TABLE OF CONTENTS

Practical 1

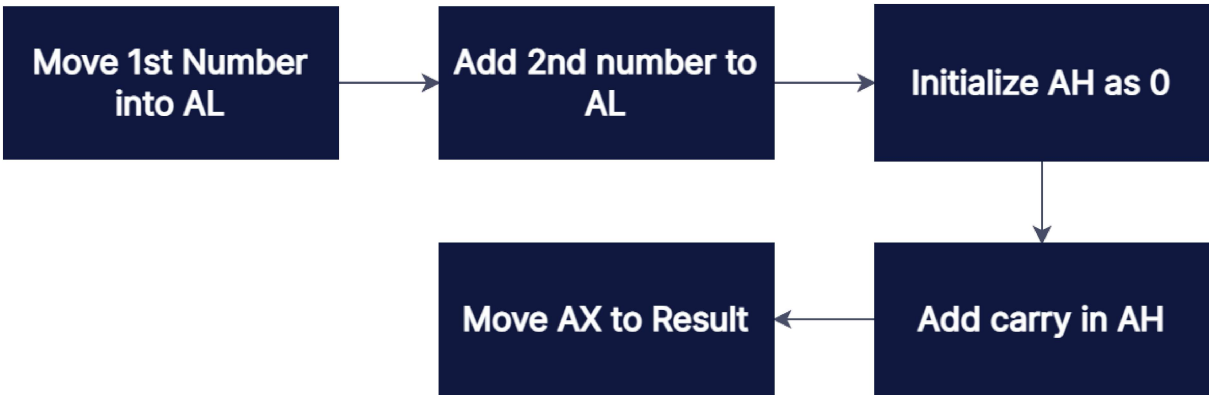
Aim -

Write a program in emu8086 to perform addition of two 8-bit numbers.

Software Required –

Emu8086

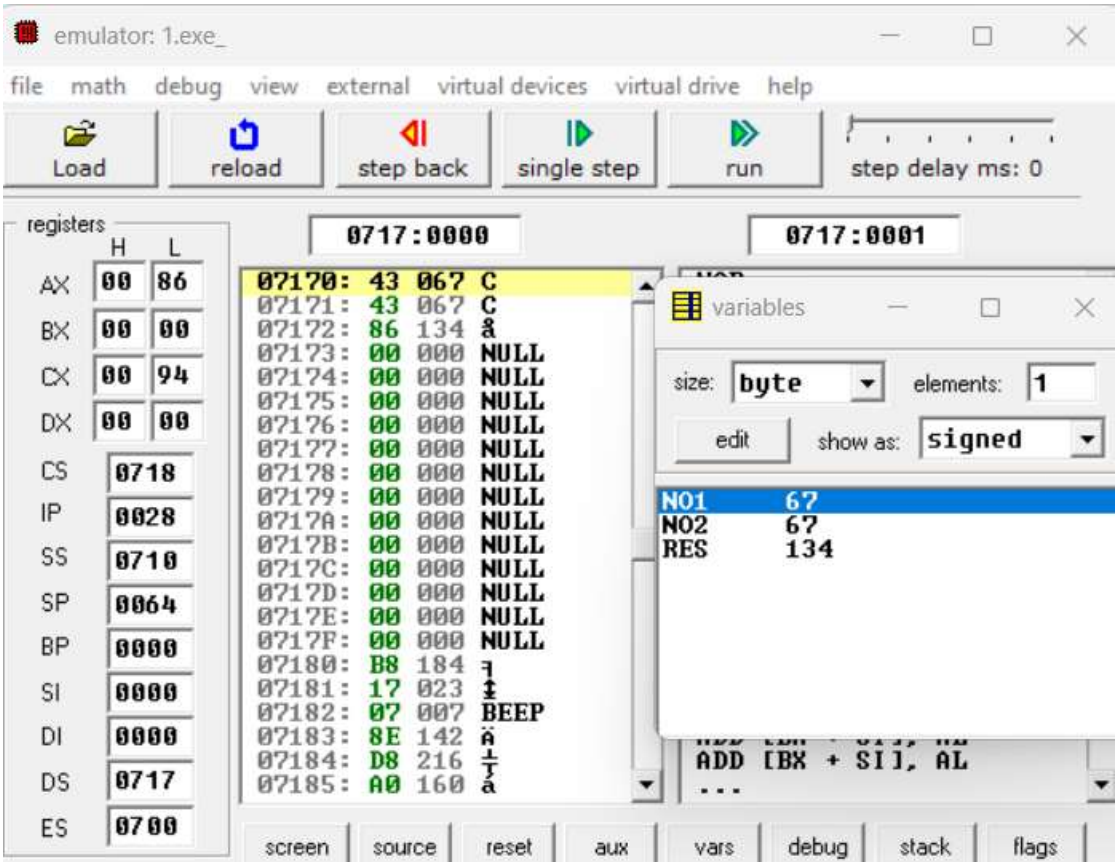
Flowchart –



Code –

```
.model small
.stack 100

.data
    no1 db 43H   ;
    no2 db 43H   ;
    res dw ?     ;

.code
START:
    MOV ax,@data ;
    Mov ds,ax;
    mov al,no1;
    add al,no2;
    mov ah,00h;
    adc ah,00h;
    mov res,ax;
    end start;
```

Output –


emulator: 1.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 00 86
BX 00 00
CX 00 94
DX 00 00
CS 0718
IP 0028
SS 0710
SP 0064
BP 0000
SI 0000
DI 0000
DS 0717
ES 0700
0717:0000
0717:0001
07170: 43 067 C
07171: 43 067 C
07172: 86 134 å
07173: 00 000 NULL
07174: 00 000 NULL
07175: 00 000 NULL
07176: 00 000 NULL
07177: 00 000 NULL
07178: 00 000 NULL
07179: 00 000 NULL
0717A: 00 000 NULL
0717B: 00 000 NULL
0717C: 00 000 NULL
0717D: 00 000 NULL
0717E: 00 000 NULL
0717F: 00 000 NULL
07180: B8 184
07181: 17 023
07182: 07 007 BEEP
07183: 8E 142 Ä
07184: D8 216
07185: A0 160 á
variables
size: byte
elements: 1
edit
show as: signed
NO1 67
NO2 67
RES 134
ADD [BX + SI], AL
...
screen source reset aux vars debug stack flags

Practical 2

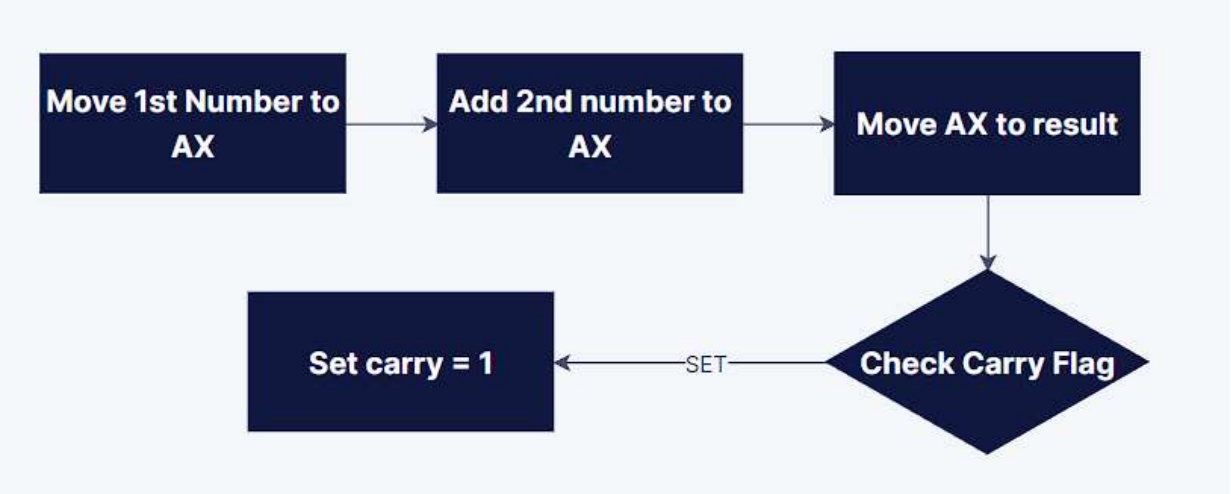
Aim -

Write a program in emu8086 to perform addition of two 16-bit numbers.

