andinstruction sets have been studied.

EX.NO: 2(a) **DATE:**	**PERFORM 16-BIT ADDITION OPERATION**

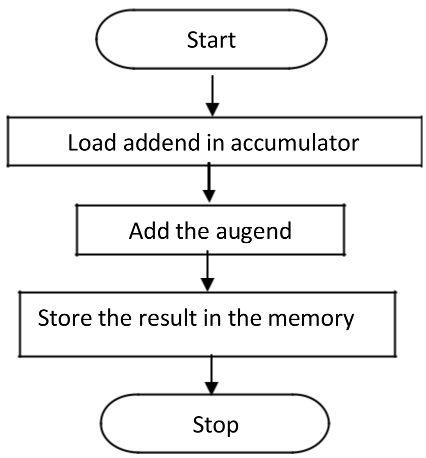
Aim:

To perform 16-bit addition operation using 8086 microprocessor kit

Theory:

Move the first data to accumulator. Then using the add instruction 16 bit addition is performed.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
A2 dw 1102h
A3 dw 1104h
datahere ends
codehere segment
assume cs:codehere, ds:datahere
ORG 1000h
MOV AX, [A1]
ADD AX, [A2]
MOV [A3], AX
HLT
```

```
codehere ends
end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 11	MOV AX,[A1]	Addend in AX
1003 1004 1005 1006	03 06 02 11	ADD AX,[A2]	Add
1007 1008 1009	A3 00 12	MOV [A3],AX	Store the result
100A	F4	HLT	Halt

Sample Data:

Input:	**Output:**
[1100] =11	[1200]=33
[1101]=11	[1201]=33
[1102]=22	
[1103]=22	

Result:

The 16-bit Arithmetic operation for addition is performed using 8086 microprocessor kit.

EX.NO: 2(b) DATE:	PERFORM 16-BIT SUBTRACTION OPERATION

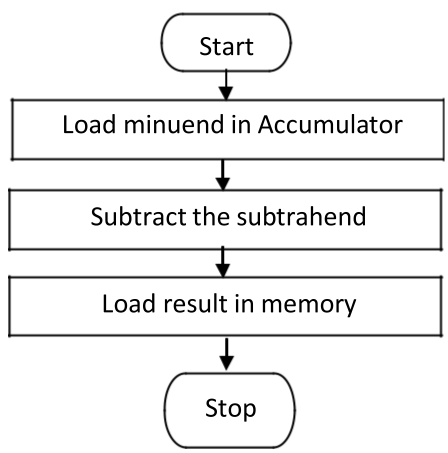
Aim:

To Perform 16-bit subtraction operation using 8086 microprocessor kit.

Theory:

Move the minuend to a register pair. Then using the sub instruction 16 bit subtractionis performed.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
A2 dw 1102h
A3 dw 1104h
datahere ends
codehere segment
assume cs: codehere, ds:datahere
ORG 1000h
MOV AX, [A1]
SUB AX, [A2]
MOV [A3], AX
HLT
```

```
codehere ends
end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 11	MOV AX,[A1]	Minuend in AX
1003 1004 1005 1006	2B 06 02 11	SUB AX,[A2]	Subtract
1007 1008 1009	A3 00 12	MOV [A3],AX	Store the result
100A	F4	HLT	Halt

Sample Data:

Input:

[1100] =33

[1101]=33

[1102]=22

[1103]=22

Output:

[1200]=11

[1201]=11

Result:

The 16-bit Arithmetic operation for subtraction is performed using 8086 microprocessor kit

EX.NO: 3(a) DATE:	PERFORM 16-BIT MULTIPLICATION OPERATION

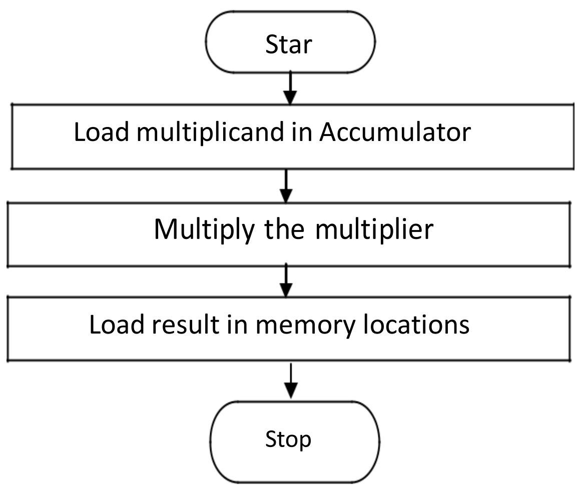
Aim :

To Perform 16-bit multiplication operation using 8086 microprocessor kit.