Software Required –

Emu8086

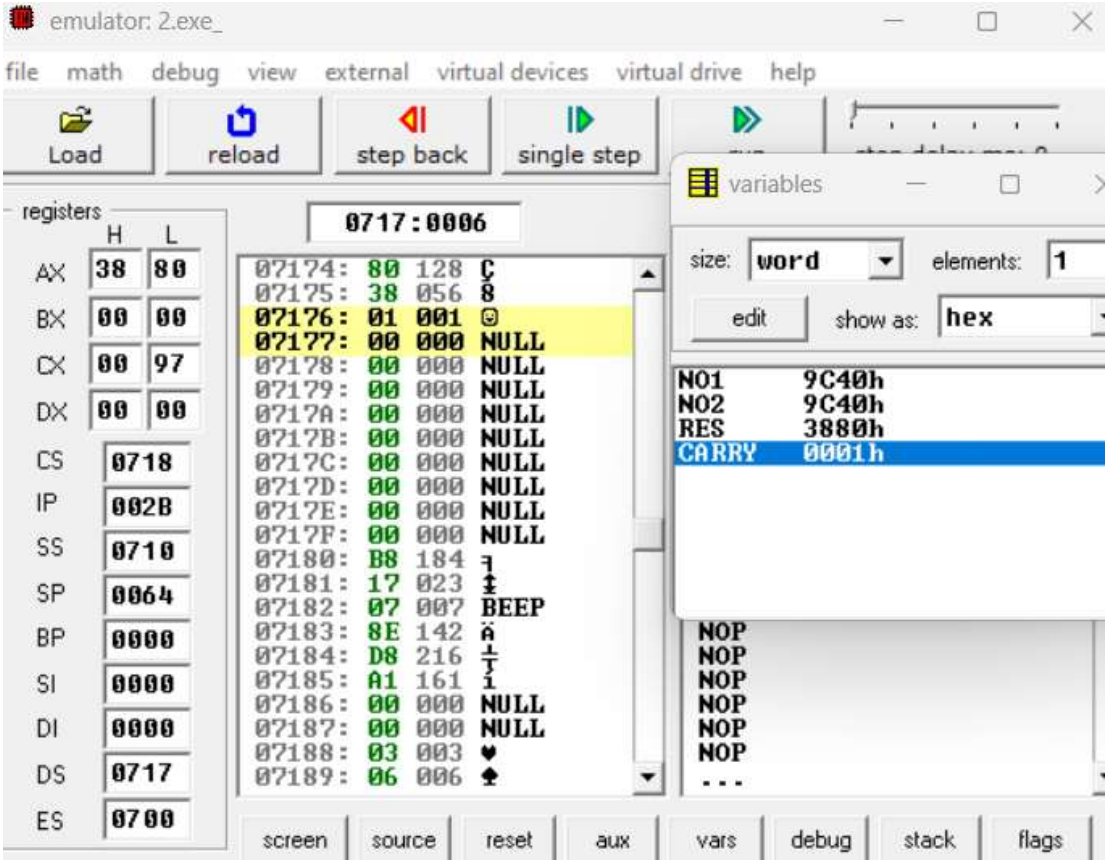
Flowchart –



Code –

```
.model small
.stack 100

.data
    no1 dw 40000   ;
    no2 dw 40000   ;
    res dw ?     ;
    carry dw 0   ;

.code
START:
    MOV ax,@data ;
    Mov ds,ax;
    mov ax,no1;
    add ax,no2;
    mov res,ax;
    jnc set;
    mov carry,1;
    set:
    end START;
```

Output –


emulator: 2.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
registers
H L
AX 38 80
BX 00 00
CX 00 97
DX 00 00
CS 0718
IP 002B
SS 0710
SP 0064
BP 0000
SI 0000
DI 0000
DS 0717
ES 0700
0717:0006
07174: 80 128 Ç
07175: 38 056 8
07176: 01 001 ☺
07177: 00 000 NULL
07178: 00 000 NULL
07179: 00 000 NULL
0717A: 00 000 NULL
0717B: 00 000 NULL
0717C: 00 000 NULL
0717D: 00 000 NULL
0717E: 00 000 NULL
0717F: 00 000 NULL
07180: B8 184 ╕
07181: 17 023 ↨
07182: 07 007 BEEP
07183: 8E 142 Ä
07184: D8 216 ╪
07185: A1 161 í
07186: 00 000 NULL
07187: 00 000 NULL
07188: 03 003 ♥
07189: 06 006 ♠
variables
size: word
elements: 1
edit
show as: hex
NO1 9C40h
NO2 9C40h
RES 3880h
CARRY 0001h
NOP
NOP
NOP
NOP
NOP
NOP
...
screen source reset aux vars debug stack flags

Practical 3

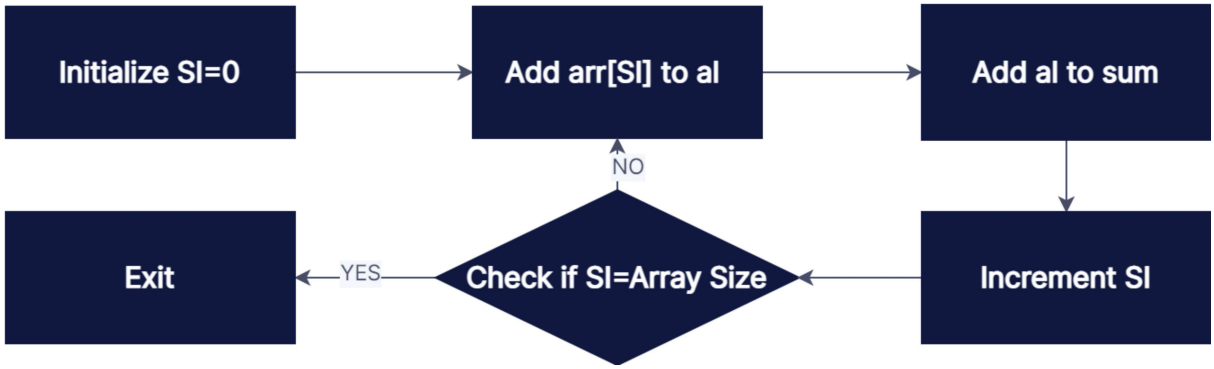
Aim -

Write a program in emu8086 to perform addition on elements of an array.

Software Required –

Emu8086

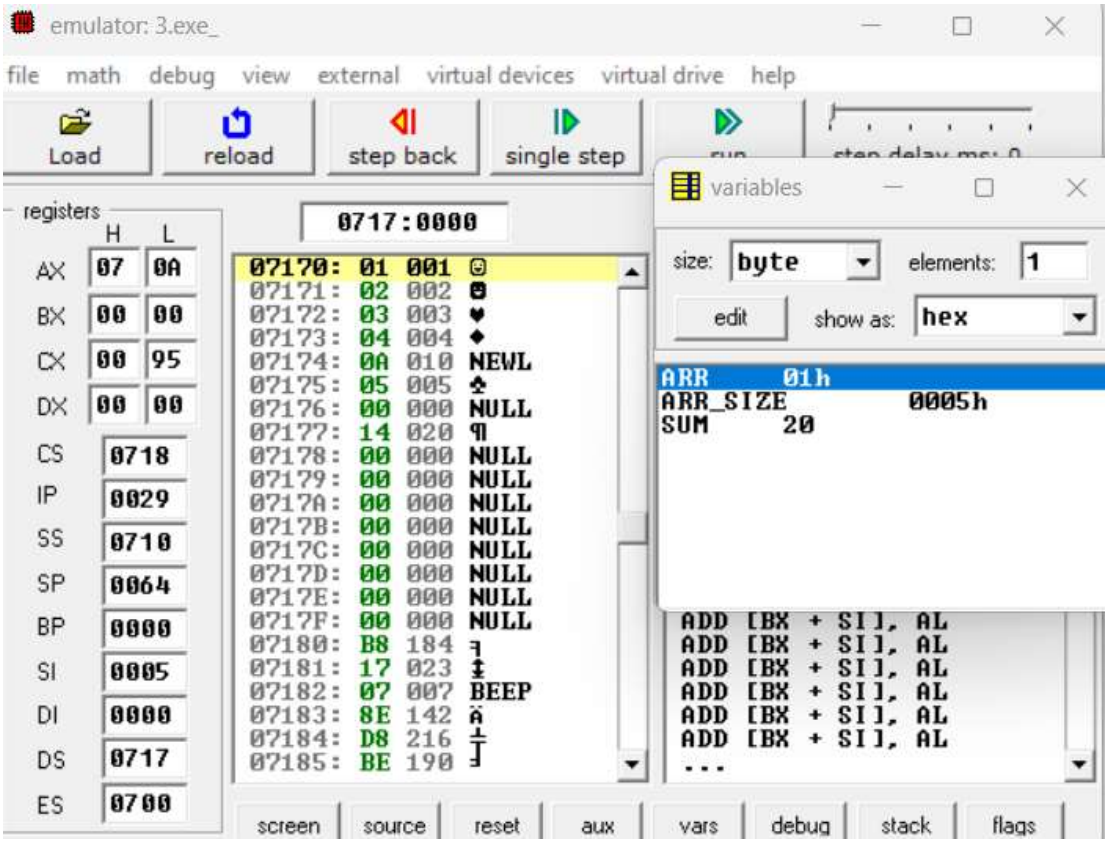
Flowchart –



Code –

```
.model small
.stack 100

.data
    arr db 1, 2, 3, 4, 10;
    arr_size dw 5;
    sum db 0;

.code
START:
    MOV ax, @data
    MOV ds, ax;
    MOV si, 0;
sum_loop:
    MOV al, arr[si]
    ADD sum, al
    INC si
    cmp arr_size,si;
    jne sum_loop
end START
```

Output –


emulator: 3.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
registers
H L
AX 07 0A
BX 00 00
CX 00 95
DX 00 00
CS 0718
IP 0029
SS 0710
SP 0064
BP 0000
SI 0005
DI 0000
DS 0717
ES 0700
0717:0000
07170: 01 001
07171: 02 002
07172: 03 003
07173: 04 004
07174: 0A 010 NEWL
07175: 05 005
07176: 00 000 NULL
07177: 14 020
07178: 00 000 NULL
07179: 00 000 NULL
0717A: 00 000 NULL
0717B: 00 000 NULL
0717C: 00 000 NULL
0717D: 00 000 NULL
0717E: 00 000 NULL
0717F: 00 000 NULL
07180: B8 184
07181: 17 023
07182: 07 007 BEEP
07183: 8E 142
07184: D8 216
07185: BE 190
variables
size: byte
elements: 1
edit
show as: hex
ARR 01h
ARR_SIZE 0005h
SUM 20
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
screen source reset aux vars debug stack flags

Practical 4

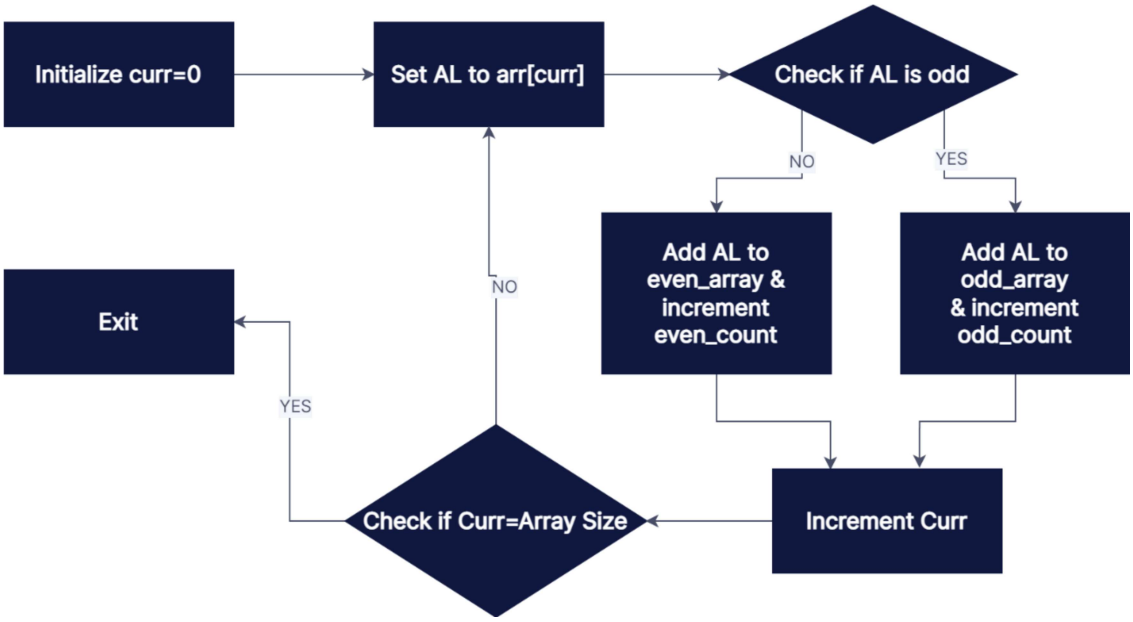
Aim -

Write a program in emu8086 to separate odd & even numbers from an array.