Theory:

Assign multiplicand and multiplier to memory location. Move the multiplicand to register AX.Multiply with multiplier. Store the product in AX and DX

Flowchart:



MASM code:

```
datahere segment
a dw 1500h
b dw 1502h
c dw 1504h
d dw 1506h
datahere ends
codehere segment
Assume cs: codehere, ds:datahere
ORG 1000h
MOV AX,[a]
MUL [b]
MOV [c],AX
```

```
MOV[d],DX
HLT
codehere ends
end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Comments
1000 1001 1002	A1 00 15	MOV AX,[a]	Move content of memory to accumulator
1003 1004 1005 1006	F7 26 02 15	MUL [b]	Multiply the memory content to accumulator
1007 1008 1009	A3 04 15	MOV [c],AX	Move accumulator to memory
100A 100B 100C 100D	89 16 06 15	MOV[d],DX	Move DX content to Memory
100E	F4	HLT	Halt the program

Sample Data :

Input:	**OUTPUT:**
1500 – 03h.	1504 – 06h
1501 – 00h.	1505 – 00h
1502 – 02h	
1503 – 00h	

RESULT:

Thus the 16-bit multiplication is performed using microprocessor.

EX.NO: 3(b) DATE:	PERFORM 16-BIT DIVISION OPERATION

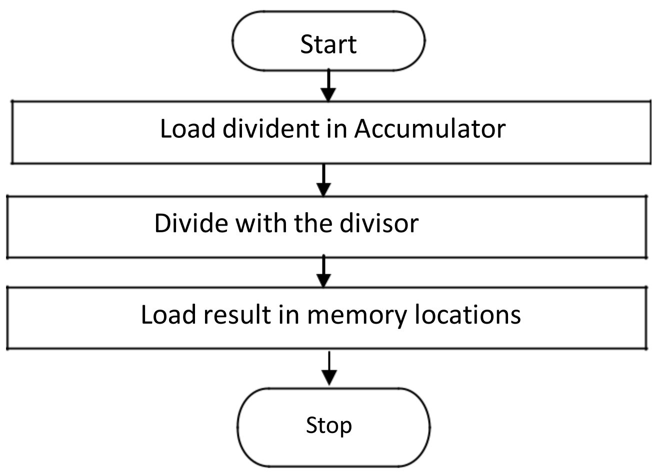
AIM :-

To perform 16-bit division operation using 8086 micropressor kit.

Theory:-

Assign dividend and divisor to memory location. Move the dividend to register AX. Divide using divisor. Store the quotient and remainder in AX and DX.

FLOWCHART:



MASM code :-