Software Required –

Emu8086

Flowchart –

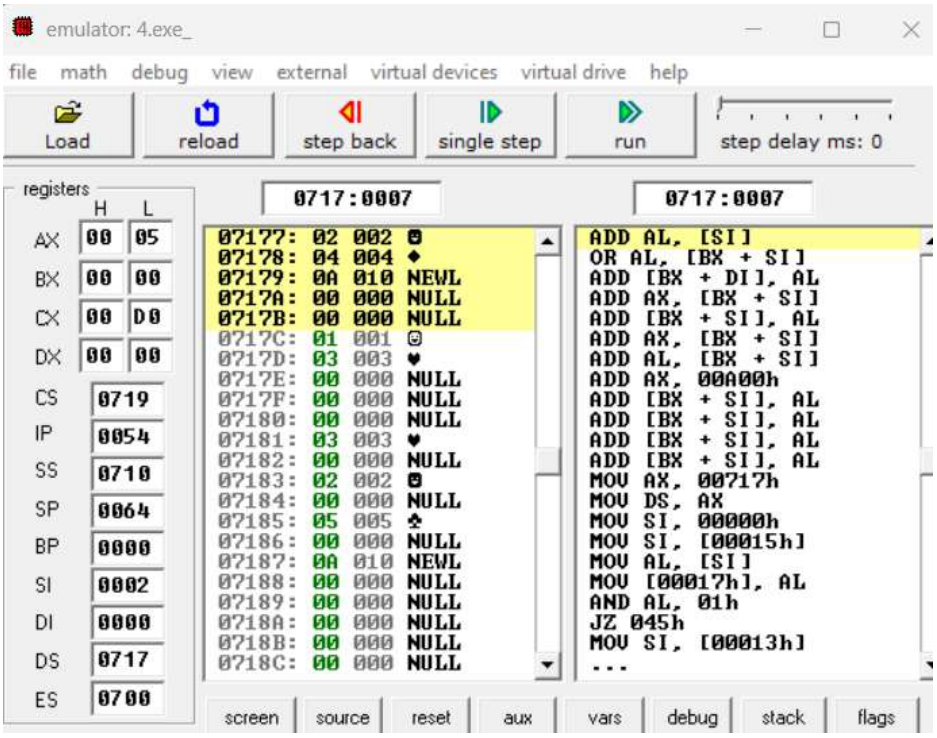


Code –

```
.model small
.stack 100
.data
    arr db 1, 2, 3, 4, 10
    arr_size dw 5
    even_arr db 5 dup(?)
    odd_arr db 5 dup(?)
    even_count dw 0
    odd_count dw 0
    curr dw 0;
    present db 0;
.code
START:
    MOV ax, @data;
    MOV ds, ax;
```

```
    MOV si, 0;
sort_loop:
    mov si,curr;
    MOV al, arr[si];
    mov present,al;
    AND al, 1;
    JZ even;
    mov si,odd_count;
    mov al,  present;
    MOV odd_arr[si], al;
    INC odd_count;
    JMP next;
even:
    mov si,even_count;
    mov al,  present;
    MOV even_arr[si], al;
    INC even_count;
next:
    INC curr;
    mov ax,curr;
    CMP ax, arr_size;
    JNE sort_loop;
end START
```

Output –



Practical 5

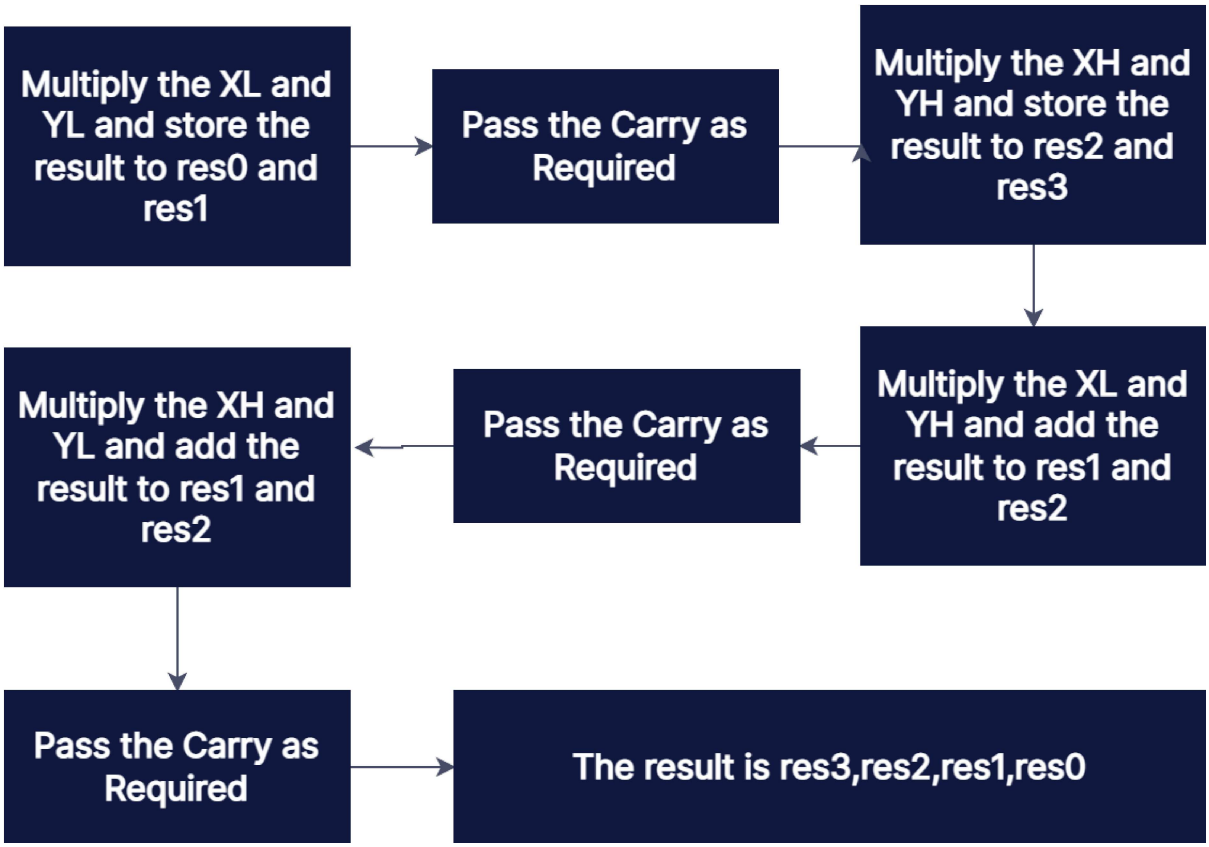
Aim -

Write a program in emu8086 to perform multiplication of two 32-Bit Numbers.

Software Required –

Emu8086

Flowchart –



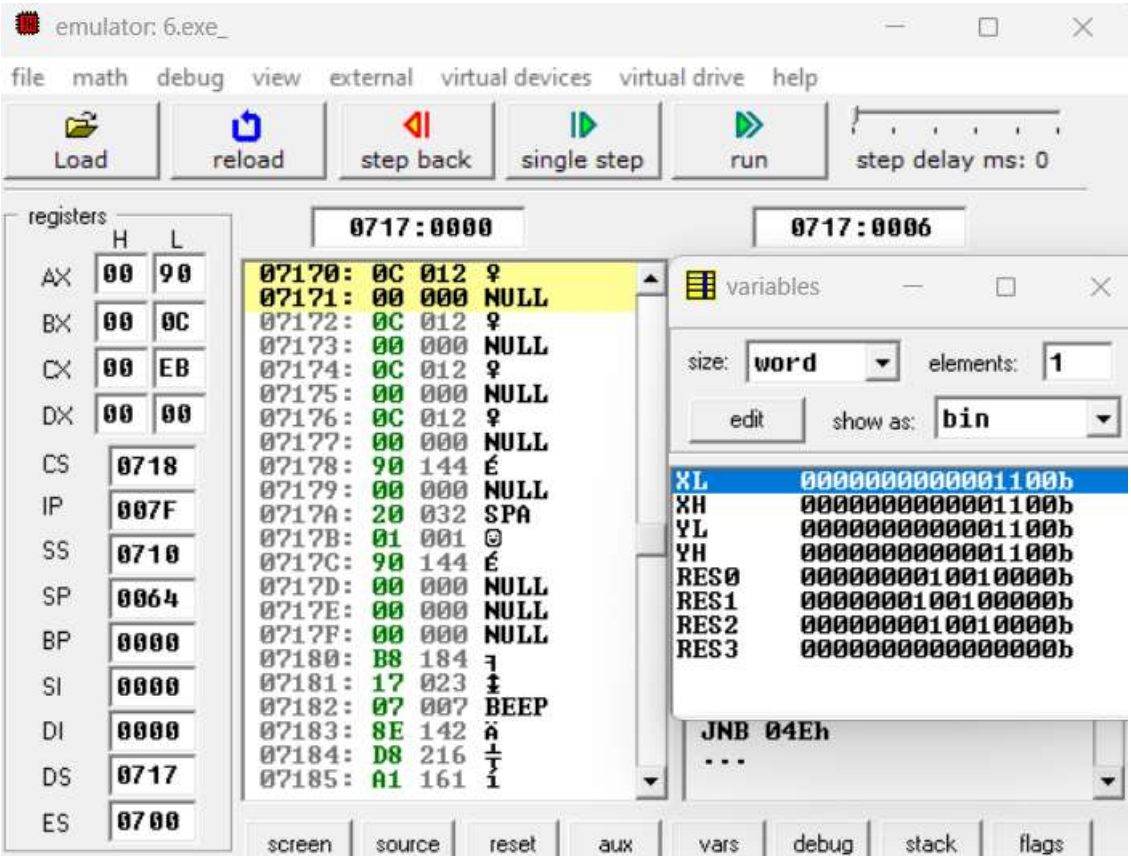
Code –

```
.model small
.stack 100

.data
    xl dw 12;
    xh dw 12;
    yl dw 12;
    yh dw 12;
    res0 dw  0;
    res1 dw  0;
    res2 dw  0;
```

```
    res3 dw  0;

.code
START:
    MOV ax, @data;
    MOV ds, ax;

    mov ax,xl;
    mov bx,yl;
    mul bx;
    mov res0,ax;
    mov res1,dx;

    mov ax,xh;
    mov bx,yh;
    mul bx;
    mov res2,ax;
    mov res3,dx;


    mov ax,xh;
    mov bx,yl;
    mul bx;
    add res1,ax;
    JNC skip1
    inc res2;
    JNC skip1
    inc res3;
skip1:
    add res2,dx;
    JNC skip2
    inc res3;
skip2:
```

```
    mov ax,xl;
    mov bx,yh;
    mul bx;
    add res1,ax;
    JNC skip3
    inc res2;
    JNC skip3
    inc res3;
skip3:
    add res2,dx;
    JNC skip4
    inc res3;
skip4:
end START
```

Output –



Practical 6

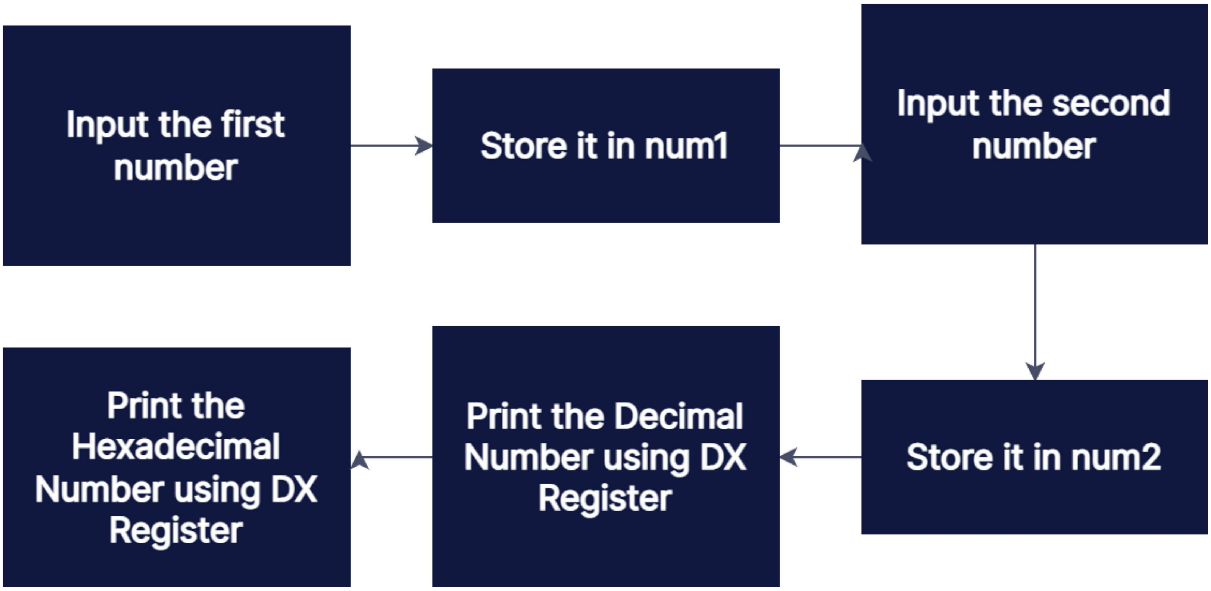
Aim -

Write a program in emu8086 for adding two 8 bits user-given data and display the output in hex form and decimal form.

Software Required –

Emu8086

Flowchart –



Code –

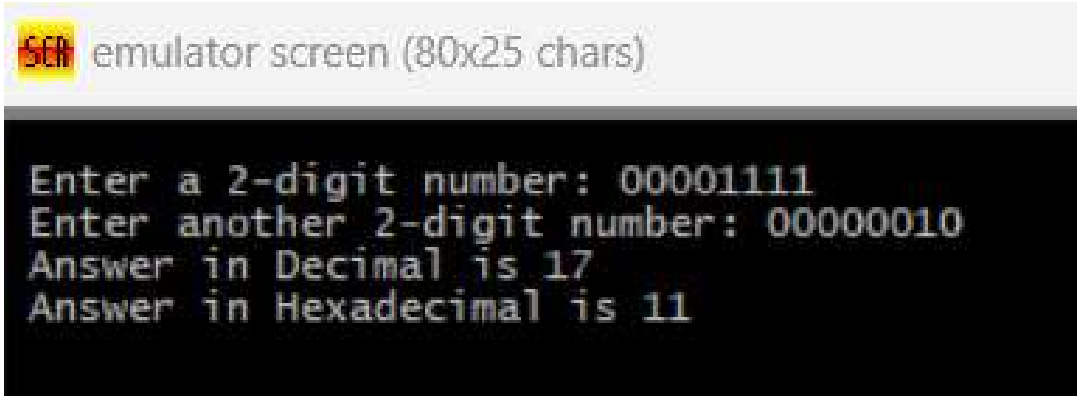
```
.model small
.data
str1 db "Enter a 2-digit number: $"
n_line db 0Ah,0Dh,"$"
str2 db "Enter another 2-digit number: $"
ans1 db "Answer in Decimal is $"
ans2 db "Answer in Hexadecimal is $"
num1 db 0
num2 db 0
.code
mov ax, @data
mov ds, ax
lea dx,n_line
mov ah, 09h
int 21h
```

```
lea dx,str1
mov ah, 09h
int 21h
mov cl, 8
mov al, 0
loop1:
mov ah, 01h
int 21h
sub al, '0'
mov bl, al
mov al,num1
mov dx,2
mul dx
add al, bl
mov num1,al
dec cl
jnz loop1
lea dx,n_line
mov ah, 09h
int 21h
lea dx,str2
mov ah, 09h
int 21h
mov cl, 8
mov al, 0
loop2:
mov ah, 01h
int 21h
sub al, '0'
mov bl, al
mov al,num2
mov dx,2
mul dx
add al, bl
```

```
mov num2,al
dec cl
jnz loop2
mov al,num1
add al,num2
mov ah,0
adc ah,0
lea dx,n_line
mov ah, 09h
int 21h
lea dx,ans1
mov ah, 09h
int 21h
mov cx,0
mov dx,0
label1:
    cmp ax,0
    je print1
    mov bx,10
    div bx
    push dx
    inc cx
    xor dx,dx
    jmp label1
print1:
    pop dx
    add dx,48
    mov ah,02h
    int 21h
    dec cx
    jnz print1
    mov cx,0
    mov dx,0
label2:
```

```
    cmp ax,0
    je print2
    mov bx, 16
    div bx
    push dx
    inc cx
    xor dx,dx
    jmp label2
print2:
    cmp cx,0
    je exit
    pop dx
    cmp dx, 9
    jle continue
    add dx,7
continue:
    add dx,48
    mov ah,02h
    int 21h
    dec cx
    jmp print2
```