```
datahere segment

a dw1600h
b dw 1602h
c dw 1604h
d dw 1606h
datahere ends
codehere segment
assume cs:codehere,ds:datahere
ORG 2010h
MOV AX ,[a]
DIV [b]
MOV[c],AX
```

MOV[d],DX
HLT
Codehere ends
end

OPCODE TABLE:

Memory Address	Opcode	Mnemonics	Comments
2010 2011 2012	A1 00 16	MOV AX,[a]	Move the content of memory to accumulator
2013 2014 2015 2016	F7 36 02 16	DIV [b]	Divide the content of memory with accumulator
2017 2018 2019	A3 04 16	MOV[c],AX	Move accumulator to memory
201A 201B 201C 201D	89 16 06 16	MOV[d],DX	Move the DX content to memory
201E	F4	HLT	Halt the program

SAMPLE DATA :

INPUT :

1600 – 08h

1601 – 00h

1602 – 02h

1603 – 00h

OUTPUT :-

1604 – 04h

1605 – 00h

RESULT :-

Thus the 16-bit division is performed using microprocessor

EX.NO: 4 DATE:	**Calculate the length of a string**

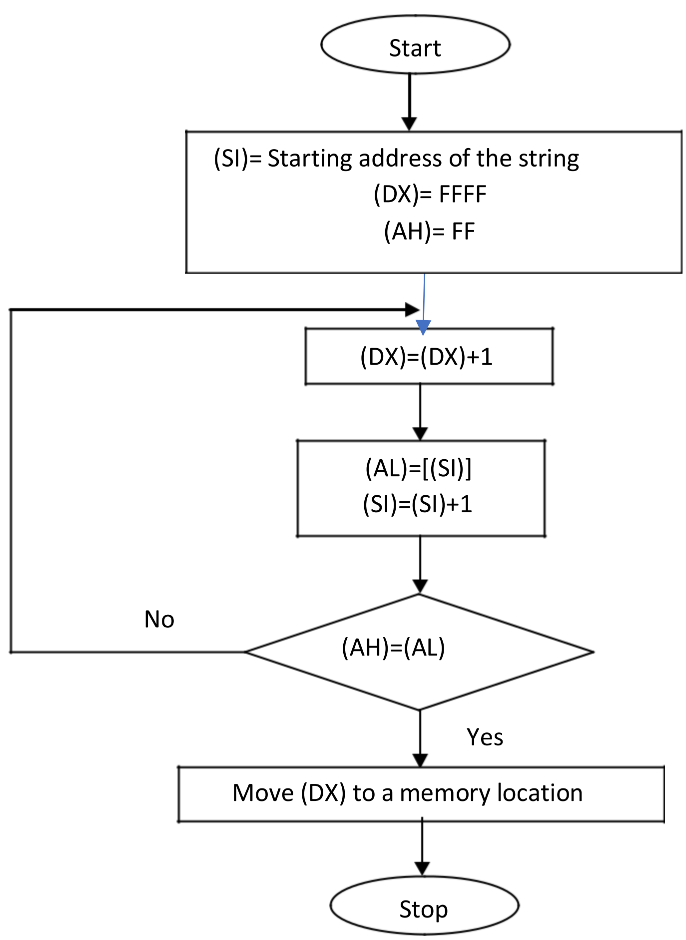
Aim :

To find the number of characters in a string.

Theory :

Addressing the string is done using SI register, and the DX register is used to store the number of characters. End of string is detected using FF. Hence each character is detected using FF.Hence each character is fetched from memory and is compared with FF. If the zero flag is set,then it denotes end of string, the count have been stored in DX, by incrementing it after each comparison.

Flowchart:



MASM code:

```
Datahere segment
A1 dw 1100h
Datahere ends
Codehere segment
Assume cs:codehere , ds: datahere
         ORG 1000h
        MOV SI,1200h
        MOV DX,0FFFFh
        MOV AH,0FFh
L1 :    INC  DX
         MOV AL,[SI]
         INC SI
          CMP AH,AL
          JNZ  L1
          MOV [A1],DX
          HLT
Codehere ends
end
```

Opcode Table:

Address	Opcode	Mnemonics	Comments
1000	BE	MOV SI,1200	Load the starting
1001	00		Address of the string in SI
1002	12		
1003	BA	MOV DX,FFFF	Initialize DX
1004	FF		
1005	B4	MOV AH,FF	Load AH with end of the string
1006	FF		
1007	42	NOTEND: INC DX	Increment count
1008	8A	MOV AL,[SI]	Get string character to AL
1009	04		
100A	46	INC SI	Increment String index
100B	3A	CMP AH,AL	Compare string with FF
100C	E0		
100D	75	JNZ NOTEND	Jump if not end
100E	F8		
100F	89		
1011	16	MOV [1200],DX	Store the length
1012	00		
1013	11		
1014	F4	HLT	Halt the process

Sample data :

Input :

1200 28

1201 13

1202 10

1203 11

1204 FF

Output:

1100 04

Result:

Thus the program to find the number of character in a string is executed and verified.

EX.NO: 5 DATE:	FIND THE SUM OF THE NUMBERS IN A WORD ARRAY

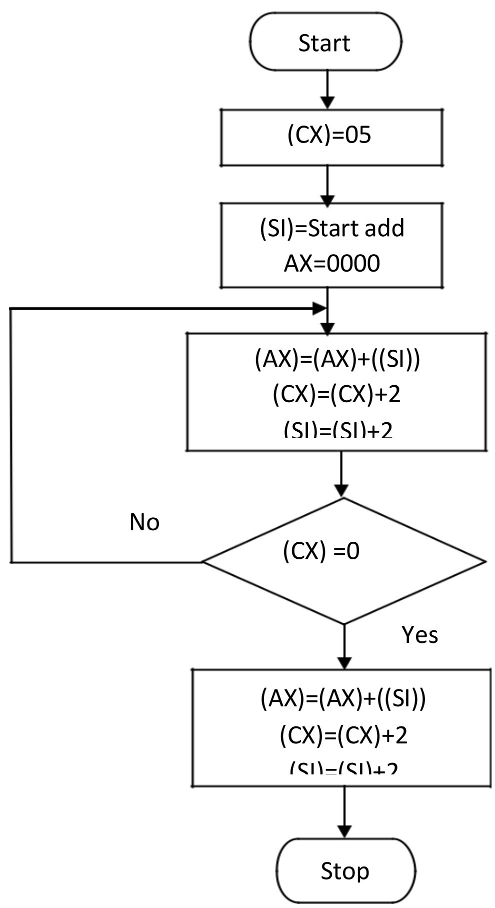
Aim:

To obtain the sum of a 16-bit array in memory, using index register and store the result in
memory.

Theory:

Initialize the index register SI with the start address and CX with the length of the array. Clear the accumulator and add the contents of SI in it. Increment the index to point to thenext word and decrement CX by 1. Repeat until CX=0 and store the sum in a memory location.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
Sum dw 1200h
datahere ends
codehere segment
assume cs: codehere, ds: datahere
                ORG 1000h
                MOV CX, 05h
                MOV AX, 0
                MOV SI,AX
        L1:     ADD AX, A1[SI]
                ADD SI, 2
                LOOP L1
                MOV [Sum], AX
                HLT
  Codehere ends
  end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Remarks
1000 1001 1002	B9 05 00	MOV CX, 05h	CX= 5 No of elements
1003 1004 1005	B8 00 00	MOV AX, 0	Clear AX
1006 1007	8B F0	MOV SI,AX	Initialize SI to start of the array
1008 1009 100A 100B	03 84 00 11	L1: ADD AX, START[A1]	Add accumulator and content of Array
100C 100D 100E	83 C6 02	ADD SI,2	
100F 1010	E2 F7	LOOP L1	Decrement CX and check if zero
1011 1012 1013	A3 00 12	MOV [Sum],AX	Store the result
1014	F4	HLT	Halt the process

Sample Data:

Input:

Number of Elements = 5.

[1100] = 0001

[1102] = 0002

[1104] = 0003

[1106] = 0004

[1108] = 0005

Output:

[1200] = 000F

RESULT:

Thus, the sum of the numbers in a word array is obtained.

EX.NO: 6 DATE:	SORTING NUMBER IN DESCENDING ORDER OF AN UNSORTED ARRAY

Aim:

To arrange an array of unsorted words in descending order.

Theory:

The algorithm used here is bubble sort. Let

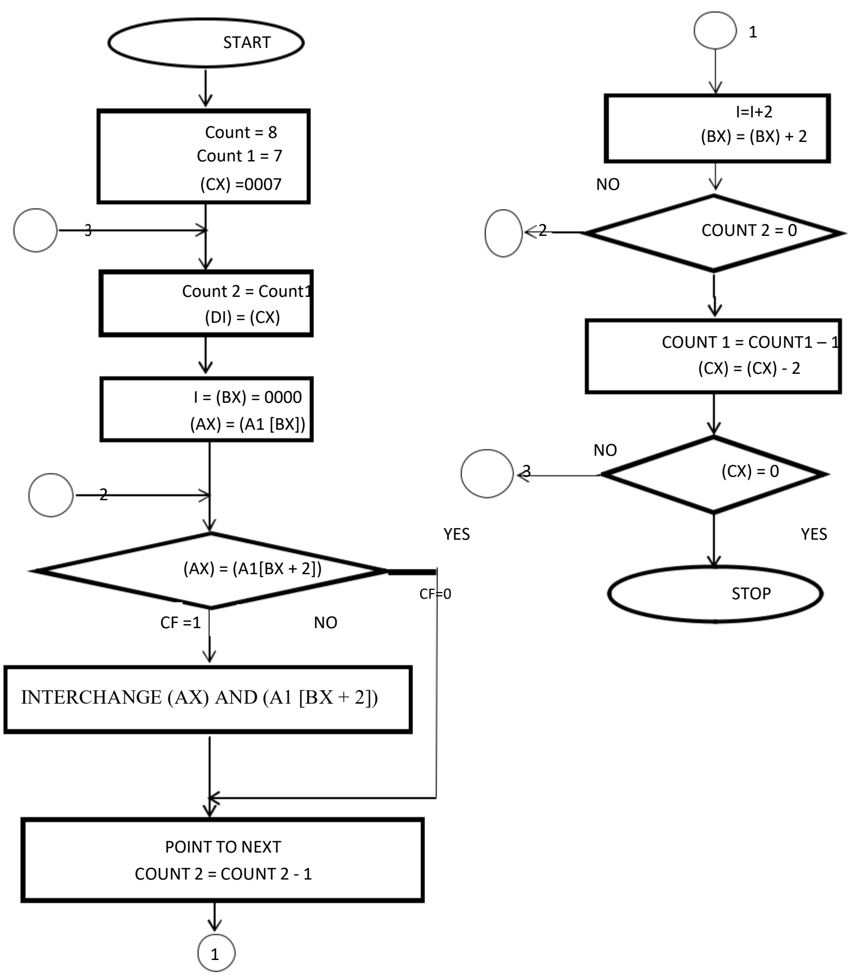
N : Number of elements in the array, content of CX.

A1 : Array name of start address of array.

I : Index in this array, here content of DI.

Start at the Beginning of the array, and considering a pair of elements at a time, put the pair in descending order. Thus arrange successive pairs of elements in descending order. After the firstpass through the array the smallest element in at the end of the array :Hence during the second pass consider only the first N – 1 element and so on.

Flowchart:



MASM code:

```
datahere segment
A1 dw 1100h
datahere ends
codehere segment
assume cs: codehere, cs: datahere
                ORG 1000h
                MOV CX, 07h
L1:             MOV D1, CX
                MOV BX, 00h
L2:             MOV AX, A1 [BX]
                CMP AX, A1 [BX + 2]
                JNC PROCEED
                XCHG AX, A1 [BX + 2]
                MOV A1 [BX], AX
PROCEED:        ADD BX, 2
                LOOP L2
                NOP
                MOV CX, DI
                LOOP L1
                HLT
Codehere ends
end
```

Opcode Table:

Memory Address	Opcode	Mnemonics	Remarks
1000 1001 1002	B9 07 00	MOV CX, 07	CX= count -1
1003 1004	8B F9	L1: MOV D1, CX	Save CX in DX
1005 1006 1007	BB 00 00	MOV BX,0	Clear BX
1008 1009 100A 100B	8B 87 00 11	L2: MOV AX, A1[BX]	

100C 100D 100E 100F	3B 87 02 11	CMP AX, A1 [BX+ 2]	Compare first to elements
1010 1011	73 08	JNC PROCEED	
1012 1013 1014 1015	87 87 02 11	XCHG AX, A1 [BX+ 2]	Interchange if less
1016 1017 1018 1019	89 87 00 11	MOV A1 [BX], AX	
101A 101B 101C	83 C3 02	PROCEED:ADD BX, 2	Increment index
101D 101E	E2 E9	LOOP L2	And proceed if not
101F	90	NOP	
1020 1021	8B CF	MOV CX, D1	Move a copy of count in CX
1022 1023	E2 DF	LOOP L1	Repeat until CX=0
1024	F4	HLT	Halt the process

Sample Input and output:

Number of Elements = 8

Unsorted array Starting from A1 = 1100

[1100]	=	0022
[1102]	=	0011
[1104]	=	0033
[1106]	=	0077
[1108]	=	0055
[110A]	=	0066
[110C]	=	0088
[110E]	=	0044

Sorted array :

[1100]	=	0088
[1102]	=	0077
[1104]	=	0066
[1106]	=	0055
[1108]	=	0044
[110A]	=	0033
[110C]	=	0022
[110E]	=	0011

RESULT:

Thus, an array of unsorted words is arranged in descending order.

EX.NO: 7 DATE:	MOVE A BYTE STRING FROM SOURCE TO DESTINATION

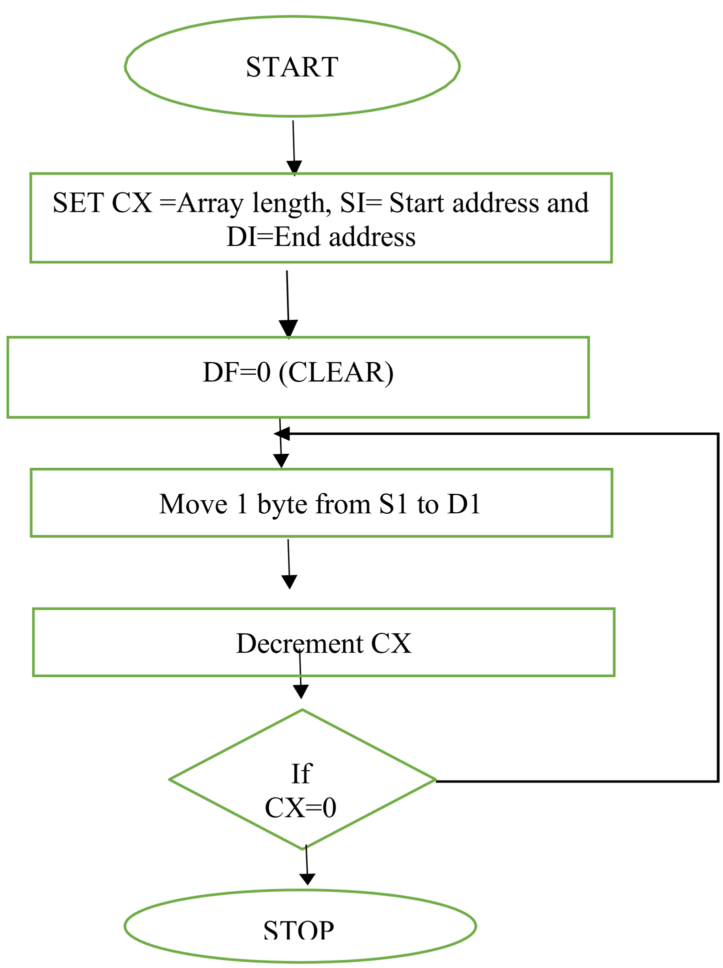
AIM :

To Move a Byte String of length FF from a source to a destination.

THEORY:

String primitives require initialization of the index registers and SI and DI registers are initialized to start of the source and start of the destination array respectively. The direction flag is cleared to facilitate auto incrementing of the index registers. The CX register is used to perform the operation repeatedly. The string primitive used in MOVSB which moves one byte from source operand to destination operand. The SI and DI registers are incremented automatically as DE=0.

FLOW CHART:



MASM Code:

```
        Datahere segment

        SOU dw 100Eh

        DES dw 110Eh

         Datahere ends

        Codehere segment

        Assume cs: codehere, ds: datahere

        ORG 1000h
        MOV SI, [SOU]
        MOV DI, [DES]
        MOV CX, 0FFH
        CLD
MOVE : MOVSB
          Loop MOVE

          HLT

        Codehere ends

        end
```

Opcode:

Memory Address	Opcode	Mnemonics	Remarks
1000 1001 1002	BE 0E 10	MOV SI, [SOU]	Move source address to SI
1003 1004 1005	BF 0E 11	MOV DI,[DES]	MOVE destination address to DI
1006 1007 1008	B9 FF 00	MOV CX, 0Ah	CX=Count=10
1009	FC	CLD	Clear the destination flag
100A	A4	MOVE: MOVSB	CX=Count=10
100B 100C	E2 FD	Loop MOVE	Repeat Until CX=0
100D	F4	HLT	Halt the process

SAMPLE DATA:

INPUT:	**OUTPUT:**
Fill the location from 100E to 10 locations with 55	[110E] to [110E]= 55
S -Array=100E	
D -Array=110E	
[100E] = 55	[110E] = 55
[100F] = 55	[110F] = 55
[1010] = 55	[1110] = 55
[1011] = 55	[1111] = 55
[1012] = 55	[1112] = 55
[1013] = 55	[1113] = 55
[1014] = 55	[1114] = 55
[1015] = 55	[1115] = 55
[1016] = 55	[1116] = 55
[1017] = 55	[1117] = 55

RESULT:

Thus the program to move string from a source to a destination is executed and verified.