Output –



Practical 7

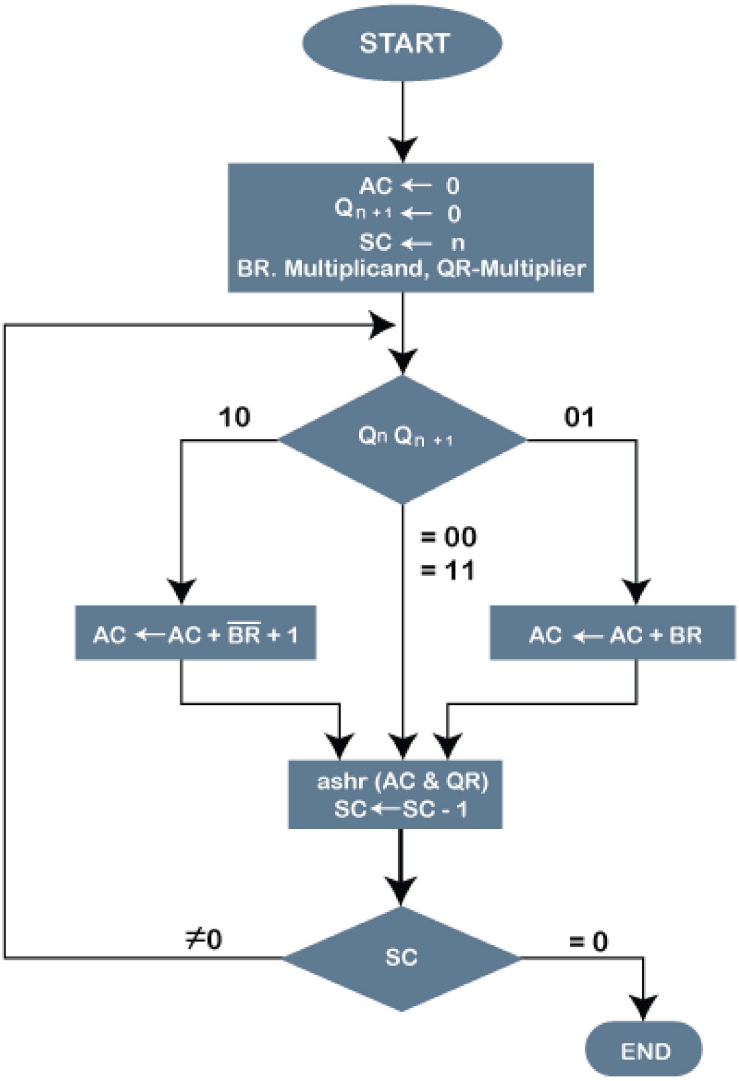
Aim -

Write a program in emu8086 to perform multiplication of two numbers using Booth Algorithm.

Software Required –

Emu8086

Flowchart –



Code –

```
.model small
.stack 100
.data
msg1 db 10, 13,          "ENTER THE MULTIPLICAND :$"
msg2 db 10, 13,          "ENTER THE MULTIPLIER :$"
result db                "THE RESULT OF MULTIPLICATION IS :$"
```

```
newline db 10,13,        "$"
m db 0
q db 0
a db 0
q_1 db 0
cnt db 8
sign_flag db 0
.code
mov ax, @data
mov ds, ax
print newline
print msg1
        call get_data
        cmp sign_flag, 01
        jnz post_m
        neg bl
        mov sign_flag, 0
post_m :
        mov m, bl
        print msg2
        call get_data
        cmp sign_flag, 01
        jnz post_q
        neg bl
        mov sign_flag, 0
post_q :
        mov q, bl
        call booth_algo
        mov ah, 4ch
        int 21h
        get_data proc near
        mov ah, 01h
        int 21h
        cmp al, '-'
```

```
        jne post
        mov sign_flag, 1
post :
        mov cx, 0204h
        mov bl, 0
accept :
        mov ah, 08h
        int 21h
        cmp al, 0dh
        jz complt
        cmp al, 30h
        jb accept
        cmp al, 39h
        jg accept
        mov dl, al
        mov ah, 02h
        int 21h
        sub al, 30h
        shl bl, cl
        add bl, al
        dec ch
        jnz accept
complt :
        call bcd2hx
ret
get_data endp
booth_algo proc near
        mov q_1, 0
        mov bh, a
        mov bl, q
        call check
        call shift
        dec cnt
        jnz go_on
```

```
        print newline
        print newline
        print result
        mov ax, bx
        and ah, 80h
        cmp ah, 00
        jz pos_ans
        neg bx
        mov dl, '-'
        mov ah, 02h
        int 21h
pos_ans :
        call disp_ans
booth_algo endp
        check proc near
        mov cl, bl
        and cl, 01
        cmp cl, q_1
        je skip
        jg subt
        add bh, m
        jmp skip
subt :
        sub bh, m
skip :
        ret
check endp
shift proc near
        mov cx, bx
        and cx, 01
        mov q_1, cl
        sar bx, 1
        ret
shift endp
```

```
disp_ans proc near
        call hex2bcd
        mov cx, 0404h
dispx :
        rol bx, cl
        mov dl, bl
        and dl, 0fh
        add dl, 30h
        mov ah, 02h
        int 21h
        dec ch
        jnz dispx
        ret
disp_ans endp
hex2bcd proc near
        mov cx, 00
thousd :
        cmp bx, 1000
        jb hund
        sub bx, 1000
        add cx, 1000h
        jmp thousd
hund :
        cmp bx, 100
        jb tens
        sub bx, 100
        add cx, 100h
        jmp hund
tens :
        cmp bx, 10
        jb unit
        sub bx, 10
        add cx,10h
        jmp tens
```

```
unit :
        add cx, bx
        mov bx, cx
        ret
hex2bcd endp
bcd2hx proc near
        mov cl, 00
tens1 :
        cmp bl, 10h
        jb unit1
        sub bl, 10h
        add cl, 10
        jmp tens1
unit1 :
        add cl, bl
        mov bl, cl
        ret
bcd2hx endp
end
```

Output –

Practical 8

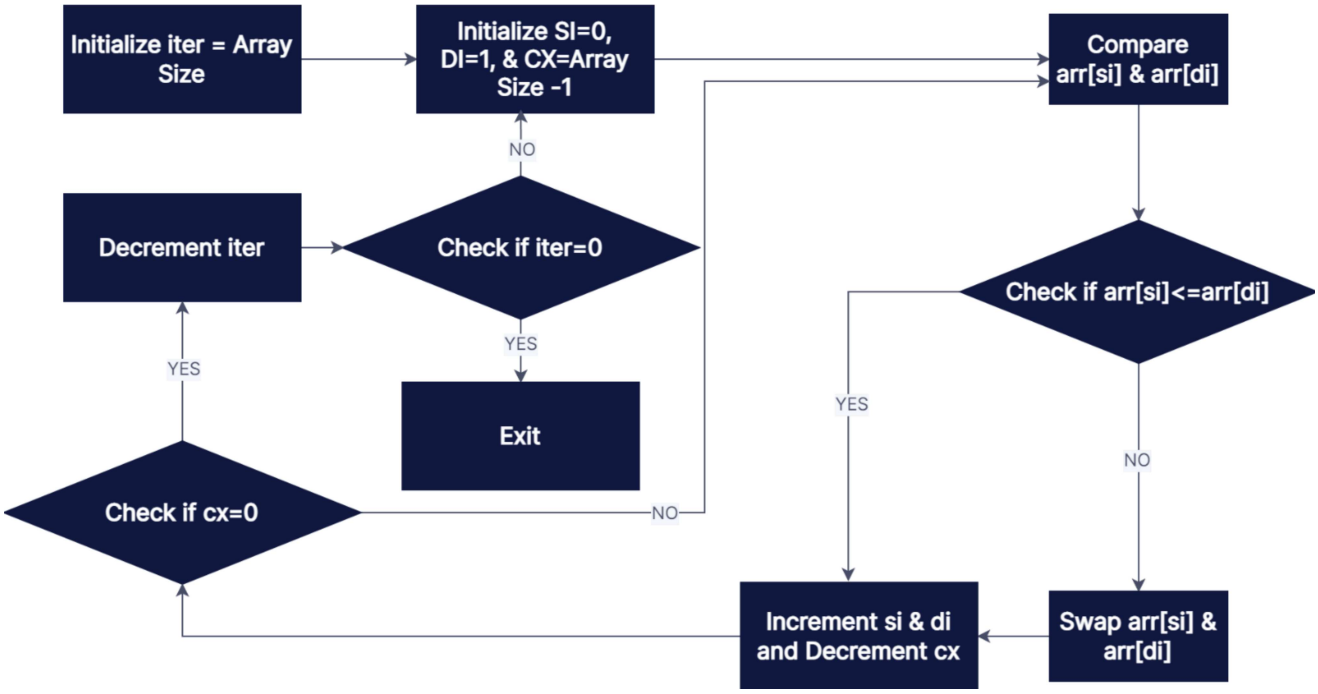
Aim -

Write a program in emu8086 to arrange 10 Numbers in ascending order.

Software Required –

Emu8086

Flowchart –

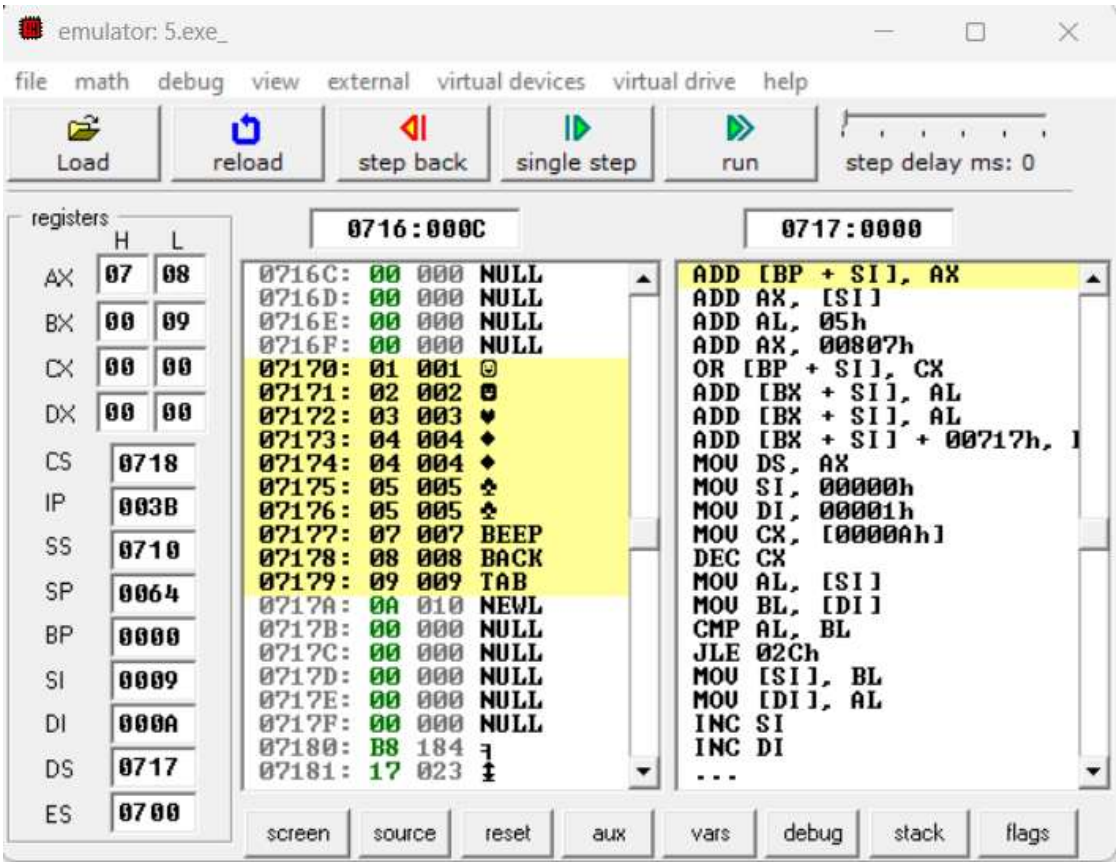


Code –

```
.model small
.stack 100

.data
    arr db 7, 3, 9, 1, 5, 8, 4, 5, 4, 2
    arr_size dw 10
    iter dw 9;
    temp dw 0;

.code
START:
    MOV ax, @data
    MOV ds, ax


sort_loop:
    MOV si, 0
    MOV di, 1
    mov cx,arr_size;
    dec cx;
    inner:
        mov al,arr[si];
```

```
        mov bl,arr[di];
        cmp al,bl;
        JLE noswap;
        mov arr[si],bl;
        mov arr[di],al;
        noswap:
        inc si;
        inc di;
        dec cx;
        JNZ inner;
    dec iter
    JNE sort_loop

end START
```

Output –



Practical 9

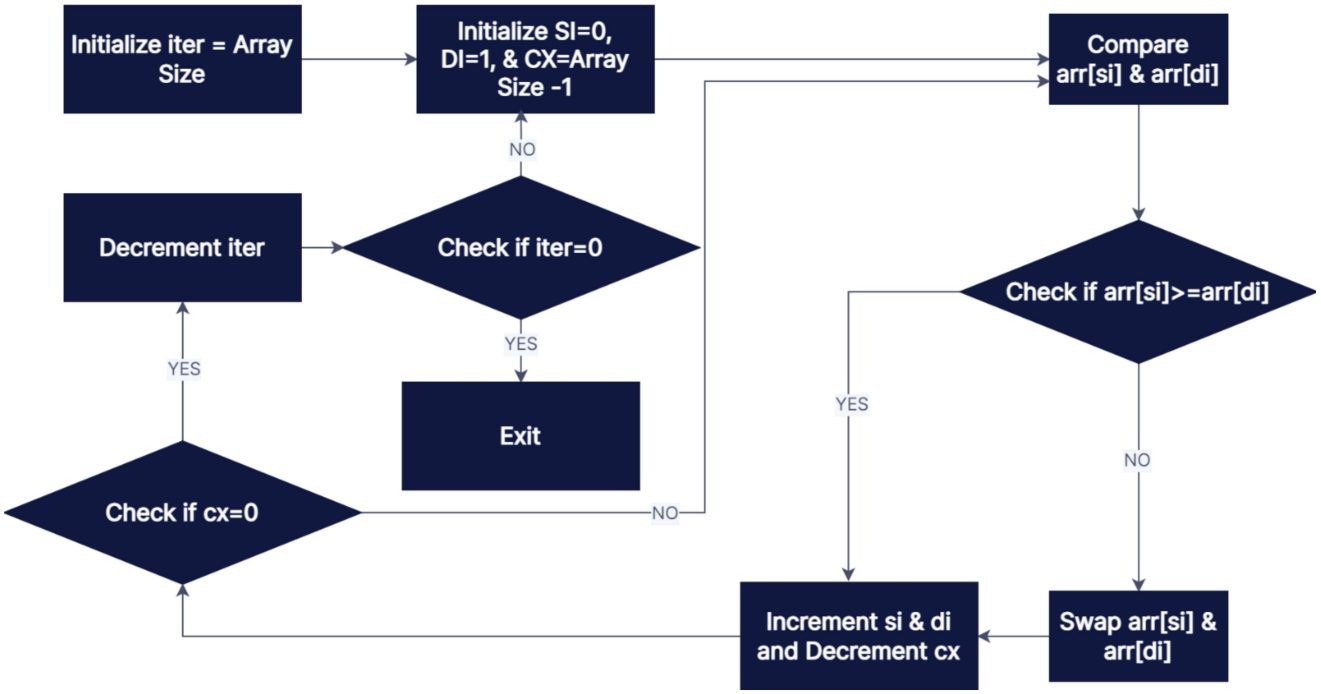
Aim -

Write a program in emu8086 to arrange 10 Numbers in descending order.

Software Required –

Emu8086

Flowchart –

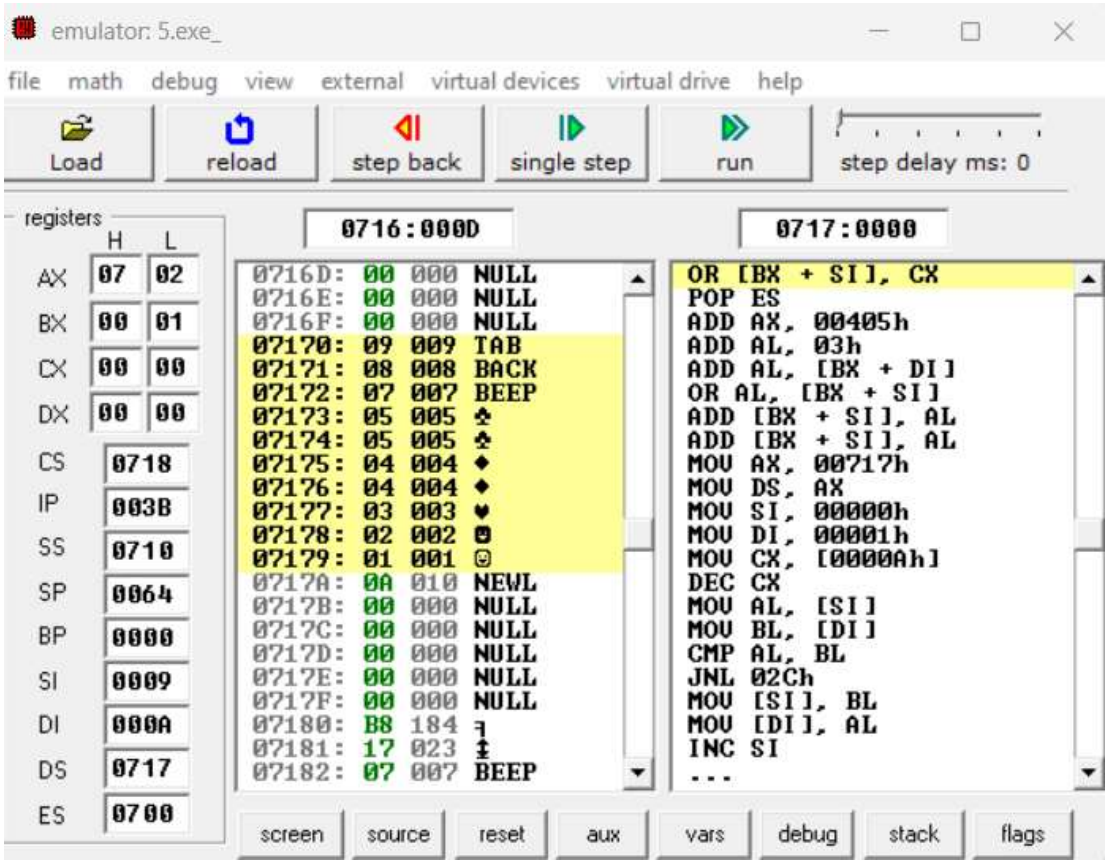


Code –

```
.model small
.stack 100

.data
    arr db 7, 3, 9, 1, 5, 8, 4, 5, 4, 2
    arr_size dw 10
    iter dw 9;
    temp dw 0;

.code
START:
    MOV ax, @data
    MOV ds, ax


sort_loop:
    MOV si, 0
    MOV di, 1
    mov cx,arr_size;
    dec cx;
    inner:
        mov al,arr[si];
```

```
        mov bl,arr[di];
        cmp al,bl;
        JGE noswap;
        mov arr[si],bl;
        mov arr[di],al;
        noswap:
        inc si;
        inc di;
        dec cx;
        JNZ inner;
    dec iter
    JNE sort_loop

end START
```

Output –



Practical 10

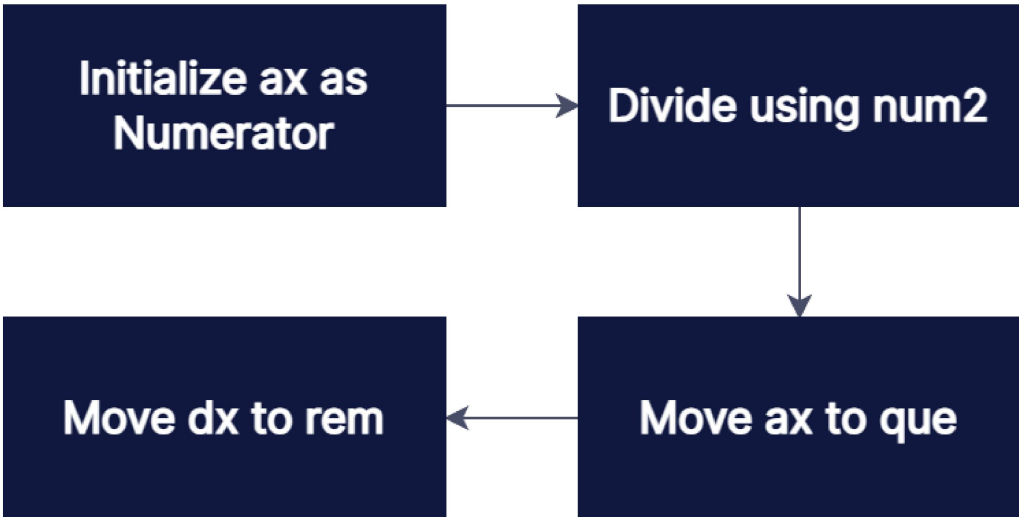
Aim -

Write a program in emu8086 to divide two Numbers.

Software Required –

Emu8086

Flowchart –



Code –

```
ORG 100h
.MODEL SMALL
.DATA
NUM_1 DW 0F213H
NUM_2 DW 41A8H
que DW ?
rem DW ?
.CODE
MOV AX, NUM_1
DIV NUM_2
mov que,ax;
mov rem,dx;
RET
```